NebulaGraph

Nebula Graph Database Manual

v2.0.1

Min Wu, Yao Zhou, Cooper Liang

Table of contents

1. Welcome to Nebula Graph 2.0.1 Documentation

This manual is revised on 2021-8-30, with GitHub commit 860563eb3.

Nebula Graph is a distributed, scalable, and lightning-fast graph database.

It is the optimal solution in the world capable of hosting graphs with dozens of billions of vertices (nodes) and trillions of edges (relationships) with millisecond latency.

1.1 Tutorial Video

- YouTube

1.2 Getting started

- What is Nebula Graph
- Quick start workflow
- FAQ
- Links

Last update: August 5, 2021

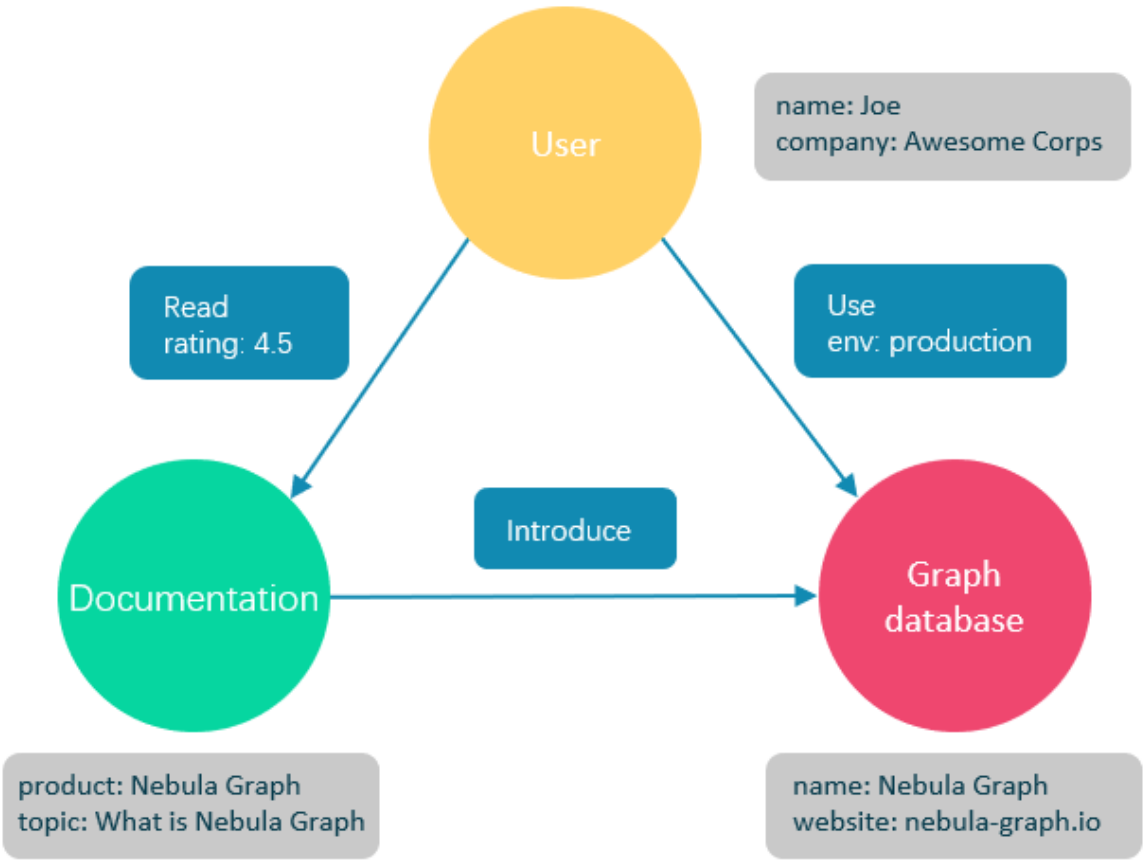
2. Introduction

2.1 What is Nebula Graph

Nebula Graph is an open-source, distributed, easily scalable, and native graph database. It is capable of hosting graphs with hundreds of billions of vertices and trillions of edges, and serving queries with millisecond-latency.

2.1.1 What is a graph database

A graph database, such as Nebula Graph, is a database that specializes in storing vast graph networks and retrieving information from them. It efficiently stores data as vertices (nodes) and edges (relationships) in labeled property graphs. Properties can be attached to both vertices and edges. Each vertex can have one or multiple tags (labels).



Graph databases are well suited for storing most kinds of data models abstracted from reality. Things are connected in almost all fields in the world. Modeling systems like relational databases extract the relationships between entities and squeeze them into table columns alone, with their types and properties stored in other columns or even other tables. This makes the data management time-consuming and cost-ineffective. Nebula Graph, as a typical native graph database, allows you to store the rich relationships as edges with edge types and properties directly attached to them.

2.1.2 Benefits of Nebula Graph

Open-source

Nebula Graph is open under the Apache 2.0 and the Commons Clause 1.0 licenses. More and more people such as database developers, data scientists, security experts, and algorithm engineers are participating in the designing and development of Nebula Graph. To join the opening of source code and ideas, surf the Nebula Graph GitHub page.

Outstanding performance

Written in C++ and born for graph, Nebula Graph handles graph queries in milliseconds. Among most databases, Nebula Graph shows superior performance in providing graph data services. The larger the data size, the greater the superiority of Nebula Graph. For more information, see Nebula Graph benchmarking.

Developer friendly

Nebula Graph supports clients in popular programming languages like Java, Python, C++, and Go, and more are being developed. For more information, see Nebula Graph clients.

Diversified ecosystem

More and more native tools of Nebula Graph have been released, such as Nebula Graph Studio, Nebula Console, and Nebula Exchange. Besides, Nebula Graph has the ability to be integrated with many cutting-edge technologies, such as Spark, Flink, and HBase, for the purpose of mutual strengthening in a world of increasing challenges and chances. For more information, see Ecosystem development.

OpenCypher-compatible query language

The native Nebula Graph Query Language, also known as nGQL, is a declarative, openCypher-compatible textual query language. It is easy to understand and easy to use. For more information, see nGQL guide.

Easy data modeling and high flexibility

You can easily model the connected data into Nebula Graph for your business without forcing them into a structure such as a relational table, and properties can be added, updated, and deleted freely. For more information, see Data modeling.

Reliable access control

Nebula Graph supports strict role-based access control and external authentication servers such as LDAP (Lightweight Directory Access Protocol) servers to enhance data security. For more information, see Authentication and authorization.

High scalability

Nebula Graph is designed in a shared-nothing architecture and supports scaling in and out without interrupting the database service.

High popularity

Nebula Graph is being used by tech leaders such as Tencent, Vivo, Meituan, and JD Digits. For more information, visit the Nebula Graph official website.

2.1.3 Use cases

Nebula Graph can be used to support various graph-based scenarios. To spare the time spent on pushing the kinds of data mentioned in this section into relational databases and on bothering with join queries, use Nebula Graph.

Fraud detection

Financial institutions have to traverse countless transactions to piece together potential crimes and understand how combinations of transactions and devices might be related to a single fraud scheme. This kind of scenario can be modeled in graphs, and with the help of Nebula Graph, fraud rings and other sophisticated scams can be easily detected.

Real-time recommendation

Nebula Graph offers the ability to instantly process the real-time information produced by a visitor and make accurate recommendations on articles, videos, products, and services.

Intelligent question-answer system

Natural languages can be transformed into knowledge graphs and stored in Nebula Graph. A question organized in a natural language can be resolved by a semantic parser in an intelligent question-answer system and re-organized. Then, possible answers to the question can be retrieved from the knowledge graph and provided to the one who asked the question.

Social networking

Information on people and their relationships are typical graph data. Nebula Graph can easily handle the social networking information of billions of people and trillions of relationships, and provide lightning-fast queries for friend recommendations and job promotions in the case of massive concurrency.

Last update: May 10, 2021

2.2 Data modeling

A data model is a model that organizes data and specifies how they are related to one another. This topic describes the Nebula Graph data model and provides suggestions for data modeling with Nebula Graph.

2.2.1 Data structures

Nebula Graph data model uses five data structures to store data. They are vertices, edges, properties, tags, and edge types.

- **Vertices**: Vertices are used to store entities.
  - In Nebula Graph, vertices are identified with vertex identifiers (i.e. `VID`). The `VID` must be unique in the same graph space.
  - A vertex must have at least one tag.
- **Edges**: Edges are used to connect vertices. An edge is a connection or behavior between two vertices.
  - An edge is identified uniquely with a source vertex, an edge type, a rank value, and a destination vertex.
  - Edges are directed. `->` identifies the directions of edges. Edges can be traversed in either direction.
  - An edge must have one and only one edge type.
  - The rank value is an immutable user-assigned 64-bit signed integer. It identifies the edges with the same edge type between two vertices. Edges are sorted by their rank values. The edge with the greatest rank value is listed first. The default rank value is zero.
- **Properties**: Properties are key-value pairs. Both vertices and edges are containers for properties.
- **Tags**: Tags are used to categorize vertices. Vertices that have the same tag share the same definition of properties.
- **Edge types**: Edge types are used to categorize edges. Edges that have the same edge type share the same definition of properties.

2.2.2 Directed property graph

Nebula Graph stores data in directed property graphs. A directed property graph has a set of vertices connected by edges. And the edges have directions. A directed property graph is represented as:

$$\mathbf{G = < V, E, P_V, P_E >}$$

- **V** is a set of vertices.
- **E** is a set of directed edges.
- $\mathbf{P_V}$ is the property of vertices.
- $\mathbf{P_E}$ is the property of edges.

The following table is an example of the structure of the basketball player dataset. We have two types of vertices, that is **player** and **team**, and two types of edges, that is ***serve*** and ***like***.

Element	Name	Property name (Data type)	Description
Tag	**player**	name (string) age (int)	Represents players in the team.
Tag	**team**	name (string)	Represents the teams.
Edge type	**serve**	start_year (int) end_year (int)	Represents actions taken by players in the team. An action links a player and a team and the direction is from a player to a team.
Edge type	**like**	likeness (int)	Represents actions taken by players in the team. An action links a player and another player and the direction is from one player to the other player.

2.2.3 Graph data modeling suggestions

This section provides general suggestions for modeling data in Nebula Graph.

> **Note**
>
> The following suggestions may not apply to some special scenarios. In these cases, find help in the Nebula Graph community.

Model for performance

There is no perfect method to model in Nebula Graph. Graph modeling depends on the questions that you want to know from the data. Your data drives your graph model. Graph data modeling is intuitive and convenient. Create your data model based on your business model. Test your model and gradually optimize it to fit your business. To get better performance, you can change or re-design your model multiple times.

Edges as properties

Traversal depth decreases the traversal performance. To decrease the traversal depth, use vertex properties instead of edges.

For example, to model a graph that have the name, age, and eye color elements, you can:

- (RECOMMENDED) Create a tag `person`, then add the name, age, and eye color as its properties.
- (WRONG WAY) Create a new tag `eye color` and a new edge type `has`, then create an edge to indicate that a person has an eye color.

The first modeling solution leads to much better performance. DO NOT use the second solution unless you have to.

Multiple properties under one tag are permitted. But make sure that tags are fine-grained. For more information, see the Granulated vertices section.

Granulated vertices

In graph modeling, use the data models with a higher level of granularity. Put a set of parallel properties into one tag, i.e., separate different concepts.

Use indexes correctly

Correct use of indexes speeds up queries, but indexes reduce the write performance by 90% or more. **ONLY** use indexes when you locate vertices or edges by their properties.

No long string properties on edges

Be careful when you create long string properties for edges. Nebula Graph supports storing such properties on edges. But note that these properties are stored both in the outgoing edges and the incoming edges. Thus be careful with the write amplification.

Last update: April 22, 2021

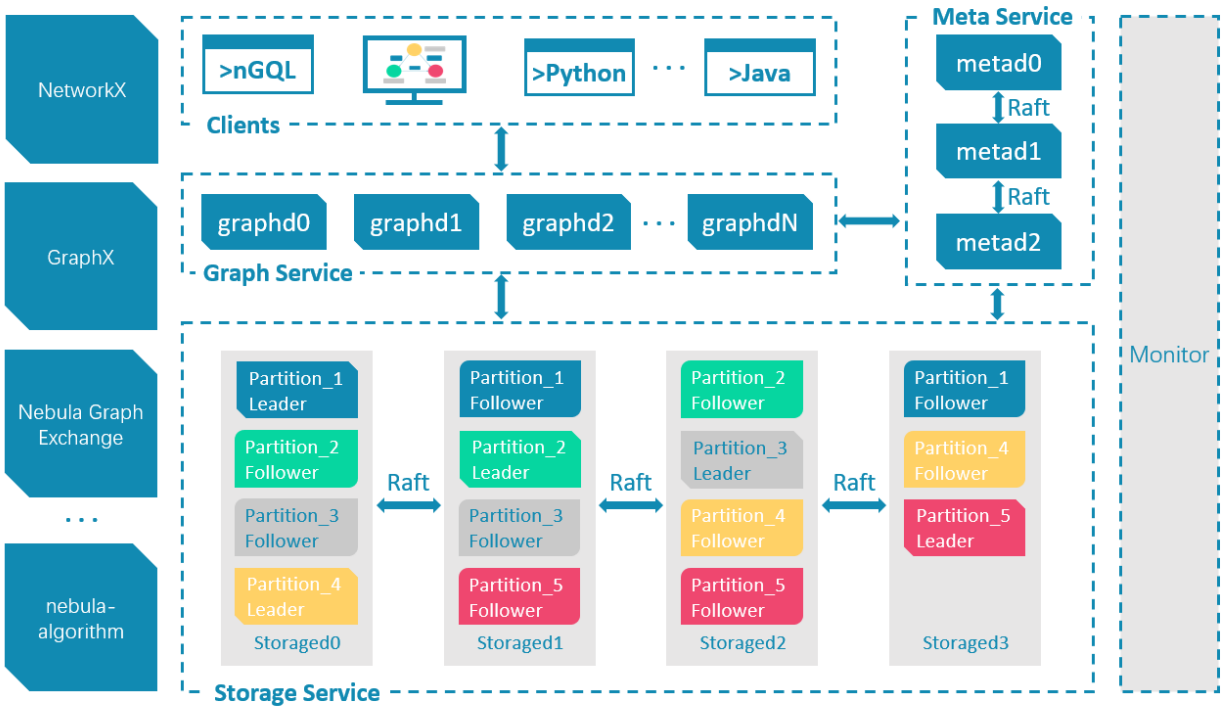
2.3 Nebula Graph architecture

2.3.1 Architecture overview

Nebula Graph consists of three services: the Graph Service, the Storage Service, and the Meta Service.

Each service has its executable binaries and processes launched from the binaries. You can deploy a Nebula Graph cluster on a single machine or multiple machines using these binaries.

The following figure shows the architecture of a typical Nebula Graph cluster.



The Meta Service

The Meta Service in the Nebula Graph architecture is run by the nebula-metad processes. It is responsible for metadata management, such as schema operations, cluster administration, and user privilege management.

For details on the Meta Service, see Meta Service.

The Graph Service and the Storage Service

Nebula Graph applies a disaggregated storage and compute architecture. The Graph Service is responsible for querying. The Storage Service is responsible for storage. And they run on different processes, i.e., nebula-graphd and nebula-storaged. The benefits of disaggregated storage and compute are as follows:

- Great scalability. A disaggregated structure makes both the Graph Service and the Storage Service flexible and easy to scale in or out.
- High availability. If part of the Graph Service fails, the data stored by the Storage Service suffers no loss. And if the rest part of the Graph Service is still able to serve the clients, service recovery can be performed quickly, or even unfelt by the users.
- Cost-effective. The separation of computing and storage provides a higher resource utilization rate, and it enables you to manage the cost flexibly according to business demands. The cost savings can be more significant if you use the Nebula Graph Cloud service.
- Open to more possibilities. With the ability to run separately, the Graph Service may work with multiple types of storage engines, and the Storage Service may serve more types of computing engines.

For details on the Graph Service and the Storage Service, see Graph Service and Storage Service.

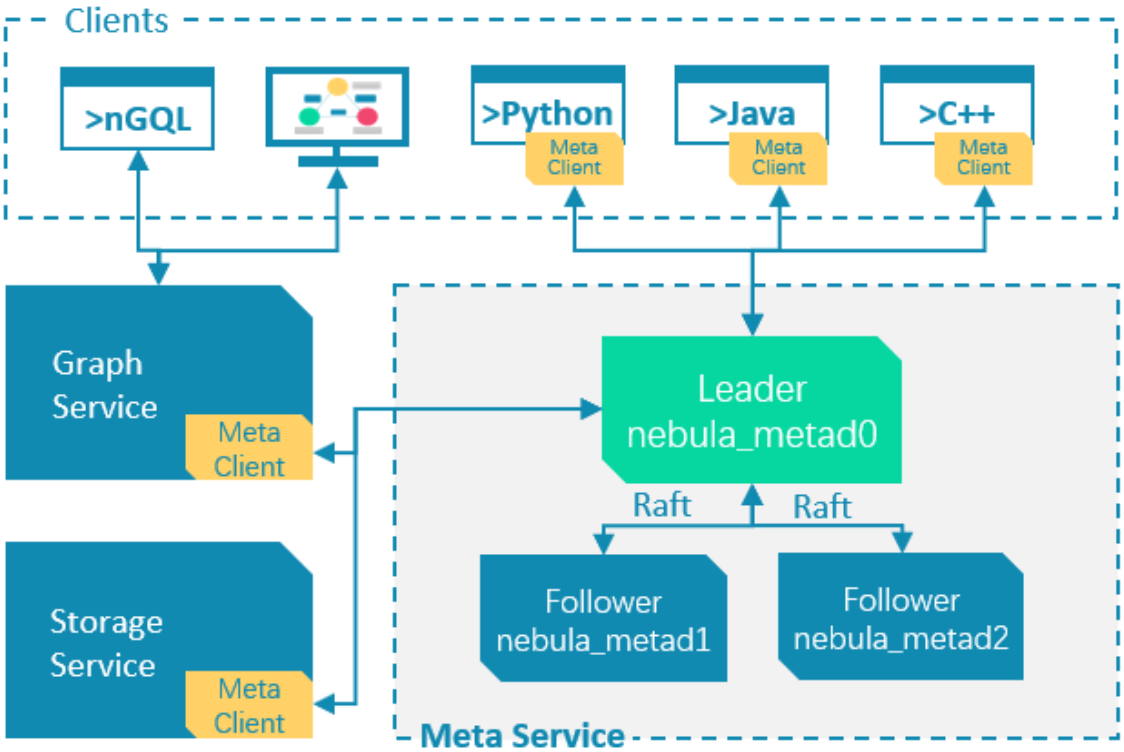
Last update: March 25, 2021

2.3.2 Meta Service

This topic describes the architecture and functions of the Meta Service.

The architecture of the Meta Service

The architecture of the Meta Service is as follows.



The Meta Service is run by the nebula-metad processes. You can deploy nebula-metad processes according to the scenario:

- In a test environment, you can deploy one or three nebula-metad processes on different machines or a single machine.
- In a production environment, we recommend that you deploy three processes on different machines for high availability.

All the nebula-metad processes form a Raft-based cluster, with one process as the leader and the others as the followers. The leader is elected by quorum, and only the leader can provide service to the clients and other components of Nebula Graph. The followers run in a standby way and each has a data replication of the leader. Once the leader fails, one of the followers will be elected as the new leader.

Functions of the Meta Service

MANAGES USER ACCOUNTS

The Meta Service stores the information of user accounts and the privileges granted to the accounts. When the clients send queries to the Graph Service through an account, the Graph Service checks the account information and whether the account has the right privileges to execute the queries or not.

For more information on Nebula Graph access control, see Authentication and authorization.

MANAGES PARTITIONS

The Meta Service stores and manages the locations of the storage partitions and helps balance the partitions.

MANAGES GRAPH SPACES

Nebula Graph supports multiple graph spaces. Data stored in different graph spaces are securely isolated. The Meta Service stores the metadata of all graph spaces and tracks the changes of them, such as adding or dropping a graph space.

MANAGES SCHEMA INFORMATION

Nebula Graph is a strong-typed graph database. Its schema contains tags (i.e., the vertex types), edge types, tag properties, and edge type properties.

The Meta Service stores the schema information. Besides, it performs the addition, modification, and deletion of the schema, and logs the versions of them.

For more information on Nebula Graph schema, see Data model.

MANAGES TTL-BASED DATA EVICTION

The Meta Service provides automatic data eviction and space reclamation based on TTL (time to live) options for Nebula Graph.

For more information on TTL, see TTL options.

MANAGES JOBS

The Job Manager module in the Meta Service is responsible for the creation, queuing, querying and deletion of jobs.

Last update: March 19, 2021

2.3.3 Graph Service

> **Note**
>
> Writing this topic is listed in the training plan for the next Nebula Graph Technical Writer. If you want to learn about the Graph Service, see An Introduction to Nebula Graph 2.0 Query Engine for now.

Last update: April 22, 2021

2.3.4 Storage Service

> **Note**
>
> We are using this topic in recruitment tests. So the official version of it won't be released until the end of April. Feel free to contact us if you want to join the team. You may also contribute to this topic if interested.
>
> References:
>
> - An Introduction to Nebula Graph's Storage Engine
> - Architecture overview
> - Meta Service

Last update: April 22, 2021

3. Quick start

3.1 FAQ

This topic lists the frequently asked questions for using Nebula Graph. You can use the search box in the help center or the search function of the browser to match the questions you are looking for.

If the solutions described in this topic cannot solve the problem, ask for help on the Nebula Graph forum or submit an issue on GitHub.

3.1.1 About manual updates

Why is the behavior of manual not consistent with the system?

Nebula Graph is still under development. Its behavior changes from time to time. Please tell us if the manual and the system are not consistent.

Some errors in this manula

1. Click the `pencil` button at the top right side of this page.
2. Use markdown to fix this error Then "Commit changes" at the bottom, which will start a Github pull request.
3. Sign the CLA. This pull request (and the fix) will be merged after to reviewer's accept.

3.1.2 About forward and backward compatibility

Major version compatibility

Neubla Graph 2.0.1 is not compatible with Nebula Graph 1.x nor 2.0-RC in both data formats and RPC-protocols, and vice versa. Check how to upgrade to Neubla Graph 2.0.1. You must upgrade all clients.

Micro version compatibility

Neubla Graph 2.0.1 is compatible with Nebula Graph in both data formats and RPC-protocols.

3.1.3 About openCypher compatibility

Is nGQL compatible with openCypher 9?

nGQL is partially compatible with openCypher 9. Known incompatible items are listed in Nebula Graph Issues. Submit an issue with the `incompatible` tag if you find a new issue of this type. You can search in this manual with the keyword `compatibility` to find major compatibility issues.

The following are some major differences (by design incompatible) between nGQL and openCypher.

openCypher 9	nGQL
schema optional	strong schema
equality operator '='	equality operator '=='
math exponentiation `^`	`^` not supported. Use pow(x, y) instead.
no such concept	edge rank (reference by @)
all DMLs (`CREATE`, `MERGE`, etc), and `OPTIONAL MATCH` are not supported.	

OpenCypher 9 and Cypher have some differences (in grammar and licence). For example, Cypher requires that **All Cypher statements are explicitly run within a transaction**. While openCypher has no such requirement of `transaction`. And nGQL does not support transaction.

Where can I find more nGQL examples?

Find more than **2500** nGQL examples in the features directory on the Nebula Graph GitHub page.

The `features` directory consists of `.feature` files. Each file records scenarios that you can use as nGQL examples.

Here is an example:

```
Feature: Match seek by tag

  Background: Prepare space
    Given a graph with space named "basketballplayer"

  Scenario: seek by empty tag index
    When executing query:
      """
      MATCH (v:bachelor)
      RETURN id(v) AS vid
      """
    Then the result should be, in any order:
      | vid          |
      | 'Tim Duncan' |
    And no side effects
    When executing query:
      """
      MATCH (v:bachelor)
      RETURN id(v) AS vid, v.age AS age
      """
    Then the result should be, in any order:
      | vid          | age |
      | 'Tim Duncan' | 42  |
    And no side effects
```

The keywords in the preceding example are described as follows:

Keyword	Description
`Feature`	Describes the topic of the current `.feature` file.
`Background`	Describes the background information of the current `.feature` file.
`Given`	Describes the prerequisites of running the test statements in the current `.feature` file.
`Scenario`	Describes the purpose of the scenario. If there is the `@skip` before `Scenario`, this scenario may not work and don't use it as a working example.
`When`	Describes the nGQL statement to be executed.
`Then`	Describes the expected result of running the statement in the `When` clause. If the result in your environment does not match the result described in the `.feature` file, submit an issue to inform the Nebula Graph team.
`And`	Describes the side effects of running the statement in the `When` clause.
@skip	This test case will be skipped. Commonly, the to-be-tested code is not ready.

Welcome to add more practical scenarios and become a Nebula Graph contributor.

Does it support TinkerPop Gremlin?

No. And no plan to support that.

3.1.4 About Data Model

Does Nebula Graph support W3C RDF (SPARQL), or GraphQL?

No. And no plan to support that.

Nebula Graph's data model is the property graph, and it is a strong schema system.

It doesn't support rdf.

Nebula Graph Query Language does not support `SPARQL` nor `GraphQL`.

3.1.5 About executions

How is the time spent value at the end of each return message calculated?

Take the return message of `SHOW SPACES` as an example:

```
nebula> SHOW SPACES;
+------------------+
| Name             |
+------------------+
| basketballplayer |
+------------------+
Got 1 rows (time spent 1235/1934 us)
```

- The first number 1235 shows the time spent by the database itself, that is, the time it takes for the query engine to receive a query from the client, fetch the data from the storage server and perform a series of calculations.
- The second number 1934 shows the time spent from the client's perspective, that is, the time it takes for the client from sending a request, receiving a response, and displaying the result on the screen.

Can I set replica_factor as an even number in CREATE SPACE statements, e.g., replica_factor = 2?

NO.

The Storage Service garantees its availability based on the Raft consensus protocol. The number of failed replicas must not exceed half of the total replica number.

When `replica_factor=2`, if one replica fails, the Storage Service fails. No matter `replica_factor=3` or `replica_factor=4`, if more than one replica fails, the Storage Service fails, so `replica_factor=3` is recommended.

To prevent unnecessary waste of resources, we recommend that you set an odd replica number.

We suggest that you set replica_factor to 3 for the production environment and 1 for the test environment. Do not use an even number.

How to resolve `[ERROR (-7)]: SyntaxError: syntax error near`?

In most cases, a query statement requires a `YIELD` or a `RETURN`. Check your query statement to see if `YIELD` or `RETURN` is provided.

How to count the vertices/edges number of each tag/edge type?

See show-stats.

How to get all the vertices/edge of each tag/edge type?

1. Create and rebuild the index.

```
> CREATE TAG INDEX i_player ON player();
> REBUILD TAG INDEX i_player;
```

2. Use `LOOKUP` or `MATCH`.

```
> LOOKUP ON player;
> MATCH (n:player) RETURN n;
```

See `INDEX`, `LOOKUP` and `MATCH`.

How to resolve the `can’t solve the start vids from the sentence` error?

The graphd requires `start vids` to begin a graph traversal. The `start vids` can either be specified by the user, for example,

```
> GO FROM ${vids} ...
> MATCH (src) WHERE id(src) == ${vids}
# The start vids are explicitly given by ${vids}.
```

or be found from a (property) index, for example,

```
# CREATE TAG INDEX i_player ON player(name(20));
# REBUILD TAG INDEX i_player;

> LOOKUP ON player WHERE player.name == "abc" | ... YIELD ...
> MATCH (src) WHERE src.name == "abc" ...
# The start vids are found from the property index on name.
```

Otherwise, an error like `can’t solve the start vids from the sentence` will be raised.

How to resolve error `Storage Error: The VID must be a 64-bit integer or a string.`?

Check your vid is an integer or a `fix_string(N)`. If it is a string type, make sure your input is not longer than `N` (default value is `8`). See create space.

How to resolve error `edge conflict` or `vertex conflict`?

Nebula Graph returns such errors when the Storage Service receives multiple requests to insert or update the same vertex or edge within milliseconds. Try the failed requests again later.

Storage Error E_RPC_FAILURE

Storaged returns too many data back to graphd.Possible solutions:

1. Check whether storaged is Out-of-memory. (`dmesg| grep nebula`).
2. In `nebula-graphd.conf`, modify or add the item `--storage_client_timeout_ms=60000` to change the timeout(ms).
3. Modify your nGQL to reduce full scans (including limit sentence).
4. Use better hardware (NVMe, more memory).
5. retry.

The leader has changed. Try again later

Known Issue. Just retry 1 to N times, where N is the partition number.The reason is that meta client needs some heartbeats to update or errors to trigger the new leader information.

How to stop a slow query

You can't. Even killing a client will not stop the running slow query. You have to wait the query to complete.

3.1.6 About operation and maintenance

The log files are too large. How to recycle the logs?

Nebula Graph uses glog to print logs. `glog` can't recycle the outdated files. You can use crontab to delete them by yourself. Refer to the discussions of `Glog should delete old log files automatically`.

How to check the Nebula Graph version?

1. Use the `<binary_path> --version` command to get the Git commit IDs of the Nebula Graph binary files.

   For exmaple, to check the version of the Graph Service, go to the directories where the `nebula-graphd` binary files are stored, and run `./nebula-graphd --version` as follows to get the commit IDs.

   ```
   $ ./nebula-graphd --version
   nebula-graphd version Git: ab4f683, Build Time: Mar 24 2021 02:17:30
   This source code is licensed under Apache 2.0 License, attached with Common Clause Condition 1.0.
   ```

2. Search for the commit ID obtained in the preceding step on the GitHub commits page.
3. Compare the commit time of the binary files with the release time of Nebula Graph versions to find out the version of the Nebula Graph services.

How to scale out or scale in?

Nebula Graph 2.0.1 doesn't provide any commands or tools to support automatic scale out/in. You can do by the following steps：

1. metad： metad can not be scaled out or scale in. The process can't be moved to a new machine. You can not add a new metad process to the service.
2. Scale in graphd： remove graphd from client's code. Close this graphd process.
3. Scale out graphd： prepare graphd's binary and config files in the new host. Modify the config files and add all existing metad's addresses. Then start the new graphd process.
4. Scale in storaged：All spaces' replace number must be greater than 1：ref to Balance remove command. After the command finish, stop this storaged process.
5. Scale out storaged：All spaces' replace number must be greater than 1：prepare storaged's binary and config files in the new host, Modify the config files and add all existing metad's adDresses. Then start the new storaged process.

You may also need to run Balance Data and Balance leader after scaling in/out storaged.

3.1.7 About connections

Which ports should be opened on the firewalls?

If you have not changed the predefined ports in the configurations, open the following ports for the Nebula Graph services:

Service	Ports
Meta	9559, 9560, 19559, 19560
Graph	9669, 19669, 19670
Storage	9777 ~ 9780, 19779, 19780

If you have customized the configuration files and changed the predefined ports, find the port numbers in your configuration files and open them on the firewalls.

How to test whether a port is open or closed?

You can use telnet as follows to check for port status.

```
telnet <ip> <port>
```

For example:

```
// If the port is open:
$ telnet 192.168.1.10 9669
Trying 192.168.1.10...
Connected to 192.168.1.10.
Escape character is '^]'.

// If the port is closed or blocked:
$ telnet 192.168.1.10 9777
Trying 192.168.1.10...
telnet: connect to address 192.168.1.10: Connection refused
```

If you cannot use the telnet command, check if telnet is installed or enabled on your host.

Last update: July 7, 2021

3.2 Quick start workflow

The quick start introduces the simplest workflow to using Nebula Graph, including deploying Nebula Graph, connecting to Nebula Graph, and doing basic CRUD.

1. Deploy Nebula Graph with Docker Compose
2. Connect to Nebula Graph
3. CRUD in Nebula Graph

Other frequently read topics are recommended as follows. They are not in the quick start, but you may need them as soon as you pass the quick start phase.

- Read FAQ
- Deploy a Nebula Graph cluster
- Some useful links
- Compaction

Last update: April 1, 2021

3.3 Deploy Nebula Graph with Docker Compose

There are multiple ways to deploy Nebula Graph, but using Docker Compose is usually considered to be a fast starter.

3.3.1 Reading guide

If you are reading this topic with the questions listed below, click them to jump to their answers.

- What do I need to do before deploying Nebula Graph?
- How to fast deploy Nebula Graph with Docker Compose?
- How to check the status and ports of the Nebula Graph services?
- How to check the data and logs of the Nebula Graph services?
- How to stop the Nebula Graph services?
- What are the other ways to install Nebula Graph?

3.3.2 Prerequisites

- You have installed the following applications on your host.

Application	Recommended version	Official installation reference
Docker	Latest	Install Docker Engine
Docker Compose	Latest	Install Docker Compose
Git	Latest	Download Git

- If you are deploying Nebula Graph as a non-root user, grant the user with Docker-related privileges. For a detailed instruction, see Docker document: Manage Docker as a non-root user.
- You have started the Docker service on your host.
- If you have already deployed another version of Nebula Graph with Docker Compose on your host, to avoid compatibility issues, back up the service data if you need, and delete the `nebula-docker-compose/data` directory.

> **Note**
>
> To backup the Nebula Graph data, see Use B&R to backup data. TODO: It is not released.

3.3.3 How to deploy

1. Clone the `master` branch of the `nebula-docker-compose` repository to your host with Git.

> **Danger**
>
> The `master` branch contains the Docker Compose solution for the latest Nebula Graph development release. **DON'T** use this release for production.

```
$ git clone https://github.com/vesoft-inc/nebula-docker-compose.git
```

2. Go to the `nebula-docker-compose` directory.

```
$ cd nebula-docker-compose/
```

3. Run the following command to start all the Nebula Graph services.

> **Note**
>
> Update the Nebula Graph images and Nebula Console images first if they are out of date.

```
nebula-docker-compose]$ docker-compose up -d
Creating nebula-docker-compose_metad0_1 ... done
Creating nebula-docker-compose_metad2_1 ... done
Creating nebula-docker-compose_metad1_1 ... done
Creating nebula-docker-compose_graphd2_1    ... done
Creating nebula-docker-compose_graphd_1     ... done
Creating nebula-docker-compose_graphd1_1    ... done
Creating nebula-docker-compose_storaged0_1 ... done
Creating nebula-docker-compose_storaged2_1 ... done
Creating nebula-docker-compose_storaged1_1 ... done
```

> **Note**
>
> For more information of the preceding services, see Nebula Graph architecture.

4. Connect to Nebula Graph.

   a. Run the following command to start a new docker container with the Nebula Console image, and connect the container to the network where Nebula Graph is deployed.

   ```
   $ docker run --rm -ti --network nebula-docker-compose_nebula-net \
   --entrypoint=/bin/sh vesoft/nebula-console:v2-nightly
   ```

   > **Note**
   >
   > Your local network (nebula-docker-compose_nebula-net) may be different from the example above. Use the following command.
   >
   > ```
   > $ docker network  ls
   > NETWORK ID          NAME                                DRIVER              SCOPE
   > a74c312b1d16        bridge                              bridge              local
   > dbfa82505f0e        host                                host                local
   > ed55ccf356ae        nebula-docker-compose_nebula-net    bridge              local
   > 93ba48b4b288        none                                null                local
   > ```

   b. Connect to Nebula Graph with Nebula Console.

   ```
   docker> nebula-console -u user -p password --address=graphd --port=9669
   ```

   > **Note**
   >
   > By default, the authentication is off, you can log in with any user name and password. To turn it on, see Enable authentication.

   c. Run the `SHOW HOSTS` statement to check the status of the `nebula-storaged` processes.

   ```
   nebula> SHOW HOSTS;
   +-------------+------+----------+--------------+----------------------+------------------------+
   | Host        | Port | Status   | Leader count | Leader distribution  | Partition distribution |
   +-------------+------+----------+--------------+----------------------+------------------------+
   | "storaged0" | 9779 | "ONLINE" | 0            | "No valid partition" | "No valid partition"   |
   +-------------+------+----------+--------------+----------------------+------------------------+
   | "storaged1" | 9779 | "ONLINE" | 0            | "No valid partition" | "No valid partition"   |
   +-------------+------+----------+--------------+----------------------+------------------------+
   | "storaged2" | 9779 | "ONLINE" | 0            | "No valid partition" | "No valid partition"   |
   +-------------+------+----------+--------------+----------------------+------------------------+
   | "Total"     |      |          | 0            |                      |                        |
   +-------------+------+----------+--------------+----------------------+------------------------+
   ```

5. Run `exit` twice to switch back to your terminal (shell). You can run Step 4 to login Nebula Graph again.

3.3.4 Check the Nebula Graph service status and port

Run `docker-compose ps` to list all the services of Nebula Graph and their status and ports.

```
$ docker-compose ps
nebula-docker-compose_graphd1_1     /usr/local/nebula/bin/nebu ...   Up (healthy)   0.0.0.0:33170->19669/tcp, 0.0.0.0:33169->19670/tcp, 0.0.0.0:33173->9669/tcp
nebula-docker-compose_graphd2_1     /usr/local/nebula/bin/nebu ...   Up (healthy)   0.0.0.0:33174->19669/tcp, 0.0.0.0:33171->19670/tcp, 0.0.0.0:33176->9669/tcp
nebula-docker-compose_graphd_1      /usr/local/nebula/bin/nebu ...   Up (healthy)   0.0.0.0:33205->19669/tcp, 0.0.0.0:33204->19670/tcp, 0.0.0.0:9669->9669/tcp
nebula-docker-compose_metad0_1      ./bin/nebula-metad --flagf ...   Up (healthy)   0.0.0.0:33165->19559/tcp, 0.0.0.0:33162->19560/tcp, 0.0.0.0:33167->9559/tcp,
                                                                                    9560/tcp
nebula-docker-compose_metad1_1      ./bin/nebula-metad --flagf ...   Up (healthy)   0.0.0.0:33166->19559/tcp, 0.0.0.0:33163->19560/tcp, 0.0.0.0:33168->9559/tcp,
                                                                                    9560/tcp
nebula-docker-compose_metad2_1      ./bin/nebula-metad --flagf ...   Up (healthy)   0.0.0.0:33161->19559/tcp, 0.0.0.0:33160->19560/tcp, 0.0.0.0:33164->9559/tcp,
                                                                                    9560/tcp
nebula-docker-compose_storaged0_1   ./bin/nebula-storaged --fl ...   Up (healthy)   0.0.0.0:33180->19779/tcp, 0.0.0.0:33178->19780/tcp, 9777/tcp, 9778/tcp,
                                                                                    0.0.0.0:33183->9779/tcp, 9780/tcp
nebula-docker-compose_storaged1_1   ./bin/nebula-storaged --fl ...   Up (healthy)   0.0.0.0:33175->19779/tcp, 0.0.0.0:33172->19780/tcp, 9777/tcp, 9778/tcp,
                                                                                    0.0.0.0:33177->9779/tcp, 9780/tcp
nebula-docker-compose_storaged2_1   ./bin/nebula-storaged --fl ...   Up (healthy)   0.0.0.0:33184->19779/tcp, 0.0.0.0:33181->19780/tcp, 9777/tcp, 9778/tcp,
                                                                                    0.0.0.0:33185->9779/tcp, 9780/tcp
```

Nebula Graph provides services to the clients through port `9669` by default. To use other ports, modify the `docker-compose.yaml` file in the `nebula-docker-compose` directory and restart the Nebula Graph services.

3.3.5 Check the service data and logs

All the data and logs of Nebula Graph are stored persistently in the `nebula-docker-compose/data` and `nebula-docker-compose/logs` directories.

The structure of the directories is as follows:

```
nebula-docker-compose/
  |-- docker-compose.yaml
  ├── data
  │   ├── meta0
  │   ├── meta1
  │   ├── meta2
  │   ├── storage0
  │   ├── storage1
  │   └── storage2
  └── logs
      ├── graph
      ├── graph1
      ├── graph2
      ├── meta0
      ├── meta1
      ├── meta2
      ├── storage0
      ├── storage1
      └── storage2
```

3.3.6 Stop the Nebula Graph services

You can run the following command to stop the Nebula Graph services:

```
$ docker-compose down
```

The following information indicates you have successfully stopped the Nebula Graph services:

```
Stopping nebula-docker-compose_storaged0_1 ... done
Stopping nebula-docker-compose_graphd1_1   ... done
Stopping nebula-docker-compose_graphd_1    ... done
Stopping nebula-docker-compose_storaged1_1 ... done
Stopping nebula-docker-compose_graphd2_1   ... done
Stopping nebula-docker-compose_storaged2_1 ... done
Stopping nebula-docker-compose_metad0_1    ... done
Stopping nebula-docker-compose_metad2_1    ... done
Stopping nebula-docker-compose_metad1_1    ... done
Removing nebula-docker-compose_storaged0_1 ... done
Removing nebula-docker-compose_graphd1_1   ... done
Removing nebula-docker-compose_graphd_1    ... done
Removing nebula-docker-compose_storaged1_1 ... done
Removing nebula-docker-compose_graphd2_1   ... done
Removing nebula-docker-compose_storaged2_1 ... done
Removing nebula-docker-compose_metad0_1    ... done
Removing nebula-docker-compose_metad2_1    ... done
Removing nebula-docker-compose_metad1_1    ... done
Removing network nebula-docker-compose_nebula-net
```

> **Note**
>
> Command `docker-compose down -v` will **delete** all your local Nebula Graph storage data. Try this command if you're using a developing/nightly version and having some compatibility issues.

3.3.7 Other ways to install Nebula Graph

- Use Source Code
- Use RPM or DEB package
- Deploy Nebula Graph cluster

3.3.8 FAQ

How to update the docker images of Nebula Graph services

To update the images of the Graph Service, Storage Service, and Meta Service, run `docker-compose pull` in the `nebula-docker-compose` directory.

`ERROR: toomanyrequests` when `docker-compose pull`

You may meet the following error.

`ERROR: toomanyrequests: You have reached your pull rate limit. You may increase the limit by authenticating and upgrading: https://www.docker.com/increase-rate-limit`.

You have met the rate limit of Docker Hub. Learn more on Understanding Docker Hub Rate Limiting.

How to update the Nebula Console client

To update the Nebula Console client, run the following command.

```
docker pull vesoft/nebula-console:v2-nightly
```

How to upgrade Nebula Graph services

To upgrade Nebula Graph, update the Nebula Graph docker images and restart the services.

1. In the `nebula-docker-compose` directory, run `docker-compose pull` to update the Nebula Graph docker images.

   > **Caution**
   >
   > Make sure that you have backed up all important data before following the next step to stop the Nebula Graph services.

2. Run `docker-compose down` to stop the Nebula Graph services.
3. Run `docker-compose up -d` to start the Nebula Graph services again.

Why can't I connect to Nebula Graph through port 3699 after updating the nebula-docker-compose repository? (Nebula Graph 2.0.0-RC)

On the release of Nebula Graph 2.0.0-RC, the default port for connection changed from 3699 to 9669. To connect to Nebula Graph after updating the repository, use port 9669 or modify the port number in the `docker-compose.yaml` file.

Why can't I access the data after updating the nebula-docker-compose repository? (Jan 4, 2021)

If you updated the nebula-docker-compose repository after Jan 4, 2021 and there are pre-existing data, modify the `docker-compose.yaml` file and change the port numbers to the previous ones before connecting to Nebula Graph.

Why can't I access the data after updating the nebula-docker-compose repository? (Jan 27, 2021)

The data format is incompatible before and after in Jan 27, 2021. Run `docker-compose down -v` to delete all your local data.

Where are the data stored when Nebula Graph is deployed with Docker Compose

If deployed with Docker Compose, Nebula Graph stores all data in `nebula-docker-compose/data/`.

Last update: April 22, 2021

3.4 Manage Nebula Graph services

You can use the `nebula.service` script to start, stop, restart, terminate, and check the Nebula Graph services. This topic takes starting, stopping and checking the Nebula Graph services for examples.

`nebula.service` is stored in the `/usr/local/nebula/` directory by default, which is also the default installation path of Nebula Graph. If you have customized the path, use the actual path in your environment.

3.4.1 Syntax

```
$ sudo /usr/local/nebula/scripts/nebula.service
[-v] [-c <config_file_path>]
<start|stop|restart|status|kill>
<metad|graphd|storaged|all>
```

Parameter	Description
`-v`	Display detailed debugging information.
`-c`	Specify the configuration file path. The default path is `/usr/local/nebula/etc/`.
`start`	Start the target services.
`stop`	Stop the target services.
`restart`	Restart the target services.
`kill`	Terminate the target services.
`status`	Check the status of the target services.
`metad`	Set the Meta Service as the target service.
`graphd`	Set the Graph Service as the target service.
`storaged`	Set the Storage Service as the target service.
`all`	Set all the Nebula Graph services as the target services.

3.4.2 Start Nebula Graph

In non-container environment

Run the following command to start Nebula Graph.

```
$ sudo /usr/local/nebula/scripts/nebula.service start all
[INFO] Starting nebula-metad...
[INFO] Done
[INFO] Starting nebula-graphd...
[INFO] Done
[INFO] Starting nebula-storaged...
[INFO] Done
```

In docker container (deployed with docker-compose)

Run the following command in the `nebula-docker-compose/` directory to start Nebula Graph.

```
nebula-docker-compose]$ docker-compose up -d
Building with native build. Learn about native build in Compose here: https://docs.docker.com/go/compose-native-build/
Creating network "nebula-docker-compose_nebula-net" with the default driver
Creating nebula-docker-compose_metad0_1 ... done
Creating nebula-docker-compose_metad2_1 ... done
Creating nebula-docker-compose_metad1_1 ... done
Creating nebula-docker-compose_storaged2_1 ... done
Creating nebula-docker-compose_graphd1_1   ... done
Creating nebula-docker-compose_storaged1_1 ... done
Creating nebula-docker-compose_storaged0_1 ... done
Creating nebula-docker-compose_graphd2_1   ... done
Creating nebula-docker-compose_graphd_1    ... done
```

3.4.3 Stop Nebula Graph

> **Danger**
>
> Don't run `kill -9` to forcibly terminate the processes, otherwise, there is a low probability of data loss.

In non-container environment

Run the following command to stop Nebula Graph.

```
sudo /usr/local/nebula/scripts/nebula.service stop all
[INFO] Stopping nebula-metad...
[INFO] Done
[INFO] Stopping nebula-graphd...
[INFO] Done
[INFO] Stopping nebula-storaged...
[INFO] Done
```

In docker container (deployed with docker-compose)

Run the following command in the `nebula-docker-compose/` directory to stop Nebula Graph.

```
nebula-docker-compose]$ docker-compose down
Stopping nebula-docker-compose_graphd_1    ... done
Stopping nebula-docker-compose_graphd2_1   ... done
Stopping nebula-docker-compose_storaged0_1 ... done
Stopping nebula-docker-compose_storaged1_1 ... done
Stopping nebula-docker-compose_graphd1_1   ... done
Stopping nebula-docker-compose_storaged2_1 ... done
Stopping nebula-docker-compose_metad1_1    ... done
Stopping nebula-docker-compose_metad2_1    ... done
Stopping nebula-docker-compose_metad0_1    ... done
Removing nebula-docker-compose_graphd_1    ... done
Removing nebula-docker-compose_graphd2_1   ... done
Removing nebula-docker-compose_storaged0_1 ... done
Removing nebula-docker-compose_storaged1_1 ... done
Removing nebula-docker-compose_graphd1_1   ... done
Removing nebula-docker-compose_storaged2_1 ... done
Removing nebula-docker-compose_metad1_1    ... done
Removing nebula-docker-compose_metad2_1    ... done
Removing nebula-docker-compose_metad0_1    ... done
Removing network nebula-docker-compose_nebula-net
```

If you are using a development or nightly version for testing and have compatibility issues, try to run 'docker-compose down-v' to **DELETE** all data stored in Nebula Graph and import data again.

3.4.4 Check the service status

In non-container environment

Run the following command to check the service status of Nebula Graph.

```
$ sudo /usr/local/nebula/scripts/nebula.service status all
```

- Nebula Graph is running normally if the following information is returned.

```
[INFO] nebula-metad: Running as 26601, Listening on 9559
[INFO] nebula-graphd: Running as 26644, Listening on 9669
[INFO] nebula-storaged: Running as 26709, Listening on 9779
```

- If the return information is similar to the following one, there is a problem.

```
[INFO] nebula-metad: Running as 25600, Listening on 9559
[INFO] nebula-graphd: Exited
[INFO] nebula-storaged: Running as 25646, Listening on 9779
```

The Nebula Graph services consist of the Meta Service, Graph Service, and Storage Service. The configuration files for all three services are stored in the `/usr/local/nebula/etc/` directory by default. You can check the configuration files according to the return information to troubleshoot problems.

You may also go to the Nebula Graph community for help.

In docker container (deployed with docker-compose)

Run the following command in the `nebula-docker-compose/` directory to check the service status of Nebula Graph.

```
nebula-docker-compose]$ docker-compose ps
            Name                           Command                  State                                        Ports
-------------------------------------------------------------------------------------------------------------------------------------------------------------------
nebula-docker-compose_graphd1_1     /usr/local/nebula/bin/nebu ...   Up (healthy)   0.0.0.0:49223->19669/tcp, 0.0.0.0:49222->19670/tcp, 0.0.0.0:49224->9669/tcp
nebula-docker-compose_graphd2_1     /usr/local/nebula/bin/nebu ...   Up (healthy)   0.0.0.0:49229->19669/tcp, 0.0.0.0:49228->19670/tcp, 0.0.0.0:49230->9669/tcp
nebula-docker-compose_graphd_1      /usr/local/nebula/bin/nebu ...   Up (healthy)   0.0.0.0:49221->19669/tcp, 0.0.0.0:49220->19670/tcp, 0.0.0.0:9669->9669/tcp
nebula-docker-compose_metad0_1      ./bin/nebula-metad --flagf ...   Up (healthy)   0.0.0.0:49212->19559/tcp, 0.0.0.0:49211->19560/tcp, 0.0.0.0:49213->9559/tcp,
                                                                                    9560/tcp
nebula-docker-compose_metad1_1      ./bin/nebula-metad --flagf ...   Up (healthy)   0.0.0.0:49209->19559/tcp, 0.0.0.0:49208->19560/tcp, 0.0.0.0:49210->9559/tcp,
                                                                                    9560/tcp
nebula-docker-compose_metad2_1      ./bin/nebula-metad --flagf ...   Up (healthy)   0.0.0.0:49206->19559/tcp, 0.0.0.0:49205->19560/tcp, 0.0.0.0:49207->9559/tcp,
                                                                                    9560/tcp
nebula-docker-compose_storaged0_1   ./bin/nebula-storaged --fl ...   Up (healthy)   0.0.0.0:49218->19779/tcp, 0.0.0.0:49217->19780/tcp, 9777/tcp, 9778/tcp,
                                                                                    0.0.0.0:49219->9779/tcp, 9780/tcp
nebula-docker-compose_storaged1_1   ./bin/nebula-storaged --fl ...   Up (healthy)   0.0.0.0:49215->19779/tcp, 0.0.0.0:49214->19780/tcp, 9777/tcp, 9778/tcp,
                                                                                     0.0.0.0:49216->9779/tcp, 9780/tcp
nebula-docker-compose_storaged2_1   ./bin/nebula-storaged --fl ...   Up (healthy)   0.0.0.0:49226->19779/tcp, 0.0.0.0:49225->19780/tcp, 9777/tcp, 9778/tcp,
                                                                                    0.0.0.0:49227->9779/tcp, 9780/tcp
```

To troubleshoot for a specific service:

1. Confirm the container name in the preceding return information.
2. Run `docker ps` to find the `CONTAINER ID`.
3. Use the `CONTAINER ID` to log in the container and troubleshoot.

```
nebula-docker-compose]$ docker exec -it 2a6c56c405f5 bash
[root@2a6c56c405f5 nebula]#
```

Last update: April 22, 2021

3.5 Connect to Nebula Graph

Nebula Graph supports multiple types of clients, including a CLI client, a GUI client, and clients developed in popular programming languages. This topic provides an overview of Nebula Graph clients and basic instructions on how to use the native CLI client, Nebula Console.

3.5.1 Nebula Graph clients

You can use supported clients or console to connect to Nebula Graph.

3.5.2 Use Nebula Console to connect to Nebula Graph

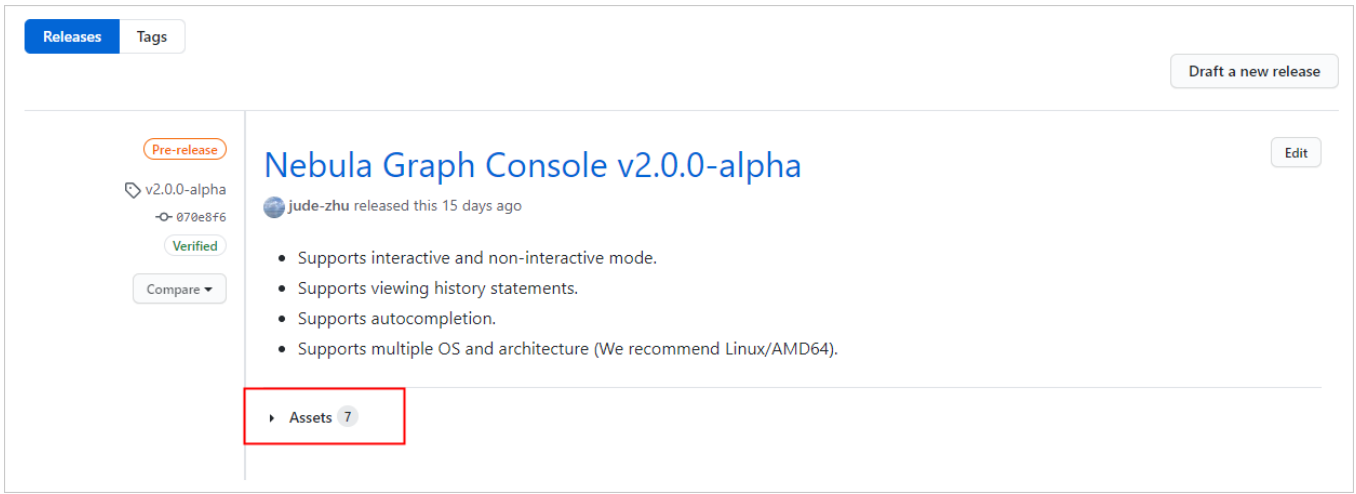
Prerequisites

- You have started the Nebula Graph services. For how to start the services, see Start and Stop Nebula Graph.
- The machine you plan to run Nebula Console on has network access to the Nebula Graph services.

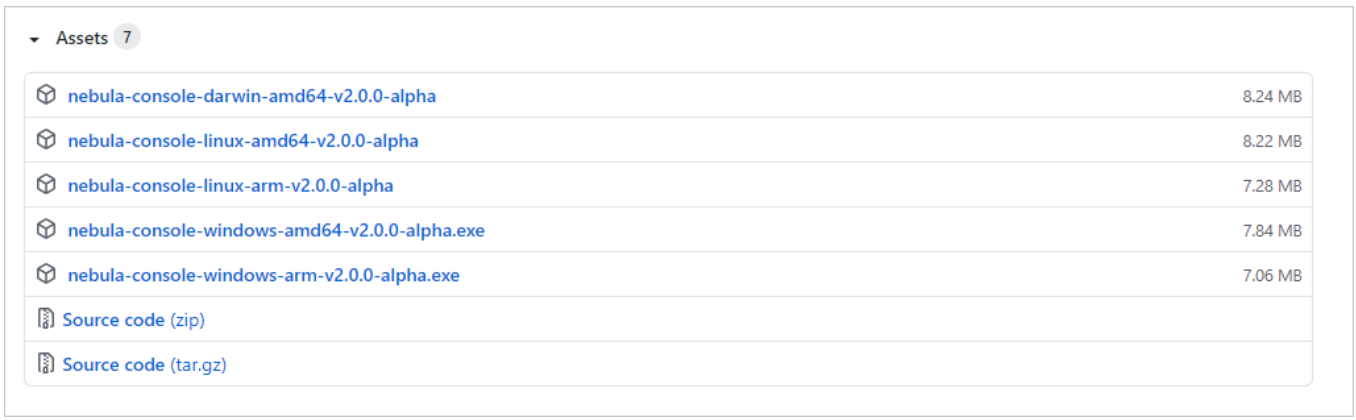
Steps

1. On the nebula-console page, select a Nebula Console version and click **Assets**.

> **Note**
>
> We recommend that you select the **latest** release.



2. In the **Assets** area, find the correct binary file for the machine where you want to run Nebula Console and download the file to the machine.



3. (Optional) Rename the binary file to `nebula-console` for convenience.

> **Note**
>
> For Windows, rename the file to `nebula-console.exe`.

4. On the machine to run Nebula Console, grant the execute permission of the nebula-console binary file to the user.

> **Note**
>
> For Windows, skip this step.

```
$ chmod 111 nebula-console
```

5. In the command line interface, change the working directory to the one where the nebula-console binary file is stored.
6. Run the following command to connect to Nebula Graph.

   • For Linux or macOS:

```
$ ./nebula-console -addr <ip> -port <port> -u <username> -p <password>
[-t 120] [-e "nGQL_statement" | -f filename.nGQL]
```

   • For Windows:

```
> nebula-console.exe -addr <ip> -port <port> -u <username> -p <password>
[-t 120] [-e "nGQL_statement" | -f filename.nGQL]
```

The description of the parameters is as follows.

Option	Description
`-h`	Shows the help menu.
`-addr`	Sets the IP address of the graphd service. The default address is 127.0.0.1.
`-port`	Sets the port number of the graphd service. The default port number is 9669.
`-u/-user`	Sets the username of your Nebula Graph account. Before enabling authentication, you can use any characters as the username.
`-p/-password`	Sets the password of your Nebula Graph account. Before enabling authentication, you can use any characters as the password.
`-t/-timeout`	Sets an integer-type timeout threshold of the connection. The unit is second. The default value is 120.
`-e/-eval`	Sets a string-type nGQL statement. The nGQL statement is executed once the connection succeeds. The connection stops after the result is returned.
`-f/-file`	Sets the path of an nGQL file. The nGQL statements in the file are executed once the connection succeeds. You'll get the return messages and the connection stops then.

You can find more details in the Nebula Console Repository.

3.5.3 Nebula Console export mode

When the export mode is enabled, Nebula Console exports all the query results into a CSV file. When the export mode is disabled, the export stops. The syntax is as follows.

> **Note**
>
> - The following commands are case insensitive.
> - The CSV file is stored in the working directory. Run the Linux command `pwd` to show the working directory.

- Enable Nebula Console export mode:

```
nebula> :SET CSV <your_file.csv>
```

- Disable Nebula Console export mode:

```
nebula> :UNSET CSV
```

3.5.4 Disconnect Nebula Console from Nebula Graph

You can use `:EXIT` or `:QUIT` to disconnect from Nebula Graph. For convenience, Nebula Console supports using these commands in lower case without the colon (":"), such as `quit`.

```
nebula> :QUIT

Bye root!
```

3.5.5 FAQ

How can I install Nebula Console from the source code

To download and compile the latest source code of Nebula Console, follow the instructions on the nebula console GitHub page.

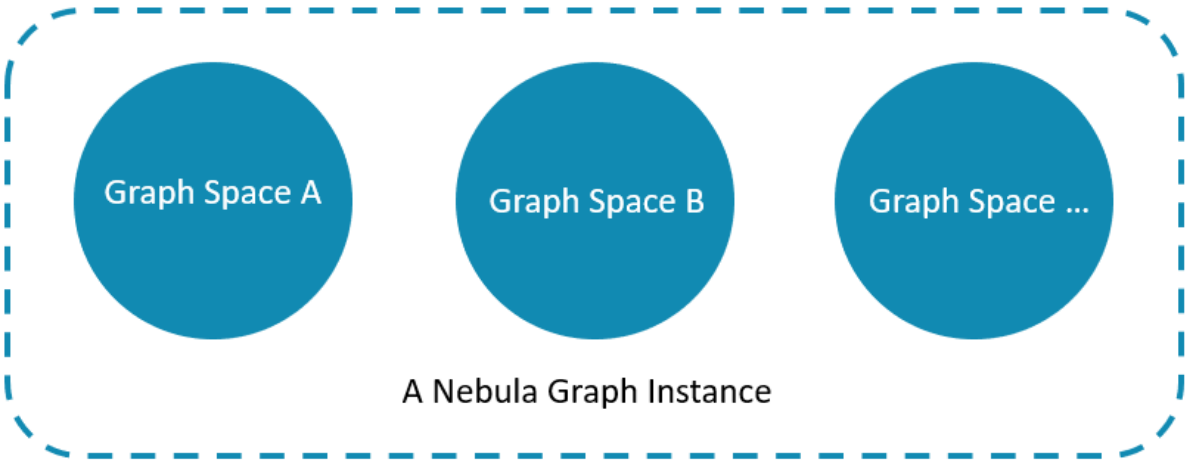
Last update: April 22, 2021

3.6 Nebula Graph CRUD

This topic describes the basic CRUD operations in Nebula Graph.

3.6.1 Graph space and Nebula Graph schema

A Nebula Graph instance consists of one or more graph spaces. Graph spaces are physically isolated from each other. You can use different graph spaces in the same instance to store different datasets.

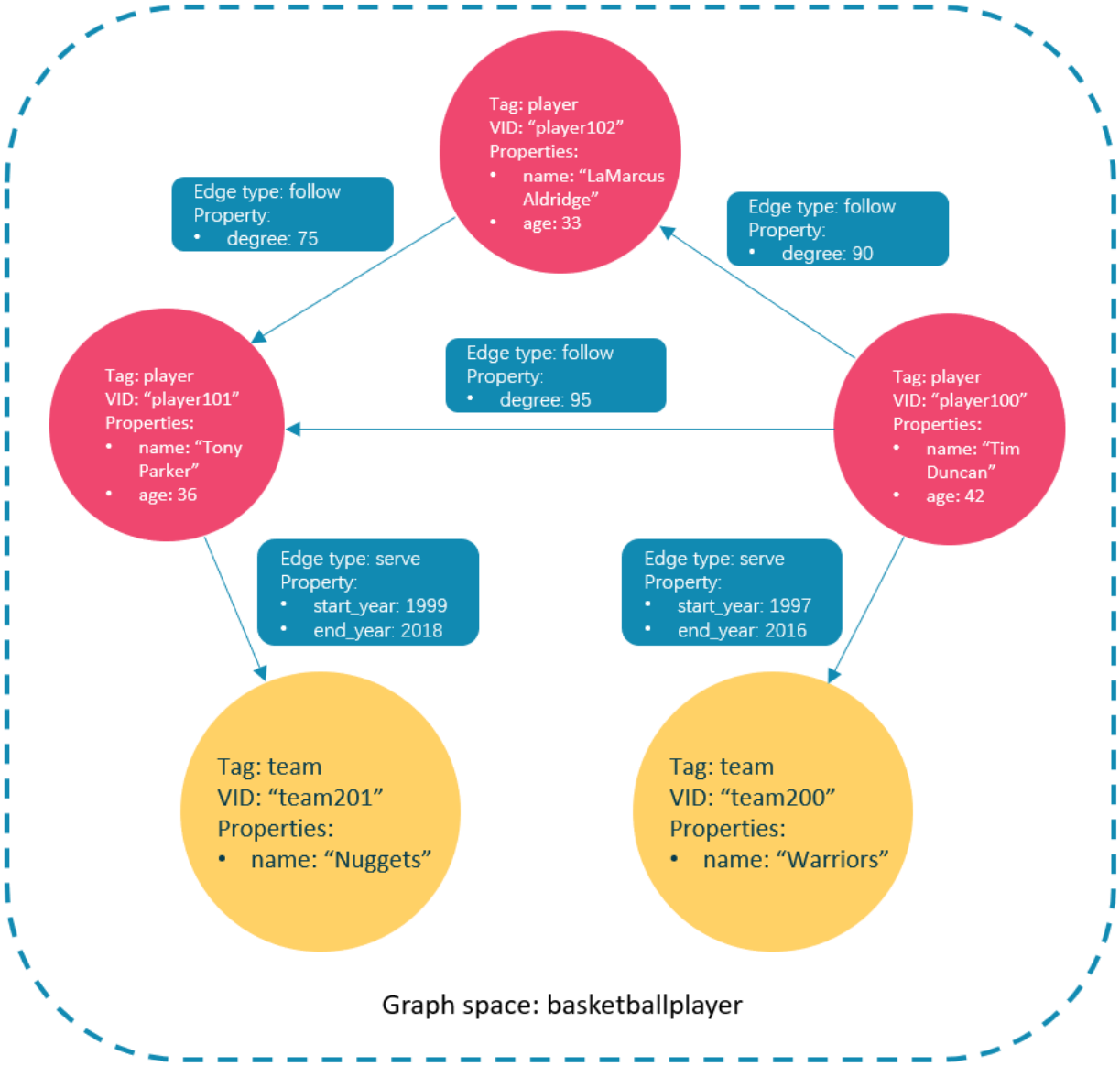


To insert data into a graph space, define a schema for the graph database. Nebula Graph schema is based on the following components.

Schema component	Description
Vertex	Represents an entity in the real world. A vertex can have one or more tags.
Tag	The type of a vertex. It defines a group of properties that describes a type of vertices.
Edge	Represents a **directed** relationship between two vertices.
Edge type	The type of an edge. It defines a group of properties that describes a type of edges.

For more information, see Data modeling.

In this topic, we use the following dataset to demonstrate basic CRUD operations.



3.6.2 Check the machine status in the Nebula Graph cluster

First, we recommend that you check the machine status to make sure that all the Storage services are connected to the Meta Services. Run `SHOW HOSTS` as follows.

```
nebula> SHOW HOSTS;
+-------------+-----------+-----------+--------------+----------------------+------------------------+
| Host        | Port      | Status    | Leader count | Leader distribution  | Partition distribution |
+-------------+-----------+-----------+--------------+----------------------+------------------------+
| "storaged0" | 9779      | "ONLINE"  | 0            | "No valid partition" | "No valid partition"   |
+-------------+-----------+-----------+--------------+----------------------+------------------------+
| "storaged1" | 9779      | "ONLINE"  | 0            | "No valid partition" | "No valid partition"   |
+-------------+-----------+-----------+--------------+----------------------+------------------------+
| "storaged2" | 9779      | "ONLINE"  | 0            | "No valid partition" | "No valid partition"   |
+-------------+-----------+-----------+--------------+----------------------+------------------------+
| "Total"     | __EMPTY__ | __EMPTY__ | 0            | __EMPTY__            | __EMPTY__              |
+-------------+-----------+-----------+--------------+----------------------+------------------------+
Got 4 rows (time spent 1061/2251 us)
```

From the **Status** column of the table in the return message, you can see that all the Storage services are online.

Asynchronous implementation of creation and alteration

Nebula Graph implements the following creation or alteration operations asynchronously in the next heartbeat cycle. The operations won't take effect until they finish.

- `CREATE SPACE`
- `CREATE TAG`
- `CREATE EDGE`
- `ALTER TAG`
- `ALTER EDGE`
- `CREATE TAG INDEX`
- `CREATE EDGE INDEX`

> **Note**
>
> The default heartbeat interval is 10 seconds. To change the heartbeat interval, modify the `heartbeat_interval_secs` parameter in the configuration files for all services.

To make sure the follow-up operations work as expected, take one of the following approaches:

- Run `SHOW` or `DESCRIBE` statements accordingly to check the status of the objects, and make sure the creation or alteration is complete. If it is not, wait a few seconds and try again.
- Wait for two heartbeat cycles, i.e., 20 seconds.

3.6.3 Create and use a graph space

nGQL syntax

- Create a graph space:

```
CREATE SPACE [IF NOT EXISTS] <graph_space_name>
    [(partition_num = <partition_number>,
    replica_factor = <replica_number>,
    vid_type = {FIXED_STRING(<N>) | INT64})];
```

```
| Property       |
Description
|
| -------------- |
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
|
| partition_num  | Specifies the number of partitions in each replica. The suggested number is the number of hard disks in the cluster times 5. For example, if you
have 3 hard disks in the cluster, we recommend that you set 15 partitions.           |
| replica_factor | Specifies the number of replicas in the Nebula Graph cluster. The suggested number is 3 in a production environment and 1 in a test environment.
The replica number must always be an **odd** number for the need of quorum-based voting. |
|vid_type | Specifies the data type of VIDs in a graph space. Available values are `FIXED_STRING(N)` and `INT64`. `N` represents the maximum length of the VIDs and
it must be a positive integer. The default value is `FIXED_STRING(8)`. If you set a VID length greater than `N`, Nebula Graph throws an error. To set the integer VID
for vertices, set `vid_type` to `INT64`. |
```

- List graph spaces and check if the creation is successful:

```
nebula> SHOW SPACES;
```

- Use a graph space:

```
USE <graph_space_name>
```

Examples

1. Use the following statement to create a graph space named `basketballplayer`.

```
nebula> CREATE SPACE basketballplayer(partition_num=15, replica_factor=1, vid_type=fixed_string(30));
Execution succeeded (time spent 2817/3280 us)
```

2. Check the partition distribution with `SHOW HOSTS` to make sure that the partitions are distributed in a balanced way.

```
nebula> SHOW HOSTS;
+-------------+-----------+-----------+--------------+---------------------------------+------------------------+
| Host        | Port      | Status    | Leader count | Leader distribution             | Partition distribution |
+-------------+-----------+-----------+--------------+---------------------------------+------------------------+
| "storaged0" | 9779      | "ONLINE"  | 5            | "basketballplayer:5"            | "basketballplayer:5"   |
+-------------+-----------+-----------+--------------+---------------------------------+------------------------+
| "storaged1" | 9779      | "ONLINE"  | 5            | "basketballplayer:5"            | "basketballplayer:5"   |
+-------------+-----------+-----------+--------------+---------------------------------+------------------------+
| "storaged2" | 9779      | "ONLINE"  | 5            | "basketballplayer:5"            | "basketballplayer:5"   |
+-------------+-----------+-----------+--------------+---------------------------------+------------------------+
| "Total"     | __EMPTY__ | __EMPTY__ | 15           | "basketballplayer:15"           | "basketballplayer:15"  |
+-------------+-----------+-----------+--------------+---------------------------------+------------------------+
Got 4 rows (time spent 1633/2867 us)
```

If the **Leader distribution** is uneven, use `BALANCE LEADER` to redistribute the partitions. For more information, see BALANCE.

3. Use the `basketballplayer` graph space.

```
nebula> USE basketballplayer;
Execution succeeded (time spent 1322/2206 us)
```

You can use `SHOW SPACES` to check the graph space you created.

```
nebula> SHOW SPACES;
+------------------+
| Name             |
+------------------+
| basketballplayer |
+------------------+
Got 1 rows (time spent 1235/1934 us)
```

3.6.4 Create tags and edge types

nGQL syntax

```
CREATE {TAG | EDGE} {<tag_name> | <edge_type>}(<property_name> <data_type>
[, <property_name> <data_type> ...]);
```

Examples

Create tags `player` and `team`, edge types `follow` and `serve`.

Component name	Type	Property
player	Tag	name (string), age (int)
team	Tag	name (string)
follow	Edge type	degree (int)
serve	Edge type	start_year (int), end_year (int)

```
nebula> CREATE TAG player(name string, age int);
Execution succeeded (time spent 2694/3116 us)

Thu, 15 Oct 2020 06:22:29 UTC

nebula> CREATE TAG team(name string);
Execution succeeded (time spent 2630/3002 us)

Thu, 15 Oct 2020 06:22:37 UTC

nebula> CREATE EDGE follow(degree int);
Execution succeeded (time spent 3087/3467 us)

Thu, 15 Oct 2020 06:22:43 UTC

nebula> CREATE EDGE serve(start_year int, end_year int);
Execution succeeded (time spent 2645/3123 us)

Thu, 15 Oct 2020 06:22:50 UTC
```

3.6.5 Insert vertices and edges

You can use the `INSERT` statement to insert vertices or edges based on existing tags or edge types.

nGQL syntax

- Insert vertices:

```
INSERT VERTEX <tag_name> (<property_name>[, <property_name>...])
[, <tag_name> (<property_name>[, <property_name>...]), ...]
{VALUES | VALUE} <vid>: (<property_value>[, <property_value>...])
[, <vid>: (<property_value>[, <property_value>...];
```

`VID` is short for vertex ID. A `VID` must be a unique string value in a graph space.

- Insert edges:

```
INSERT EDGE <edge_type> (<property_name>[, <property_name>...])
{VALUES | VALUE} <src_vid> -> <dst_vid>[@<rank>] : (<property_value>[, <property_value>...])
[, <src_vid> -> <dst_vid>[@<rank> : (<property_name>[, <property_name>...]), ...]
```

Examples

- Insert vertices representing basketball players and teams:

```
nebula> INSERT VERTEX player(name, age) VALUES "player100":("Tim Duncan", 42);
Execution succeeded (time spent 2919/3485 us)

Fri, 16 Oct 2020 03:41:00 UTC

nebula> INSERT VERTEX player(name, age) VALUES "player101":("Tony Parker", 36);
Execution succeeded (time spent 3007/3539 us)

Fri, 16 Oct 2020 03:41:58 UTC

nebula> INSERT VERTEX player(name, age) VALUES "player102":("LaMarcus Aldridge", 33);
Execution succeeded (time spent 2449/2934 us)

Fri, 16 Oct 2020 03:42:16 UTC

nebula> INSERT VERTEX team(name) VALUES "team200":("Warriors"), "team201":("Nuggets");
Execution succeeded (time spent 3514/4331 us)

Fri, 16 Oct 2020 03:42:45 UTC
```

- Insert edges representing the relations between basketball players and teams:

```
nebula> INSERT EDGE follow(degree) VALUES "player100" -> "player101":(95);
Execution succeeded (time spent 1488/1918 us)

Wed, 21 Oct 2020 06:57:32 UTC

nebula> INSERT EDGE follow(degree) VALUES "player100" -> "player102":(90);
Execution succeeded (time spent 2483/2890 us)

Wed, 21 Oct 2020 07:05:48 UTC

nebula> INSERT EDGE follow(degree) VALUES "player102" -> "player101":(75);
Execution succeeded (time spent 1208/1689 us)

Wed, 21 Oct 2020 07:07:12 UTC

nebula> INSERT EDGE serve(start_year, end_year) VALUES "player100" -> "team200":(1997, 2016), "player101" -> "team201":(1999,  2018);
Execution succeeded (time spent 2170/2651 us)

Wed, 21 Oct 2020 07:08:59 UTC
```

3.6.6 Read data

- The GO statement traverses the database based on specific conditions. A `GO` traversal starts from one or more vertices, along one or more edges, and return information in a form specified in the `YIELD` clause.
- The FETCH statement is used to get properties from vertices or edges.
- The LOOKUP statement is based on indexes. It is used together with the `WHERE` clause to search for the data that meet the specific conditions.

- The MATCH statement is the most commonly used statement for graph data querying. But, it relies on indexes to match data patterns in Nebula Graph.

nGQL syntax

- `GO`

```
GO [[<M> TO] <N> STEPS ] FROM <vertex_list>
OVER <edge_type_list> [REVERSELY] [BIDIRECT]
[WHERE <expression> [AND | OR expression ...])]
YIELD [DISTINCT] <return_list>
```

- `FETCH`
  - Fetch properties on tags:

    ```
    FETCH PROP ON {<tag_name> | <tag_name_list> | *} <vid_list>
    [YIELD [DISTINCT] <return_list>]
    ```

  - Fetch properties on edges:

    ```
    FETCH PROP ON <edge_type> <src_vid> -> <dst_vid>[@<rank>]
    [, <src_vid> -> <dst_vid> ...]
    [YIELD [DISTINCT] <return_list>]
    ```

- `LOOKUP`

```
LOOKUP ON {<tag_name> | <edge_type>}
WHERE <expression> [AND expression ...])]
[YIELD <return_list>]
```

- `MATCH`

```
MATCH <pattern> [<WHERE clause>] RETURN <output>
```

Examples of GO

- Find the vertices that VID "player100" follows.

```
nebula> GO FROM "player100" OVER follow;
+-------------+
| follow._dst |
+-------------+
| player101   |
+-------------+
| player102   |
```

```
+-------------+
Got 2 rows (time spent 1935/2420 us)
```

- Search for the players that the player with VID "player100" follows. Filter the players that the player with VID "player100" follows whose age is equal to or greater than 35. Rename the columns in the result with `Teammate` and `Age`.

```
nebula> GO FROM "player100" OVER follow WHERE $$.player.age >= 35 \
YIELD $$.player.name AS Teammate, $$.player.age AS Age;
+-------------+-----+
| Teammate    | Age |
+-------------+-----+
| Tony Parker | 36  |
+-------------+-----+
Got 1 rows (time spent 3871/4349 us)
```

Clause/Sign	Description
`YIELD`	Specifies what values or results you want to return from the query.
`$$`	Represents the target vertices.
`\`	A line-breaker.

- Search for the players that the player with VID "player100" follows. Then Retrieve the teams of the players that the player with VID "player100" follows. To combine the two queries, use a pipe or a temporary variable.
  - With a pipe:

```
nebula> GO FROM "player100" OVER follow YIELD follow._dst AS id | \
GO FROM $-.id OVER serve YIELD $$.team.name AS Team, \
$^.player.name AS Player;
+---------+-------------+
| Team    | Player      |
+---------+-------------+
| Nuggets | Tony Parker |
+---------+-------------+
Got 1 rows (time spent 2902/3496 us)
```

Clause/Sign	Description
`$^`	Represents the source vertex of the edge.
`|`	A pipe symbol that can combine multiple queries.
`$-`	Represents the output of the query before the pipe symbol.

  - With a temporary variable:

> **Note**
>
> Once a compound statement is submitted to the server as a whole, the life cycle of the temporary variables in the statement ends.

```
nebula> $var = GO FROM "player100" OVER follow YIELD follow._dst AS id; \
GO FROM $var.id OVER serve YIELD $$.team.name AS Team, \
$^.player.name AS Player;
+---------+-------------+
| Team    | Player      |
+---------+-------------+
| Nuggets | Tony Parker |
+---------+-------------+
Got 1 rows (time spent 3103/3711 us)
```

Example of FETCH

Use `FETCH`: Fetch the properties of the player with VID player100.

```
nebula> FETCH PROP ON player "player100";
+-------------------------------------------------+
| vertices_                                       |
+-------------------------------------------------+
| ("player100" :player{age: 42, name: "Tim Duncan"}) |
```

```
+-----------------------------------------------------+
Got 1 rows (time spent 2006/2406 us)
```

3.6.7 Update vertices and edges

You can use the `UPDATE` statement or the `UPSERT` statement to update existing data.

`UPSERT` is the combination of `UPDATE` and `INSERT`. If you update a vertex or an edge with `UPSERT`, it inserts a new vertex or edge if it does not exist.

Note: `UPSERT` operates in serial a (partition-based) order and therefore is slower comparing with `INSERT` OR `UPDATE`.

nGQL syntax

- `UPDATE` vertices:

```
UPDATE VERTEX <vid> SET <properties to be updated>
[WHEN <condition>] [YIELD <columns>]
```

- `UPDATE` edges:

```
UPDATE EDGE <source vid> -> <destination vid> [@rank] OF <edge_type>
SET <properties to be updated> [WHEN <condition>] [YIELD <columns to be output>]
```

- `UPSERT` vertices or edges:

```
UPSERT {VERTEX <vid> | EDGE <edge_type>} SET <update_columns>
[WHEN <condition>] [YIELD <columns>]
```

Examples

- `UPDATE` the `name` property of the vertex with VID "player100" and check the result with the `FETCH` statement:

```
nebula> UPDATE VERTEX "player100" SET player.name = "Tim";
Execution succeeded (time spent 3483/3914 us)

Wed, 21 Oct 2020 10:53:14 UTC

nebula> FETCH PROP ON player "player100";
+-------------------------------------------+
| vertices_                                 |
+-------------------------------------------+
| ("player100" :player{age: 42, name: "Tim"}) |
```

```
+-------------------------------------------+
Got 1 rows (time spent 2463/3042 us)
```

- `UPDATE` the degree value of an edge and check the result with the `FETCH` statement:

```
nebula> UPDATE EDGE "player100" -> "player101" OF follow SET degree = 96;
Execution succeeded (time spent 3932/4432 us)

nebula> FETCH PROP ON follow "player100" -> "player101";
+-----------------------------------------------------+
| edges_                                              |
+-----------------------------------------------------+
| [:follow "player100"->"player101" @0 {degree: 96}] |
+-----------------------------------------------------+
Got 1 rows (time spent 2205/2800 us)
```

- Insert a vertex with VID "player111" and `UPSERT` it.

```
nebula> INSERT VERTEX player(name, age) VALUES "player111":("Ben Simmons", 22);
Execution succeeded (time spent 2115/2900 us)

Wed, 21 Oct 2020 11:11:50 UTC

nebula> UPSERT VERTEX "player111" SET player.name = "Dwight Howard", player.age = $^.player.age + 11 \
WHEN $^.player.name == "Ben Simmons" AND $^.player.age > 20 \
YIELD $^.player.name AS Name, $^.player.age AS Age;
+---------------+-----+
| Name          | Age |
+---------------+-----+
| Dwight Howard | 33  |
+---------------+-----+
Got 1 rows (time spent 1815/2329 us)
```

3.6.8 Delete vertices and edges

nGQL syntax

- Delete vertices:

```
DELETE VERTEX <vid1>[, <vid2>...]
```

- Delete edges:

```
DELETE EDGE <edge_type> <src_vid> -> <dst_vid>[@<rank>]
[, <src_vid> -> <dst_vid>...]
```

Examples

- Delete vertices:

```
nebula> DELETE VERTEX "team1", "team2";
Execution succeeded (time spent 4337/4782 us)
```

- Delete edges:

```
nebula> DELETE EDGE follow "team1" -> "team2";
Execution succeeded (time spent 3700/4101 us)
```

3.6.9 About indexes

You can add indexes to tags or edge types with the CREATE INDEX statement.

Must-read for using index

- Both `MATCH` and `LOOKUP` depend on index. But indexes can dramatically reduce the write performance. The performance reduction can be as much as 90% or even more. **DO NOT** use indexes in production environments unless you are fully aware of their influences on your service.
- You **MUST** rebuild indexes for pre-existing data. Otherwise, the pre-existing data can't be indexed (and therefore can't be returned in `Match` or `Lookup`). For more information, see REBUILD INDEX.

nGQL syntax

Create an index:

```
CREATE {TAG | EDGE} INDEX [IF NOT EXISTS] <index_name>
ON {<tag_name> | <edge_name>} (prop_name_list);
```

Rebuild an index:

```
REBUILD {TAG | EDGE} INDEX <index_name>
```

Examples

Create an index for the `name` property on all vertices with the tag `player`.

```
nebula> CREATE TAG INDEX player_index_0 on player(name(20));
nebula> REBUILD TAG INDEX player_index_0;
```

> **Note**
>
> Define the index length when creating an index for a variable-length property. For more information, see CREATE INDEX

Examples of LOOKUP and MATCH (index-based)

Make sure there is an index for `LOOKUP` or `MATCH` to use. If there is not, create an index first.

Find the information of the vertex with the tag `player` and its value of the `name` property is `"Tony Parker"`.

```
// Create an index on the player name property.
nebula> CREATE TAG INDEX player_name_0 on player(name(10));
Execution succeeded (time spent 3465/4150 us)

// Rebuild the index to make sure it takes effect on pre-existing data.
nebula> REBUILD TAG INDEX player_name_0
+------------+
| New Job Id |
+------------+
| 31         |
+------------+
Got 1 rows (time spent 2379/3033 us)

// Use LOOKUP to retrieve the vertex property.
nebula> LOOKUP ON player WHERE player.name == "Tony Parker" \
YIELD player.name, player.age;
+-------------+---------------+------------+
| VertexID    | player.name   | player.age |
+-------------+---------------+------------+
| "player101" | "Tony Parker" | 36         |
+-------------+---------------+------------+

// Use MATCH to retrieve the vertex.
nebula> MATCH (v:player{name:"Tony Parker"}) RETURN v;
+-------------------------------------------------------+
| v                                                     |
+-------------------------------------------------------+
| ("player101" :player{age: 36, name: "Tony Parker"}) |
+-------------------------------------------------------+
Got 1 rows (time spent 5132/6246 us)
```

Last update: May 10, 2021

3.7 Useful links

> **Note**
>
> This page lists all the tools and clients for Nebula Graph 2.0.1 kernel.

> **Caution**
>
> Checkout the correct commit ID. And see each document carefully for the **compatibility** with Nebula Graph 2.0.1 kernel.

3.7.1 API Clients by Nebula Graph

links	commit id
C++ Client	7305c72
Go Client	542ed24
Python Client	cb48e8a
Java Client	923bc04

The following repositories of 2.0.1 are not released yet.

- Rust Client
- Node.js Client
- HTTP Client

3.7.2 Graph tools

links	commit id
Command Line Console	1f32236
Studio	5d15d59

The following repositories of 2.0.1 are not released yet.

- Dashboard

3.7.3 Big Data and other Systems support

links	commit id
csv (a.k.a. importer)	1d87c7b
Spark util	af3fdf4
nebula-docker-compose	2c2549a

The following repositories of 2.0.1 are not released yet.

- Flink connector
- Promethus connector

3.7.4 Benchmark, test, and Backup tools

The following repositories of 2.0.1 are not released yet.

- Benchmark
- Chaos Test
- Backup&Restore

3.7.5 Misc

links	commit id
Nebula Graph 1.2.1	721ae51

- Open Source Community
- 

Last update: April 26, 2021

4. nGQL guide

4.1 nGQL overview

4.1.1 Nebula Graph Query Language (nGQL)

This document gives an introduction to the query language of Nebula Graph, nGQL.

What is nGQL

nGQL is a declarative graph query language for Nebula Graph. It allows expressive and efficient graph patterns. nGQL is designed for both developers and operations professionals. nGQL is an SQL-like query language, so it's easy to learn. nGQL is a project in progress. New features and optimizations are done steadily. There can be differences between syntax and implementation. Nebula Graph 2.0 or later version support openCypher 9.

What can nGQL do

- Supports graph traverse
- Supports pattern match
- Supports aggregation
- Supports graph mutation
- Supports access control
- Supports composite queries
- Supports index
- Supports most openCypher 9 graph query syntax (but mutations and controls syntax are not supported).

Example Data

The example data in Nebula Graph document statements can be downloaded here. After downloading the example data, you can import it to Nebula Graph by using the `-f` option in Nebula Graph Console.

Placeholder Identifiers and Values

Refer to the following standards in nGQL:

- ISO/IEC 10646
- ISO/IEC 39075
- ISO/IEC NP 39075 (Draft)
- OpenCypher 9

In template code, any token that is not a keyword, a literal value, or punctuation is a placeholder identifier or a placeholder value.

For details of the symbols in nGQL, see the following table:

Token	Meaning
< >	name of a syntactic element
::=	formula that defines an element
[ ]	optional elements
{ }	explicitly specified elements
|	complete alternative elements
...	may be repeated any number of times

Last update: April 13, 2021

4.1.2 Patterns

Patterns and graph pattern matching are the very heart of a graph query language.

Patterns for vertices

A vertex is described using a pair of parentheses, and is typically given a name. For example:

```
(a)
```

This simple pattern describes a single vertex, and names that vertex using the variable `a`.

Patterns for related vertices

A more powerful construct is a pattern that describes multiple vertices and edges between them. Patterns describe edges by employing an arrow between two vertices. For example:

```
(a)-[]->(b)
```

This pattern describes a very simple data shape: two vertices, and a single edge from one to the other. In this example, the two vertices are both named as `a` and `b` respectively, and the edge is `directed`: it goes from `a` to `b`.

This manner of describing vertices and edges can be extended to cover an arbitrary number of vertices and the edges between them, for example:

```
(a)-[]->(b)<-[]-(c)
```

Such a series of connected vertices and edges is called a **"path"**.

Note that the naming of the vertices in these patterns is only necessary should one need to refer to the same vertex again, either later in the pattern or elsewhere in the query. If this is not necessary, then the name may be omitted, as follows:

```
(a)-[]->()<-[]-(c)
```

Patterns for tags

> **OpenCypher compatibility**
>
> The concept `tag` in nGQL have a few differences from `label` in openCypher. For example, you must create a tag before using it. And a tag also defines the properties' type.

In addition to simply describing the shape of a vertex in the pattern, one can also describe attributes. The most simple attribute that can be described in the pattern is a tag that the vertex must have. For example:

```
(a:User)-[]->(b)
```

One can also describe a vertex that has multiple tags: `(a:User:Admin)-[]->(b)`.

Patterns for properties

Nodes and edges are the fundamental structures in a graph. nGQL uses properties on both of these to allow for far richer models.

Properties can be expressed in patterns using a map-construct: curly brackets surrounding a number of key-expression pairs, separated by commas. E.g. a vertex with two properties on it would look like:

```
(a {name: 'Andres', sport: 'Brazilian Ju-Jitsu'})
```

An edge with expectations on it is given by:

```
(a)-[{blocked: false}]->(b)
```

Patterns for edges

The simplest way to describe an edge is by using the arrow between two vertices, as in the previous examples.

Using this syntax, you can describe that the edge should exist and the directionality of it. If you don't care about the direction of the edge, the arrowhead is omitted, as exemplified by:

```
(a)-[]-(b)
```

As with vertices, edges may also be given names. In this case, a pair of square brackets is used to break up the arrow and the variable is placed between. For example:

```
(a)-[r]->(b)
```

Much like tags on vertices, edges can have types. To describe an edge with a specific type, use the pattern as follows:

```
(a)-[r:REL_TYPE]->(b)
```

An edge can only have one edge type. But if we'd like to describe some data such that the edge could have any one of a set of types, then they can all be listed in the pattern, separating them with the pipe symbol `|` like this:

```
(a)-[r:TYPE1|TYPE2]->(b)
```

As with vertices, the name of the edge can always be omitted, as exemplified by:

```
(a)-[:REL_TYPE]->(b)
```

Variable-length pattern

Rather than describing a long path using a sequence of many vertex and edge descriptions in a pattern, many edges (and the intermediate vertices) can be described by specifying a length in the edge description of a pattern. For example:

```
(a)-[*2]->(b)
```

This describes a graph of three vertices and two edges, all in one path (a path of length 2). This is equivalent to:

```
(a)-[]->()-[]->(b)
```

A range of lengths can also be specified: such edge patterns are called 'variable-length edges'. For example:

```
(a)-[*3..5]->(b)
```

The preceding example defines a path with a minimum length of 3, and a maximum length of 5. It describes a graph of either 4 vertices and 3 edges, 5 vertices and 4 edges, or 6 vertices and 5 edges, all connected in a single path.

the lower bound can be omitted. For example, to describe paths of length 5 or less, use:

```
(a)-[*..5]->(b)
```

> **OpenCypher compatibility**
>
> The upper bound must be specified. The following are **NOT** accepted.

```
(a)-[*3..]->(b)
(a)-[*]->(b)
```

Assigning to path variables

As described above, a series of connected vertices and edges is called a "path". nGQL allows paths to be named using a variable, as exemplified by:

```
p = (a)-[*3..5]->(b)
```

You can do this in `MATCH`.

Last update: April 22, 2021

4.2 Data types

4.2.1 Numeric types

Integer

An integer is declared with keyword `int` , which is 64-bit *signed*. The supported range is [-9223372036854775808, 9223372036854775807]. Integer constants support multiple formats:

1. Decimal, for example `123456` .
2. Hexadecimal, for example `0xdeadbeaf` .
3. Octal, for example `01234567` .

Double-precision floating-point

double-precision floating-point values is used for storing double precision floating point values. E.g., 1.2, -3.0000001. The keyword used for double floating point data type is `double` .

Scientific notation is also supported. For example, `1e2, 1.1e2, .3e4, 1.e4, -1234E-10` .

Last update: March 16, 2021

4.2.2 Boolean

A boolean data type is declared with the `bool` keyword and can only take the values `true` or `false`.

Last update: November 19, 2020

4.2.3 String

The string type is used to store a sequence of characters (text). The literal constant is a sequence of characters of any length surrounded by double or single quotes. For example `"Shaquille O'Neal"` or `'"This is a double-quoted literal string"'`. Line breaks are not allowed in a string. Embedded escape sequences are supported within strings, for example:

- `"\n\t\r\b\f"`
- `"\110ello world"`

Nebula Graph supports two kind of strings: fixed length string and variable length string. For example:

```
nebula> CREATE TAG t1 (p1 FIXED_STRING(10)); -- Fixed length string type

nebula> CREATE TAG t2 (p2 string); -- Variable length string type
```

OpenCypher Compatibility

Here is a tiny difference between openCypher and Cypher, as well as nGQL.

The following is what openCypher requires. Single-quotes can't be converted to double-quotes.

```
 #File: Literals.feature
Feature: Literals

Background:
    Given any graph
 Scenario: Return a single-quoted string
    When executing query:
      """
      RETURN '' AS literal
      """
    Then the result should be, in any order:
      | literal |
      | ''      |    # Note: it should return single-quotes as openCypher required.
    And no side effects
```

While Cypher accepts both single-quotes and double quotes as the return results. nGQL follows the Cypher way.

```
nebula > YIELD '' AS quote1, "" AS quote2, "'" AS quote3, '"' AS quote4
+--------+--------+--------+--------+
| quote1 | quote2 | quote3 | quote4 |
+--------+--------+--------+--------+
| ""     | ""     | "'"    | """    |
+--------+--------+--------+--------+
```

Last update: February 5, 2021

4.2.4 Date and time types

This document describes the `DATE`, `TIME`, `DATETIME`, and `TIMESTAMP` types. Nebula Graph converts the `DATE`, `TIME`, `DATETIME`, and `TIMESTAMP` values from the current time zone to **UTC** for storage. Nebula Graph converts back from UTC to the current time zone for retrieval.

While inserting time-type property values, except for timestamps, Nebula Graph transforms them to a UTC time according to the time zone specified with the `timezone_name` parameter in the configuration files. The time-type values returned by nGQL queries are all UTC time.

> **Note**
>
> To change the time zone, modify the `timezone_name` value in the configuration files of all Nebula Graph services.

Combined with `RETURN`, functions `date()`, `time()`, `datetime()` all accept empty parameters to return the current date, time and datetime.

OpenCypher Compatibility

In nGQL:

- Year, month, day, hour, minute, and second are supported. The millisecond field is displayed in `000`.
- `localdatetime()`, `duration()` are not supported.
- Most string time formats are not supported. The only exception is `2017-03-04T22:30:40`.

DATE

The `DATE` type is used for values with a date part but no time part. Nebula Graph retrieves and displays `DATE` values in the `YYYY-MM-DD` format. The supported range is `-32768-01-01` to `32767-12-31`.

TIME

The `TIME` type is used for values with a time part but no date part. Nebula Graph retrieves and displays `TIME` values in `hh:mm:ss:usus` format. The supported range is `0:0:0:0` to `23:59:59:999999`.

DATETIME

The `DATETIME` type is used for values that contain both date and time parts. Nebula Graph retrieves and displays `DATETIME` values in `YYYY-MM-DD hh:mm:ss:ususus` format. The supported range is `-32768-01-01 00:00:00:00` to `32767-12-31 23:59:59:999999`.

TIMESTAMP

The `TIMESTAMP` data type is used for values that contain both date and time parts.

- `TIMESTAMP` has a range of `1970-01-01 00:00:01` UTC to `2262-04-11 23:47:16` UTC.
- Timestamp is measured in units of seconds.
- Supported `TIMESTAMP` inserting methods:
  - Call the `now()` function.
  - Input `TIMESTAMP` by using a string. For example: `2019-10-01 10:00:00`.
  - Input `TIMESTAMP` directly, namely the number of seconds from `1970-01-01 00:00:00`.
- The underlying storage data type is: **int64**.

Examples

Create a tag named date.

```
nebula> CREATE TAG date(p1 date, p2 time, p3 datetime);
```

Insert a vertex named Date1.

```
nebula> INSERT VERTEX date(p1, p2, p3) VALUES "Date1":(date("2017-03-04"), time("23:01:00"), datetime("2017-03-04T22:30:40"));
```

Create a tag named school.

```
nebula> CREATE TAG school(name string , found_time timestamp);
```

Insert a vertex named "stanford" with the foundation date `"1885-10-01T08:00:00"`.

```
nebula> INSERT VERTEX school(name, found_time) VALUES "Stanford":("Stanford", timestamp("1885-10-01T08:00:00"));
```

Insert a vertex named "dut" with the foundation date now.

```
nebula> INSERT VERTEX school(name, found_time) VALUES "dut":("dut", now());
```

```
nebula> WITH time({hour: 12, minute: 31, second: 14}) AS d RETURN d;
+--------------+
| d            |
+--------------+
| 12:31:14.000 |
+--------------+

nebula> WITH date({year: 1984, month: 10, day: 11}) AS x RETURN x + 1;
+------------+
| x          |
+------------+
| 1984-10-12 |
+------------+

nebula> WITH datetime({year: 1984, month: 10, day: 11, hour: 12, minute: 31, second: 14}) AS d \
        RETURN toString(d) AS ts, datetime(toString(d)) == d AS b
+-------------------------+------+
| ts                      | b    |
+-------------------------+------+
| "1984-10-11T12:31:14.0" | true |
+-------------------------+------+
```

Last update: April 22, 2021

4.2.5 NULL

You can set the properties for vertices or edges to `NULL`. Also, you can set `NOT NULL` constraint to make sure that the property values are `NOT NULL`.

If not specified, the property is set to `NULL` by default.

Logical operations with NULL

The logical operations with NULL is the same as openCypher.

Here is the truth table for AND, OR, XOR, and NOT.

a	b	a AND b	a OR b	a XOR b	NOT a
false	false	false	false	false	true
false	null	false	null	null	true
false	true	false	true	true	true
true	false	false	true	true	false
true	null	null	true	null	false
true	true	true	true	false	false
null	false	false	null	null	null
null	null	null	null	null	null
null	true	null	true	null	null

OpenCypher compatibility

The comparisons and operations about NULL are different from openCypher.

The behavior may change later.

COMPARISONS WITH NULL

The comparison operations with NULL is incompatible with openCypher.

OPERATIONS AND EXPRESSION WITH NULL

The NULL operations and RETURN with NULL is incompatible with openCypher.

Examples

Create a tag named player. Specify the property name with `NOT NULL`. Ignore the property age constraint.

```
nebula> CREATE TAG player(name string NOT NULL, age int);
Execution succeeded (time spent 5001/5980 us)
```

The property `name` is `NOT NULL`. The property `age` is `NULL` by default.

```
nebula> SHOW CREATE TAG player;
+-----------+----------------------------------+
| Tag       | Create Tag                       |
+-----------+----------------------------------+
| "student" | "CREATE TAG `player` (           |
|           |  `name` string NOT NULL,         |
|           |  `age` int64 NULL                |
|           | ) ttl_duration = 0, ttl_col = """ |
+-----------+----------------------------------+
```

```
nebula> INSERT VERTEX player(name, age) VALUES "Kobe":("Kobe",null);
Execution succeeded (time spent 6367/7357 us)
```

Last update: March 25, 2021

4.2.6 Lists

The list is a composite data type. A list is a sequence of values. Individual list elements can be accessed by their positions.

A list starts with a left square bracket `[` and ends with a right square bracket `]`. A list contains zero, one, or more expressions. List elements are separated from each other with commas (`,`). Whitespace around elements is ignored in list, thus line breaks, tab stops, and blanks can be used for formatting.

Examples

```
nebula> RETURN [1, 2, 3] AS List;
+-----------+
| List      |
+-----------+
| [1, 2, 3] |
+-----------+

nebula> RETURN range(1,5)[3];
+---------------+
| range(1,5)[3] |
+---------------+
| 4             |
+---------------+

nebula> RETURN range(1,5)[-2];
+------------------+
| range(1,5)[-(2)] |
+------------------+
| 4                |
+------------------+

nebula> RETURN [n IN range(1,5) WHERE n > 2] AS a;
+-----------+
| a         |
+-----------+
| [3, 4, 5] |
+-----------+

nebula> RETURN [n IN range(1,5) WHERE n > 2 | n + 10] AS a;
+--------------+
| a            |
+--------------+
| [13, 14, 15] |
+--------------+

nebula> RETURN [n IN range(1,5) | n + 10] AS a;
+----------------------+
| a                    |
+----------------------+
| [11, 12, 13, 14, 15] |
+----------------------+

nebula> RETURN tail([n IN range(1, 5) | 2 * n - 10]) AS a;
+-----------------+
| a               |
+-----------------+
| [-6, -4, -2, 0] |
+-----------------+

nebula> RETURN [n IN range(1, 3) WHERE true | n] AS r;
+-----------+
| r         |
+-----------+
| [1, 2, 3] |
+-----------+

nebula> GO FROM "player100" OVER follow WHERE follow.degree NOT IN [x IN [92, 90] | x + $$.player.age] \
        YIELD follow._dst AS id, follow.degree AS degree;
+-------------+--------+
| id          | degree |
+-------------+--------+
| "player101" | 95     |
+-------------+--------+
| "player102" | 90     |
+-------------+--------+

nebula> MATCH p = (n:player{name:"Tim Duncan"})-[:follow]->(m) \
        RETURN [n IN nodes(p) | n.age + 100] AS r;
+------------+
| r          |
+------------+
| [142, 136] |
+------------+
| [142, 133] |
+------------+
```

```
nebula> RETURN size([1,2,3]);
+---------------+
| size([1,2,3]) |
+---------------+
| 3             |
+---------------+
```

OpenCypher compatibility

- A composite data type (i.e., set, map, and list) **CAN NOT** be stored as properties for vertices or edges.
- Use the range() function to return the range of a list.

```
nebula> RETURN range(0,5)[0..3];
[ERROR (-7)]: SyntaxError: syntax error near `3]'
```

- In openCypher, out-of-bound single elements returns `null`. However, in nGQL, out-of-bound single elements returns `OUT_OF_RANGE`.

```
nebula> RETURN range(0,5)[-12];
+-------------------+
| range(0,5)[-(12)] |
+-------------------+
| OUT_OF_RANGE      |
+-------------------+
```

Last update: March 16, 2021

4.2.7 Sets

Set is a composite data type.

OpenCypher compatibility

Set is not a data type in openCypher. The behavior of set in nGQL is not determined yet.

Last update: April 22, 2021

4.2.8 Maps

Map is a composite data type. A composite data type cannot be stored as properties. Maps are unordered collections of key-value pairs. In maps, the key is a string. The value can have any data type. You can get the map element by using `map['key']`.

Literal maps

```
nebula> YIELD {key: 'Value', listKey: [{inner: 'Map1'}, {inner: 'Map2'}]}
+------------------------------------------------------------+
| {key:Value,listKey:[{inner:Map1},{inner:Map2}]}            |
+------------------------------------------------------------+
| {key: "Value", listKey: [{inner: "Map1"}, {inner: "Map2"}]} |
+------------------------------------------------------------+
```

OpenCypher compatibility

- A composite data type (i.e. set, map, and list) CANNOT be stored as properties of vertices or edges.
- Map projection is not supported.

Last update: March 16, 2021

4.2.9 Type Conversion/Type coercions

Converting an expression of a given type to another type is known as type conversion.

Legacy version compatibility

- NGQL 1.0 adopted the `C`-style of type conversion (implicitly or explicitly). `(type_name)expression`. For example, The results of `YIELD (int)(TRUE)` is `1`. But it is error-prone to users who are not familiar with C language.
- NGQL 2.0 chooses the openCypher way of type coercions.

Type coercions functions

Function	Description
toBoolean()	Converts a string value to a boolean value.
toFloat()	Converts an integer or string value to a floating point number.
toInteger()	Converts a floating point or string value to an integer value.
type()	Returns the string representation of the relationship type.

Examples

```
nebula> UNWIND [true, false, 'true', 'false', NULL] AS b RETURN toBoolean(b) AS b
+----------+
| b        |
+----------+
| true     |
+----------+
| false    |
+----------+
| true     |
+----------+
| false    |
+----------+
| __NULL__ |
+----------+

nebula> RETURN toFloat(1), toFloat('1.3'), toFloat('1e3'), toFloat('not a number')
+------------+----------------+----------------+-------------------------+
| toFloat(1) | toFloat("1.3") | toFloat("1e3") | toFloat("not a number") |
+------------+----------------+----------------+-------------------------+
| 1.0        | 1.3            | 1000.0         | __NULL__                |
+------------+----------------+----------------+-------------------------+

nebula> RETURN toInteger(1), toInteger('1'), toInteger('1e3'), toInteger('not a number')
+--------------+----------------+------------------+--------------------------+
| toInteger(1) | toInteger("1") | toInteger("1e3") | toInteger("not a number") |
+--------------+----------------+------------------+--------------------------+
| 1            | 1              | 1000             | __NULL__                 |
+--------------+----------------+------------------+--------------------------+

nebula> MATCH (a:player)-[e]-() RETURN type(e)
+----------+
| type(e)  |
+----------+
| "follow" |
+----------+
| "follow" |

nebula> MATCH (a:player {name: "Tim Duncan"}) WHERE toInteger(id(a)) == 100 RETURN a
+-----------------------------------------------+
| a                                             |
+-----------------------------------------------+
| ("100" :player{age: 42, name: "Tim Duncan"}) |
+-----------------------------------------------+

nebula> MATCH (n:player) WITH n LIMIT toInteger(ceil(1.8)) RETURN count(*) AS count
+-------+
| count |
+-------+
| 2     |
+-------+
```

Last update: March 25, 2021

4.3 Variables and composite queries

4.3.1 Composite queries (clause structure)

Composite queries put data from different queries together. They then use filters, group-bys, or sorting before returning the combined return results. A composite query retrieves multiple levels of related information on existing queries and presents data as a single return result.

Nebula Graph supports three methods to compose queries (or sub-queries):

- (OpenCypher style) Clauses are chained together, and they feed intermediate result sets between each other.
- (nGQL extension) More than one queries can be batched together, separated by semicolons (;). The result of the last query is returned as the result of the batch.
- (nGQL extension) Queries can be piped together by using the pipe operator (`|`). The result of the previous query can be used as the input of the next query.

OpenCypher compatibility

In a composite query, choose the openCypher-style or nGQL-extension. **NOT BOTH**.

For example, if you're in the openCypher way (`MATCH` , `RETURN` , `WITH` , etc), don't introduce any pipe or semicolons to combine the sub-clauses.

If you're in the nGQL-extension way (`FETCH` , `GO` , `LOOKUP` , etc), you must use pipe or semicolons to combine the sub-clauses.

Further more, don't put together openCypher and nGQL-extension clauses in one statement. E.g., This statement is undefined: `MATCH ... | GO ... | YIELD ...` .

Composite queries are not `transactional` queries (as in SQL/Cypher)

For example, a query composed of three sub-queries: `A B C` , `A | B | C` or `A; B; C` . In that A is a read operation, B is a computation operation, and C is a write operation. If any part fails in the execution, the whole result is undefined. There is no rollback. What is written depends on the query executor.

> **Note**
>
> OpenCypher has no requirement of `transaction` .

Examples

- OpenCypher style

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})--() \
        WITH nodes(p) AS n \
        UNWIND n AS n1 \
        RETURN DISTINCT n1;
```

- Semicolon queries

```
nebula> SHOW TAGS; SHOW EDGES;   // Only edges are shown.

nebula> INSERT VERTEX player(name, age) VALUES "player100":("Tim Duncan", 42); \
INSERT VERTEX player(name, age) VALUES "player101":("Tony Parker", 36); \
INSERT VERTEX player(name, age) VALUES "player102":("LaMarcus Aldridge", 33);
// Multiple vertices are inserted in a composite statement.
```

- Pipe queries

```
nebula> GO FROM "player100" OVER follow YIELD follow._dst AS id | \
    GO FROM $-.id OVER serve YIELD $$.team.name AS Team, \
    $^.player.name AS Player;
    +---------+-------------+
    | Team    | Player      |
    +---------+-------------+
    | Nuggets | Tony Parker |
    +---------+-------------+
```

Last update: April 22, 2021

4.3.2 User-defined variables

User-defined variables allows passing the result of one statement to another.

OpenCypher variables

In openCypher, when you refer to a variable of vertex, edge or path, you need to name it first. The name you give to the pattern is a variable. For example:

```
nebula> MATCH (v:player{name:"Tim Duncan"}) RETURN v;
+-------------------------------------------------+
| v                                               |
+-------------------------------------------------+
| ("player100" :player{name: "Tim Duncan", age: 42}) |
+-------------------------------------------------+
```

The user-defined variable in the preceding query is `v`.

nGQL extensions

User-defined variables are written as `$var_name`. The `var_name` consists of letter, number or underline characters. Any other characters are not permitted.

User-defined variables can only be used in one execution. For example, you can use user-defined variables in composite queries separated by semicolon `;` or pipe `|`. Details about composite queries, see Composite queries.

> **Note**
>
> A user-defined variable is valid only at the current session and execution.

A user-defined variable in one statement **CANNOT** be used in either other clients or other executions. The statement that defines the user-defined variable and the statement that uses it must be submitted together. When this session ends, the user-defined variable is automatically expired.

> **Note**
>
> User-defined variables are case-sensitive.

Example

```
nebula> $var = GO FROM "player100" OVER follow YIELD follow._dst AS id; \
GO FROM $var.id OVER serve YIELD $$.team.name AS Team, \
$^.player.name AS Player;
+---------+-------------+
| Team    | Player      |
+---------+-------------+
| Nuggets | Tony Parker |
+---------+-------------+
```

Last update: May 20, 2021

4.3.3 Property reference

This page applies to nGQL extensions only.

You can refer to the properties of a vertex or an edge in `WHERE` or `YIELD` syntax.

Property reference for vertex

FOR SOURCE VERTEX

```
$^.tag_name.prop_name
```

`$^` is used to get the property of the source vertex, `tag_name` is the `tag` of the vertex, and `prop_name` specifies the property name.

FOR DESTINATION VERTEX

```
$$.tag_name.prop_name
```

`$$` is used to get the property of the destination vertex, `tag_name` is the `tag` of the vertex, and `prop_name` specifies the property name.

Property reference for edge

FOR PROPERTY

Use the following syntax to get the property of an edge.

```
edge_type.prop_name
```

`edge_type` is the edge type of the edge, and `prop_name` specifies the property name.

FOR BUILT-IN PROPERTIES

There are four built-in properties in each edge:

- `_src` : source vertex ID of the edge
- `_dst` : destination vertex ID of the edge
- `_type` : edge type
- `_rank` : the rank value for the edge

You can use `_src` and `_dst` to get the starting and ending vertices' ID, and they are very commonly used to show a graph path.

Examples

```
nebula> GO FROM "player100" OVER follow YIELD $^.player.name AS startName, $$.player.age AS endAge;
+--------------+--------+
| startName    | endAge |
+--------------+--------+
| "Tim Duncan" | 36     |
+--------------+--------+
| "Tim Duncan" | 33     |
+--------------+--------+
```

The preceding query returns the `name` property of the source vertex and the `age` property of the destination vertex.

```
nebula> GO FROM "player100" OVER follow YIELD follow.degree;
+---------------+
| follow.degree |
+---------------+
| 95            |
+---------------+
| 90            |
+---------------+
```

The preceding query returns the `degree` property of the edge.

```
nebula> GO FROM "player100" OVER follow YIELD follow._src, follow._dst, follow._type, follow._rank;
+-------------+-------------+--------------+--------------+
| follow._src | follow._dst | follow._type | follow._rank |
+-------------+-------------+--------------+--------------+
| "player100" | "player101" | 136          | 0            |
+-------------+-------------+--------------+--------------+
| "player100" | "player102" | 136          | 0            |
+-------------+-------------+--------------+--------------+
```

The preceding query returns all the neighbors of vertex `"player100"` over the `follow` edges, by referencing `follow._src` as the source vertex ID (which is `"player100"`) and `follow._dst` as the destination vertex ID.

Last update: March 23, 2021

4.4 Operators

4.4.1 Comparison operators

Name	Description
`=`	Assign a value
`/`	Division operator
`==`	Equal operator
`!=`, `<>`	Not equal operator
`<`	Less than operator
`<=`	Less than or equal operator
`-`	Minus operator
`%`	Modulo operator
`+`	Addition operator
`*`	Multiplication operator
`-`	Change the sign of the argument
`IS NULL`	NULL check
`IS NOT NULL`	not NULL check

Comparison operations result in a value of *true* and *false*.

Note

Comparability between values of different types is often undefined. The result could be NULL or others.

OpenCypher compatibility

Comparing with NULL is different from openCypher. The behavior may change. `IS [NOT] NULL` is often used with `OPTIONAL MATCH`. But `OPTIONAL MATCH` is not support in nGQL.

- `==`

Equal. String comparisons are case-sensitive. Values of different types are not equal.

Note

The equality operator is `==` in nGQL and is `=` in openCypher.

```
nebula>  RETURN 'A' == 'a', toUpper('A') == toUpper('a'), toLower('A') == toLower('a')
+------------+-----------------------------+-----------------------------+
| ("A"=="a") | (toUpper("A")==toUpper("a")) | (toLower("A")==toLower("a")) |
+------------+-----------------------------+-----------------------------+
| false      | true                        | true                        |
+------------+-----------------------------+-----------------------------+

nebula> RETURN '2' == 2, toInteger('2') == 2;
+----------+--------------------+
| ("2"==2) | (toInteger("2")==2) |
+----------+--------------------+
```

```
| false    | true               |
+----------+--------------------+
```

- `>`

Greater than:

```
nebula> RETURN 3 > 2;
+-------+
| (3>2) |
+-------+
| true  |
+-------+

nebula>  WITH 4 AS one, 3 AS two RETURN one > two AS result;
+--------+
| result |
+--------+
| true   |
+--------+
```

- `>=`

Greater than or equal to:

```
nebula>  RETURN 2 >= "2", 2 >= 2
+----------+--------+
| (2>="2") | (2>=2) |
+----------+--------+
| __NULL__ | true   |
+----------+--------+
```

- `<`

Less than:

```
nebula> YIELD 2.0 < 1.9;
+---------+
| (2<1.9) |
+---------+
| false   |
+---------+
```

- `<=`

Less than or equal to:

```
nebula> YIELD 0.11 <= 0.11;
+--------------+
| (0.11<=0.11) |
+--------------+
| true         |
+--------------+
```

- `!=`

Not equal:

```
nebula> YIELD 1 != '1';
+--------+
| (1!=1) |
+--------+
| true   |
+--------+
```

- `IS [NOT] NULL`

```
nebula> RETURN null IS NULL AS value1, null == null AS value2, null != null AS value3
+--------+----------+----------+
| value1 | value2   | value3   |
+--------+----------+----------+
| true   | __NULL__ | __NULL__ |
+--------+----------+----------+

nebula> RETURN length(NULL), size(NULL), count(NULL), NULL IS NULL, NULL IS NOT NULL, sin(NULL), NULL + NULL, [1, NULL] IS NULL
+-------------+------------+-------------+--------------+------------------+-----------+-------------+-----------------+
| length(NULL) | size(NULL) | COUNT(NULL) | NULL IS NULL | NULL IS NOT NULL | sin(NULL) | (NULL+NULL) | [1,NULL] IS NULL |
+-------------+------------+-------------+--------------+------------------+-----------+-------------+-----------------+
| BAD_TYPE     | __NULL__   | 0           | true         | false            | BAD_TYPE  | __NULL__    | false           |
```

```
+-------------+------------+-------------+-------------+-----------------+-----------+-------------+-----------------+

nebula> WITH {name: null} AS map RETURN map.name IS NOT NULL
+----------------------+
| map.name IS NOT NULL |
+----------------------+
| false                |
+----------------------+

nebula> WITH {name: 'Mats', name2: 'Pontus'} AS map1, \
       {name: null} AS map2, {notName: 0, notName2: null } AS map3 \
       RETURN map1.name IS NULL, map2.name IS NOT NULL, map3.name IS NULL
+-------------------+-----------------------+-------------------+
| map1.name IS NULL | map2.name IS NOT NULL | map3.name IS NULL |
+-------------------+-----------------------+-------------------+
| false             | false                 | true              |
+-------------------+-----------------------+-------------------+

nebula> MATCH (n:player) RETURN n.age IS NULL, n.name IS NOT NULL, n.empty IS NULL
+---------------+-------------------+-----------------+
| n.age IS NULL | n.name IS NOT NULL | n.empty IS NULL |
+---------------+-------------------+-----------------+
| false         | true              | true            |
+---------------+-------------------+-----------------+
| false         | true              | true            |
+---------------+-------------------+-----------------+
| false         | true              | true            |
+---------------+-------------------+-----------------+
...
```

Last update: April 22, 2021

4.4.2 Boolean operators

Name	Description
AND	Logical AND
NOT	Logical NOT
OR	Logical OR
XOR	Logical XOR

For the precedence of the operators, refer to Operator Precedence.

For the logical operations with NULL, refer to NULL.

Legacy version compatibility

- In Nebula Graph 1.0, non-zero numbers are evaluated to *true* like c-language.
- In Nebula Graph 2.0, non-zero numbers can't be converted to boolean values.

Last update: March 25, 2021

4.4.3 Pipe operator

OpenCypher compatibility

This page applies to nGQL extensions only.

Syntax

One major difference between nGQL and SQL is how sub-queries are composed.

In SQL, to form a statement, sub-queries are nested (embedded). In nGQL the shell style `PIPE (|)` is introduced.

Examples

```
nebula> GO FROM "player100" OVER follow \
YIELD follow._dst AS dstid, $$.player.name AS Name |\
GO FROM $-.dstid OVER follow;

+-------------+
| follow._dst |
+-------------+
| "player101" |
+-------------+
```

If there is no `YIELD` clause to define the output, the destination vertex ID is returned by default. If a YIELD clause is applied, the output is defined by the YIELD clause.

You must define aliases in the `YIELD` clause for the reference operator `$-` to use, just like `$-.dstid` in the preceding example.

Last update: March 23, 2021

4.4.4 Reference operators

NGQL provides reference operators to represent a property in a `WHERE` or `YIELD` clause, or the output of the statement before the pipe symbol in a composite query.

OpenCypher compatibility

This page applies to nGQL extensions only.

Reference operator List

Reference operator	Description
`$^`	Refers to a source vertex property. For more information, see Property reference.
`$$`	Refers to a destination vertex property. For more information, see Property reference.
`$-`	Refers to the output of the statement before the pipe symbol in a composite query. For more information, see Pipe.

Examples

The following example returns the age of the source vertex and the destination vertex.

```
nebula> GO FROM "player100" OVER follow \
        YIELD $^.player.age AS SrcAge, $$.player.age AS DestAge;
+--------+---------+
| SrcAge | DestAge |
+--------+---------+
| 42     | 36      |
+--------+---------+
| 42     | 41      |
+--------+---------+
```

The following example returns the name and team of the players that "player100" follows.

```
nebula> GO FROM "player100" OVER follow \
        YIELD follow._dst AS id | \
        GO FROM $-.id OVER serve \
        YIELD $^.player.name AS Player, $$.team.name AS Team;
+-----------------+-----------+
| Player          | Team      |
+-----------------+-----------+
| "Tony Parker"   | "Spurs"   |
+-----------------+-----------+
| "Tony Parker"   | "Hornets" |
+-----------------+-----------+
| "Manu Ginobili" | "Spurs"   |
+-----------------+-----------+
```

Last update: March 23, 2021

4.4.5 Set operations

OpenCypher compatibility

This page applies to nGQL extensions only.

Syntax

This document descriptions the set operations, including `UNION`, `UNION ALL`, `INTERSECT`, and `MINUS`. To combine multiple queries, use the set operators.

All set operators have equal precedence. If a nGQL statement contains multiple set operators, Nebula Graph evaluates them from the left to right unless parentheses explicitly specify another order.

To use the set operators, always match the return results of the `GO` clause with the same number and data type.

UNION, UNION DISTINCT, and UNION ALL

```
<left> UNION [DISTINCT | ALL] <right> [ UNION [DISTINCT | ALL] <right> ...]
```

Operator `UNION DISTINCT` (or by short `UNION`) returns the union of two sets A and B without the duplicate elements.

Operator `UNION ALL` returns the union of two sets A and B with duplicated elements.

The `<left>` and `<right>` must have the same number of columns and data types. Different data types are converted according to the Type Conversion.

EXAMPLE

The following statement

```
nebula> GO FROM "player102" OVER follow \
    UNION \
    GO FROM "player100" OVER follow;
+-------------+
| follow._dst |
+-------------+
| "player101" |
+-------------+
| "player102" |
+-------------+
```

returns the neighbors' id of vertex `"player102"` and `"player100` (along with edge `follow`) without duplication.

While

```
nebula> GO FROM "player102" OVER follow \
    UNION ALL \
    GO FROM "player100" OVER follow;
+-------------+
| follow._dst |
+-------------+
| "player101" |
+-------------+
| "player101" |
+-------------+
| "player102" |
+-------------+
```

returns all the neighbors of vertex `"player102"` and `"player100`, with all possible duplications.

`UNION` can also work with the `YIELD` statement. For example, let's suppose the results of the following two queries.

```
nebula> GO FROM "player102" OVER follow YIELD follow._dst AS id, follow.degree AS Degree, $$.player.age AS Age; -- query 1

+-------------+--------+-----+
| id          | Degree | Age |
+-------------+--------+-----+
| "player101" | 75     | 36  |         -- line 1
+-------------+--------+-----+
```

```
nebula> GO FROM "player100" OVER follow YIELD follow._dst AS id, follow.degree AS Degree, $$.player.age AS Age;  -- query 2

+-------------+--------+-----+
| id          | Degree | Age |
+-------------+--------+-----+
| "player101" | 96     | 36  |          -- line 2
+-------------+--------+-----+
| "player102" | 90     | 33  |          -- line 3
+-------------+--------+-----+
```

And the following statement

```
nebula> GO FROM "player102" OVER follow YIELD follow._dst AS id, follow.degree AS Degree, $$.player.age AS Age \
    UNION /* DISTINCT */ \
    GO FROM "player100" OVER follow YIELD follow._dst AS id, follow.degree AS Degree, $$.player.age AS Age;
```

returns the follows:

```
+-------------+--------+-----+
| id          | Degree | Age |
+-------------+--------+-----+
| "player101" | 75     | 36  |        -- line 1
+-------------+--------+-----+
| "player101" | 96     | 36  |        -- line 2
+-------------+--------+-----+
| "player102" | 90     | 33  |        -- line 3
+-------------+--------+-----+
```

The `DISTINCT` check duplication by all the columns for every line. So line 1 and line 2 are different.

INTERSECT

```
<left> INTERSECT <right>
```

Operator `INTERSECT` returns the intersection of two sets A and B (denoted by A ⋂ B).

Similar to `UNION`, the `<left>` and `<right>` must have the same number of columns and data types. Only the `INTERSECT` columns of `<left>` and `<right>` are returned.

For example, the following query

```
nebula> GO FROM "player102" OVER follow YIELD follow._dst AS id, follow.degree AS Degree, $$.player.age AS Age \
INTERSECT \
GO FROM "player100" OVER follow YIELD follow._dst AS id, follow.degree AS Degree, $$.player.age AS Age;
```

returns

```
Empty set (time spent 5194/6264 us)
```

MINUS

```
<left> MINUS <right>
```

Operator `MINUS` returns the subtraction (or difference) of two sets A and B (denoted by A - B). Always pay attention to the order of the `<left>` and `<right>`. The set A - B consists of elements that are in A but not in B.

For example, the following query

```
nebula> GO FROM "player100" OVER follow \
MINUS \
GO FROM "player102" OVER follow;
```

returns

```
+-------------+
| follow._dst |
+-------------+
| "player102" |
+-------------+
```

If you reverse the `MINUS` order, the query

```
nebula> GO FROM "player102" OVER follow \
MINUS \
GO FROM "player100" OVER follow;
```

returns

```
Empty set (time spent 2243/3259 us)
```

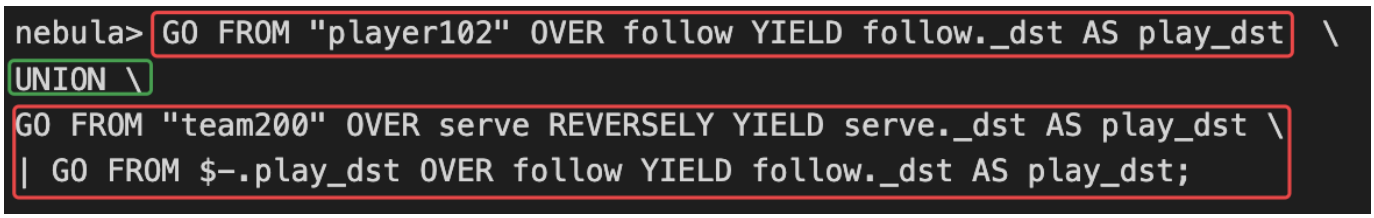
Precedence of the SET Operations and Pipe

Please note that when a query contains pipe `|` and set operations, pipe takes precedence. Refer to the Pipe Doc for details. Query `GO FROM 1 UNION GO FROM 2 | GO FROM 3` is the same as query `GO FROM 1 UNION (GO FROM 2 | GO FROM 3)`.

For example:

```
nebula> GO FROM "player102" OVER follow YIELD follow._dst AS play_dst  \
UNION \
GO FROM "team200" OVER serve REVERSELY YIELD serve._dst AS play_dst \
| GO FROM $-.play_dst OVER follow YIELD follow._dst AS play_dst;

+-------------+
| play_dst    |
+-------------+
| "player101" |
+-------------+
| "player102" |
+-------------+
```



The statements in the red bar are executed first. And then the statement in the green box is executed.

```
nebula> (GO FROM "player102" OVER follow YIELD follow._dst AS play_dst  \
UNION \
GO FROM "team200" OVER serve REVERSELY YIELD serve._dst AS play_dst) \
| GO FROM $-.play_dst OVER follow YIELD follow._dst AS play_dst;
```

In the above query, the parentheses change the execution priority, and the statements within the parentheses take the precedence.

Last update: March 17, 2021

4.4.6 String operators

Name	Description
+	concatenating strings
CONTAINS	Perform case-sensitive inclusion searching in strings
(NOT) IN	Whether a value is within a set of values
(NOT) STARTS WITH	Perform case-sensitive matching on the beginning of a string
(NOT) ENDS WITH	Perform case-sensitive matching on the ending of a string
Regular expressions	Perform regular expression matching on a string

Note

All the string matchings are case-sensitive.

Examples

- concatenation (+)

```
nebula> RETURN 'a' + 'b';
+-------+
| (a+b) |
+-------+
| "ab"  |
+-------+
nebula> UNWIND 'a' AS a UNWIND 'b' AS b RETURN a + b;
+-------+
| (a+b) |
+-------+
| "ab"  |
+-------+
```

- CONTAINS

The `CONTAINS` operator requires string type in both left and right side.

```
nebula> MATCH (s:player)-[e:serve]->(t:team) WHERE id(s) == "player101" \
    AND t.name CONTAINS "ets" RETURN s.name, e.start_year, e.end_year, t.name;
+---------------+-------------+------------+-----------+
| s.name        | e.start_year | e.end_year | t.name    |
+---------------+-------------+------------+-----------+
| "Tony Parker" | 2018         | 2019       | "Hornets" |
+---------------+-------------+------------+-----------+

nebula> GO FROM "player101" OVER serve WHERE (STRING)serve.start_year CONTAINS "19" AND \
    $^.player.name CONTAINS "ny" \
    YIELD $^.player.name, serve.start_year, serve.end_year, $$.team.name;
+----------------+-----------------+---------------+--------------+
| $^.player.name | serve.start_year | serve.end_year | $$.team.name |
+----------------+-----------------+---------------+--------------+
| "Tony Parker"  | 1999             | 2018          | "Spurs"      |
+----------------+-----------------+---------------+--------------+

nebula> GO FROM "player101" OVER serve WHERE !($$.team.name CONTAINS "ets") \
    YIELD $^.player.name, serve.start_year, serve.end_year, $$.team.name;
+----------------+-----------------+---------------+--------------+
| $^.player.name | serve.start_year | serve.end_year | $$.team.name |
+----------------+-----------------+---------------+--------------+
| "Tony Parker"  | 1999             | 2018          | "Spurs"      |
+----------------+-----------------+---------------+--------------+
```

- IN

```
nebula> RETURN  1 IN [1,2,3], "Yao" IN ["Yi", "Tim", "Kobe"], NULL in ["Yi", "Tim", "Kobe"]
+---------------+-------------------------------+-------------------------------+
| (1 IN [1,2,3]) | ("Yao" IN ["Yi","Tim","Kobe"]) | (NULL IN ["Yi","Tim","Kobe"]) |
+---------------+-------------------------------+-------------------------------+
```

```
| true           | false                         | false                        |
+----------------+-------------------------------+------------------------------+
```

- (NOT) STARTS WITH

```
nebula> RETURN 'apple' STARTS WITH 'app', 'apple' STARTS WITH 'a', 'apple' STARTS WITH toUpper('a')
+-----------------------------+---------------------------+-----------------------------------+
| ("apple" STARTS WITH "app") | ("apple" STARTS WITH "a") | ("apple" STARTS WITH toUpper("a")) |
+-----------------------------+---------------------------+-----------------------------------+
| true                        | true                      | false                             |
+-----------------------------+---------------------------+-----------------------------------+

nebula> RETURN 'apple' STARTS WITH 'b','apple' NOT STARTS WITH 'app'
+---------------------------+---------------------------------+
| ("apple" STARTS WITH "b") | ("apple" NOT STARTS WITH "app") |
+---------------------------+---------------------------------+
| false                     | false                           |
+---------------------------+---------------------------------+
```

- (NOT) ENDS WITH

```
nebula> RETURN 'apple' ENDS WITH 'app', 'apple' ENDS WITH 'e', 'apple' ENDS WITH 'E', 'apple' ENDS WITH 'b'
+---------------------------+-------------------------+-------------------------+-------------------------+
| ("apple" ENDS WITH "app") | ("apple" ENDS WITH "e") | ("apple" ENDS WITH "E") | ("apple" ENDS WITH "b") |
+---------------------------+-------------------------+-------------------------+-------------------------+
| false                     | true                    | false                   | false                   |
+---------------------------+-------------------------+-------------------------+-------------------------+
```

- Regular expressions

Nebula Graph supports filtering by using regular expressions. The regular expression syntax is inherited from `std::regex`. You can match on regular expressions by using `=~ 'regexp'`. For example:

```
nebula> RETURN "384748.39" =~ "\\d+(\\.\\d{2})?";
+----------------------------+
| (384748.39=~\d+(\.\d{2})?) |
+----------------------------+
| true                       |
+----------------------------+
```

```
nebula> MATCH (v:player) WHERE v.name =~ 'Tony.*' RETURN v.name;
+---------------+
| v.name        |
+---------------+
| "Tony Parker" |
+---------------+
```

Note

Regular expressions **CAN NOT** work with nGQL-extensions (GO/FETCH clause will return syntax error). Use it in openCypher only (e.g., in MATCH-WHERE clause).

Last update: April 22, 2021

4.4.7 List operators

List operators are:

- concatenating lists: `+`
- checking if an element exists in a list: `IN`
- accessing an element(s) in a list using the subscript operator: `[]`

Examples

```
nebula> YIELD [1,2,3,4,5]+[6,7] AS myList
+-----------------------+
| myList                |
+-----------------------+
| [1, 2, 3, 4, 5, 6, 7] |
+-----------------------+

nebula> RETURN size([NULL, 1, 2])
+------------------+
| size([NULL,1,2]) |
+------------------+
| 3                |
+------------------+

nebula> RETURN NULL IN [NULL, 1]
+--------------------+
| (NULL IN [NULL,1]) |
+--------------------+
| true               |
+--------------------+

nebula> WITH [2, 3, 4, 5] AS numberlist \
    UNWIND numberlist AS number \
    WITH number \
    WHERE number IN [2, 3, 8] \
    RETURN number
+--------+
| number |
+--------+
| 2      |
+--------+
| 3      |
+--------+
```

Last update: March 17, 2021

4.4.8 Operator precedence

The following list shows the precedence of nGQL operators in descending order. Operators that are shown together on a line have the same precedence.

```
- (negative number)
!, NOT
*, /, %
-, +
== , >=, >, <=, <, <>, !=
AND
OR, XOR
= (assignment)
```

For operators that occur at the same precedence level within an expression, evaluation proceeds left to right, with the exception that assignments evaluate right to left.

The precedence of operators determines the order of evaluation of terms in an expression. To override this order and group terms explicitly, use parentheses.

Examples

```
nebula> RETURN 2+3*5;
+-----------+
| (2+(3*5)) |
+-----------+
| 17        |
+-----------+

nebula> RETURN (2+3)*5;
+-----------+
| ((2+3)*5) |
+-----------+
| 25        |
+-----------+
```

OpenCypher compatibility

In openCypher, comparisons can be chained arbitrarily, e.g., `x < y <= z` is equivalent to `x < y AND y <= z` in openCypher. But in nGQL, it is equivalent to `(x < y) <= z`, which is a boolean (x < y) compare again an integer (z). And the result is NULL.

Last update: March 17, 2021

4.5 Functions and expressions

4.5.1 Built-in math functions

Nebula Graph supports the following built-in math functions:

Function	Description
double abs(double x)	Returns absolute value of the argument.
double floor(double x)	Returns the largest integer value smaller than or equal to the argument. (Rounds down)
double ceil(double x)	Returns the smallest integer greater than or equal to the argument. (Rounds up)
double round(double x)	Returns the integer value nearest to the argument. Returns a number farther away from 0 if the argument is in the middle.
double sqrt(double x)	Returns the square root of the argument.
double cbrt(double x)	Returns the cubic root of the argument.
double hypot(double x, double y)	Returns the hypotenuse of a right-angled triangle.
double pow(double x, double y)	Returns the result of `x` raised by the `y` th power.
double exp(double x)	Returns the value of e raised to the x power.
double exp2(double x)	Returns 2 raised to the argument.
double log(double x)	Returns natural logarithm of the argument.
double log2(double x)	Returns the base-2 logarithm of the argument.
double log10(double x)	Returns the base-10 logarithm of the argument.
double sin(double x)	Returns sine of the argument.
double asin(double x)	Returns inverse sine of the argument.
double cos(double x)	Returns cosine of the argument.
double acos(double x)	Returns inverse cosine of the argument.
double tan(double x)	Returns tangent of the argument.
double atan(double x)	Returns inverse tangent the argument.
double rand()	Returns a random floating point number in the range from 0 (inclusive) to 1 (exclusive); i.e.[0,1).
int rand32(int min, int max)	Returns a random 32-bit integer in [min, max). If you set only one argument, it is parsed as `max` and `min` is default to `0`. If you set no argument, the system returns a random signed 32-bit integer.
int rand64(int min, int max)	Returns a random 64-bit integer in [min, max). If you set only one argument, it is parsed as `max` and `min` is default to `0`. If you set no argument, the system returns a random signed 64-bit integer.
collect()	Puts all the collected values to a list.
avg()	Returns the average value of the argument.
count()	Returns the number of records.
max()	Returns the maximum value.
min()	Returns the minimum value.
std()	Returns the population standard deviation.
sum()	Returns the sum value.
bit_and()	Bitwise AND.
bit_or()	Bitwise OR.
bit_xor()	Bitwise exclusive OR (XOR).

Function	Description
int size()	Returns the number of elements in a list or a map.
int range(int start, int end, int step)	Returns a list of integers from `start` (inclusive) to `end` (inclusive) in the specified steps. `step` is optional and default to 1.
int sign(double x)	Returns the signum of the given number: 0 if the number is 0, -1 for any negative number, and 1 for any positive number.
double e()	Returns the base of the natural logarithm, e (2.718281828459045).
double pi()	Returns the mathematical constant pi (3.141592653589793).
double radians()	Converts degrees to radians. `radians(180)` returns 3.141592653589793.

Note

If the argument is set to `NULL`, the output is undefined.

Last update: April 22, 2021

4.5.2 Built-in string functions

Nebula Graph supports the following built-in string functions:

Function	Description
int strcasecmp(string a, string b)	Compares strings without case sensitivity, when a = b, Returns 0, when a > b Returnsed value is greater than 0, otherwise less than 0.
string lower(string a)	Returns the argument in lowercase.
string toLower(string a)	The same as lower().
string upper(string a)	Returns the argument in uppercase.
string toUpper(string a)	The same as upper().
int length(string a)	Returns the length of given string in bytes.
string trim(string a)	Removes leading and trailing spaces.
string ltrim(string a)	Removes leading spaces.
string rtrim(string a)	Removes trailing spaces.
string left(string a, int count)	Returns the substring in [1, count], if length a is less than count, Returns a.
string right(string a, int count)	Returns the substring in [size - count + 1, size], if length a is less than count, Returns a.
string lpad(string a, int size, string letters)	Left-pads a string with another string to a certain length.
string rpad(string a, int size, string letters)	Reft-pads a string with another string to a certain length.
string substr(string a, int pos, int count)	Returns a substring from a string, starting at the specified position `pos`, extract `count` characters.
string substring(string a, int pos, int count)	The same as substr().
string reverse(string)	Returns the reverse of a string.
string replace(string a, string b, string c)	Replaces string b in string a with string c.
list split(string a, string b)	Splits string a at string b and returns a list of strings.
string toString()	Takes in any data type and converts it into a string.
int hash()	Takes in any data type and encodes it into an integer value.

> **Note**
>
> If the argument is `NULL`, the return is undefined.

Explanations for the return of `substr()` and `substring()`

- `pos` uses a 0-based index.
- If `pos` is 0, the whole string `a` is returned.
- If `pos` is greater than the maximum string index, an empty string is returned.
- If `pos` is a negative number, `BAD_DATA` is returned.

- If `count` is omitted, the function returns the substring starting at the position given by `pos` and extending to the end of string `a`.
- Using `NULL` as any of the argument of `substr()` causes an issue.
- If `count` is 0, an empty string is returned.

OpenCypher compatibility

- In openCypher, if `a` is `null`, `null` is returned.
- In openCypher, if `pos` is 0, the returned substring starts from the first character, and extend to `count` characters.
- In openCypher, if either `pos` or `count` is null or a negative integer, an error is raised.

Last update: April 22, 2021

4.5.3 Built-in date and time functions

Nebula Graph supports the following built-in date and time functions:

Function	Description
int now()	Return the current date and time of the system time zone.
date date()	Return the current UTC date based on the current system.
time time()	Return the current UTC calendar time of the current time zone.
datetime datetime()	Return the current UTC datetime based on the current time.

The date(), time(), and datetime() functions accept three kind of parameters, namely empty, string, and map.

Examples

```
> RETURN now(), date(), time(), datetime();
+------------+------------+--------------+-------------------------+
| now()      | date()     | time()       | datetime()              |
+------------+------------+--------------+-------------------------+
| 1611907165 | 2021-01-29 | 07:59:22.000 | 2021-01-29T07:59:22.000 |
+------------+------------+--------------+-------------------------+
```

OpenCypher compatibility

- Time in openCypher is measured in milliseconds.
- Time in nGQL is measured in seconds. The milliseconds are displayed in `000`.

Last update: March 29, 2021

4.5.4 Schema functions

Nebula Graph supports the following built-in schema functions:

Function	Description
id(vertex)	Returns the id of a vertex. The data type of the result is the same as the vertex ID.
list tags(vertex)	Returns the tags of a vertex.
list labels(vertex)	Returns the tags of a vertex.
map properties(vertex_or_edge)	Takes in a vertex or an edge and returns its properties.
string type(edge)	Returns the edge type of an edge.
vertex startNode(path)	Takes in an edge or a path and returns its source vertex ID.
string endNode(path)	Takes in an edge or a path and returns its destination vertex ID.
int rank(edge)	Returns the rank value of an edge.

Examples

```
nebula> MATCH (a:player) WHERE id(a) == "player100" RETURN tags(a), labels(a), properties(a)
+------------+------------+------------------------------+
| tags(a)    | labels(a)  | properties(a)                |
+------------+------------+------------------------------+
| ["player"] | ["player"] | {age: 42, name: "Tim Duncan"} |
+------------+------------+------------------------------+

nebula> MATCH p = (a :player {name : "Tim Duncan"})-[r:serve]-(t) RETURN type(r), rank(r)
+---------+---------+
| type(r) | rank(r) |
+---------+---------+
| "serve" | 0       |
+---------+---------+

nebula> MATCH p = (a :player {name : "Tim Duncan"})-[r:serve]-(t) RETURN startNode(p), endNode(p)
+--------------------------------------------------+---------------------------------+
| startNode(p)                                     | endNode(p)                      |
+--------------------------------------------------+---------------------------------+
| ("player100" :player{age: 42, name: "Tim Duncan"}) | ("team204" :team{name: "Spurs"}) |
+--------------------------------------------------+---------------------------------+
```

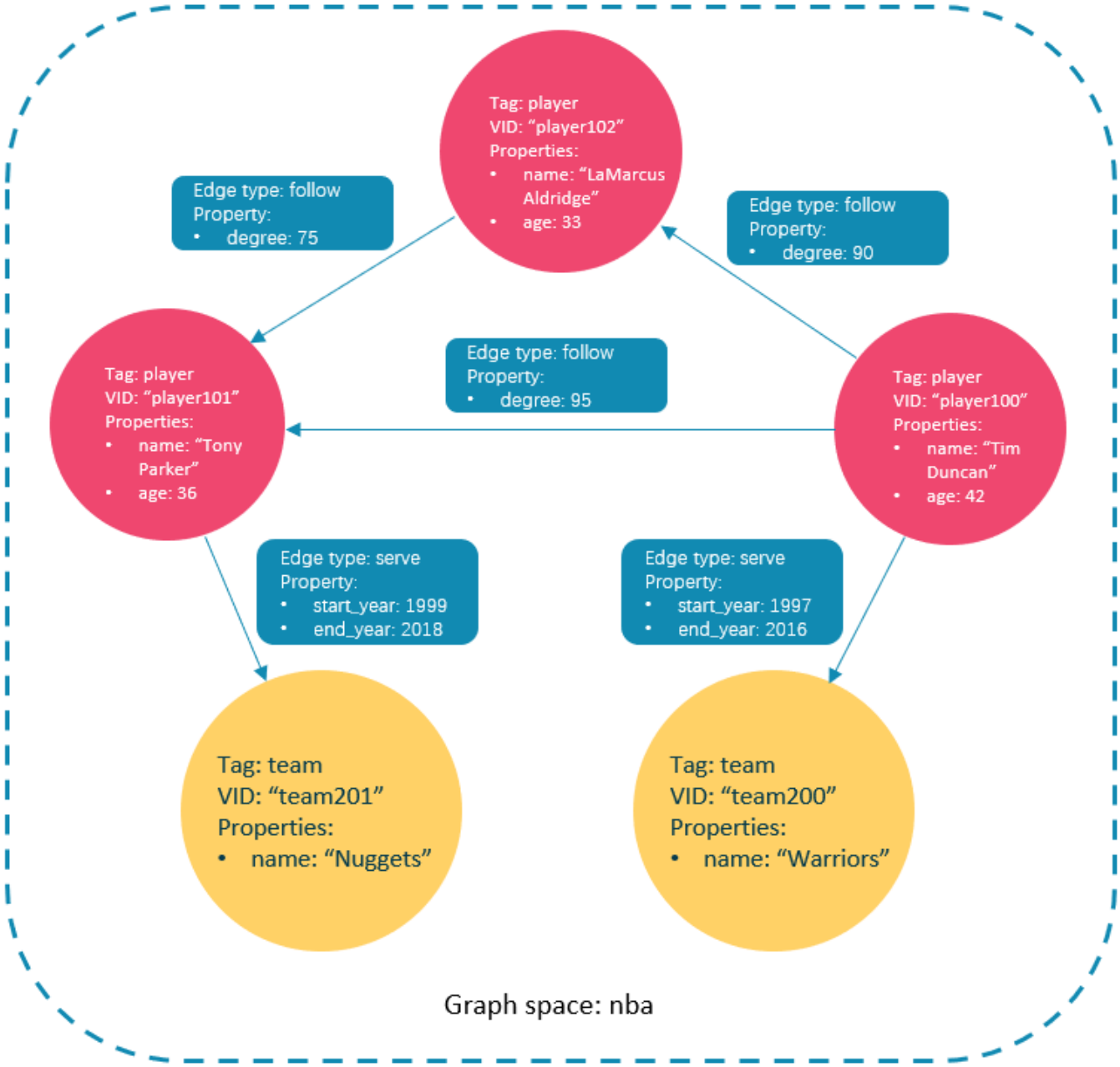
Last update: March 25, 2021

4.5.5 CASE expressions

The `CASE` expression uses conditions to filter the result of an nGQL query statement. It is usually used in the `YIELD` or `RETURN` clause. nGQL provides two forms of `CASE` expressions just like openCypher: the simple form and the generic form.

The `CASE` expression goes through conditions and returns a result when the first condition is met. Then the `CASE` expression stops reading the conditions and returns the result. If no conditions are met, it returns the result in the `ELSE` clause. If there is no `ELSE` clause and no conditions are met, it returns `NULL`.

The following graph is used for the examples in this topic.



The simple form of CASE expressions

SYNTAX

```
CASE <comparer>
WHEN <value> THEN <result>
[WHEN ...]
[ELSE <default>]
END
```

Caution

Always remember to end a `CASE` expression with `END`.

Parameters	Description
`comparer`	A value or a valid expression that outputs a value. This value is used to compare with `value`.
`value`	It will be compared with `comparer`. If they match, then this condition is met.
`result`	It is returned by the `CASE` expression if `value` matches `comparer`.
`default`	It is returned by the `CASE` expression if no conditions are met.

EXAMPLES

```
nebula> RETURN \
      CASE 2+3 \
      WHEN 4 THEN 0 \
      WHEN 5 THEN 1 \
      ELSE -1 \
      END \
      AS result;
+--------+
| result |
+--------+
| 1      |
+--------+
```

```
nebula> GO FROM "player100" OVER follow \
      YIELD $$.player.name AS Name, \
      CASE $$.player.age > 35 \
      WHEN true THEN "Yes" \
      WHEN false THEN "No" \
      ELSE "Nah" \
      END \
      AS Age_above_35;
+---------------------+--------------+
| Name                | Age_above_35 |
+---------------------+--------------+
| "Tony Parker"       | "Yes"        |
+---------------------+--------------+
| "LaMarcus Aldridge" | "No"         |
+---------------------+--------------+
```

The generic form of CASE expressions

SYNTAX

```
CASE
WHEN <condition> THEN <result>
[WHEN ...]
[ELSE <default>]
END
```

Parameters	Description
`condition`	If `condition` is evaluated as true, `result` is returned by the `CASE` expression.
`result`	It is returned by the `CASE` expression if `condition` is evaluated as true.
`default`	It is returned by the `CASE` expression if no conditions are met.

EXAMPLES

```
nebula> YIELD \
      CASE WHEN 4 > 5 THEN 0 \
      WHEN 3+4==7 THEN 1 \
```

```
      ELSE 2 \
      END \
      AS result;
+--------+
| result |
+--------+
| 1      |
+--------+
```

```
nebula> MATCH (v:player) WHERE v.age > 30 \
      RETURN v.name AS Name,  \
      CASE \
      WHEN v.name STARTS WITH "T" THEN "Yes" \
      ELSE "No" \
      END \
      AS Starts_with_T;
+---------------------+---------------+
| Name                | Starts_with_T |
+---------------------+---------------+
| "Tim"               | "Yes"         |
+---------------------+---------------+
| "LaMarcus Aldridge" | "No"          |
+---------------------+---------------+
| "Tony Parker"       | "Yes"         |
+---------------------+---------------+
```

Differences between the simple form and the generic form

To avoid the misuse of the simple form and the generic form, it is important to understand their differences. The following example can help explain them.

```
nebula> GO FROM "player100" OVER follow \
      YIELD $$.player.name AS Name, $$.player.age AS Age, \
      CASE $$.player.age \
      WHEN $$.player.age > 35 THEN "Yes" \
      ELSE "No" \
      END \
      AS Age_above_35;
+---------------------+-----+--------------+
| Name                | Age | Age_above_35 |
+---------------------+-----+--------------+
| "Tony Parker"       | 36  | "No"         |
+---------------------+-----+--------------+
| "LaMarcus Aldridge" | 33  | "No"         |
+---------------------+-----+--------------+
```

The preceding `GO` query is intended to output "Yes" when the player age is above 35. However, in this example, when the player age is 36, the actual output is not as expected: It is "No" instead of "Yes".

This is because the query uses the `CASE` expression in the simple form, and a comparison between the values of `$$.player.age` and `$$.player.age > 35` is made. When the player age is 36:

- The value of `$$.player.age` is `36`. It is an integer.
- `$$.player.age > 35` is evaluated to `true`. It is a boolean.

The values of `$$.player.age` and `$$.player.age > 35` do not match. This condition is not met and "No" is returned.

Last update: April 22, 2021

4.5.6 List functions

Function	Description
keys(expr)	Returns a list containing the string representations for all the property names of a vertex, edge, or map.
labels(vertex)	Returns the tags of a vertex.
nodes(path)	Returns a list containing all the nodes in a path.
range(start, end [, step])	A list of Integer elements.
relationships(path)	Returns a list containing all the relationships in a path.
reverse(list)	returns a list in which the order of all elements in the original list have been reversed.
tail(list)	returns all the elements, excluding the first one.
head(list)	Returns the first element of a list.
last(list)	Returns the last element of a list.
coalesce(list)	Returns the first not null value in a list.
reduce()	See reduce() function.

Note

If the parameter is `NULL`, the output is undefined.

Examples

```
nebula> WITH [NULL, 4923, 'abc', 521, 487] AS ids RETURN reverse(ids), tail(ids), head(ids), last(ids), coalesce(ids)
+----------------------------------+-------------------------+-----------+-----------+---------------+
| reverse(ids)                     | tail(ids)               | head(ids) | last(ids) | coalesce(ids) |
+----------------------------------+-------------------------+-----------+-----------+---------------+
| [487, 521, "abc", 4923, __NULL__] | [4923, "abc", 521, 487] | __NULL__  | 487       | 4923          |
+----------------------------------+-------------------------+-----------+-----------+---------------+

nebula> MATCH (a:player)-[r]->() WHERE id(a) == "player100" RETURN labels(a),  keys(r)
+------------+---------------------------+
| labels(a)  | keys(r)                   |
+------------+---------------------------+
| ["player"] | ["degree"]                |
+------------+---------------------------+
| ["player"] | ["degree"]                |
+------------+---------------------------+
| ["player"] | ["end_year", "start_year"] |
+------------+---------------------------+

nebula> MATCH p = (a:player)-[]->(b)-[]->(c:team) WHERE a.name == "Tim Duncan" AND c.name == "Spurs" RETURN nodes(p)
+-------------------------------------------------------------------------------------------------------------------------------------------------------------+
| nodes(p)                                                                                                                                                    |
+-------------------------------------------------------------------------------------------------------------------------------------------------------------+
| [("player100" :player{age: 42, name: "Tim Duncan"}), ("player101" :player{age: 36, name: "Tony Parker"}), ("team204" :team{name: "Spurs"})]                 |
+-------------------------------------------------------------------------------------------------------------------------------------------------------------+
| [("player100" :player{age: 42, name: "Tim Duncan"}), ("player125" :player{age: 41, name: "Manu Ginobili"}), ("team204" :team{name: "Spurs"})]               |
+-------------------------------------------------------------------------------------------------------------------------------------------------------------+

nebula> MATCH p = (a:player)-[]->(b)-[]->(c:team) WHERE a.name == "Tim Duncan" AND c.name == "Spurs" RETURN relationships(p)
+-----------------------------------------------------------------------------------------------------------------------+
| relationships(p)                                                                                                      |
+-----------------------------------------------------------------------------------------------------------------------+
| [[:follow "player100"->"player101" @0 {degree: 95}], [:serve "player101"->"team204" @0 {end_year: 2018, start_year: 1999}]] |
+-----------------------------------------------------------------------------------------------------------------------+
| [[:follow "player100"->"player125" @0 {degree: 95}], [:serve "player125"->"team204" @0 {end_year: 2018, start_year: 2002}]] |
+-----------------------------------------------------------------------------------------------------------------------+
```

Last update: April 22, 2021

4.5.7 The count() function

The `count()` function calculates the number of the specified values or rows.

- (nGQL-extension) You can use `count()` and `GROUP BY` together to group and count the number of specific values. Use `YIELD` to return.
- (OpenCypher style) You can use `count()` and `RETURN`. `GROUP BY` is not necessary.

Syntax

```
count({expr | *})
```

- count(*) returns the number of rows (including NULL).
- count(expr) return non-NULL values return by an expression.
- count() and size() are different.

EXAMPLES

```
nebula> WITH [NULL, 1, 1, 2, 2] As a UNWIND a AS b RETURN count(b), count(*), count(DISTINCT b)
+----------+----------+-------------------+
| COUNT(b) | COUNT(*) | COUNT(distinct b) |
+----------+----------+-------------------+
| 4        | 5        | 2                 |
+----------+----------+-------------------+
```

```
nebula> GO FROM "player101" OVER follow BIDIRECT YIELD $$.player.name AS Name | \
        GROUP BY $-.Name YIELD $-.Name, count(*);
+---------------------+----------+
| $-.Name             | COUNT(*) |
+---------------------+----------+
| "Dejounte Murray"   | 1        |
+---------------------+----------+
| "LaMarcus Aldridge" | 2        |
+---------------------+----------+
| "Tim Duncan"        | 2        |
+---------------------+----------+
| "Marco Belinelli"   | 1        |
+---------------------+----------+
| "Manu Ginobili"     | 1        |
+---------------------+----------+
| "Boris Diaw"        | 1        |
+---------------------+----------+
```

The statement in the preceding example searches for:

- People whom `player101` follows.
- People who follow `player101`.

And retrieves two columns:

- `$-.Name`, the names of the people.
- `COUNT(*)`, how many times the names show up.

Because there are no duplicate names in the `basketballplayer` dataset, the number `2` in the result shows that the person in that row and `player101` have followed each other.

```
nebula> LOOKUP ON player YIELD player.age As playerage \|
 GROUP BY $-.playerage YIELD $-.playerage as age, count(*) AS number | ORDER BY number DESC, age DESC
+-----+--------+
| age | number |
+-----+--------+
| 34  | 4      |
+-----+--------+
| 33  | 4      |
+-----+--------+
| 30  | 4      |
+-----+--------+
| 29  | 4      |
+-----+--------+
| 38  | 3      |
+-----+--------+
...
```

```
nebula> MATCH (n:player) RETURN n.age as age, count(*) as number ORDER BY number DESC, age DESC
+-----+--------+
| age | number |
+-----+--------+
| 34  | 4      |
+-----+--------+
| 33  | 4      |
+-----+--------+
| 30  | 4      |
+-----+--------+
| 29  | 4      |
+-----+--------+
| 38  | 3      |
+-----+--------+
```

The two statements in the preceding examples retrieves the age distribution of the players in the dataset.

```
nebula> MATCH (v:player{name:"Tim Duncan"}) -- (v2) RETURN count(DISTINCT v2)
+--------------------+
| COUNT(distinct v2) |
+--------------------+
| 11                 |
+--------------------+
nebula> MATCH (n:player {name : "Tim Duncan"})-[]->(friend:player)-[]->(fof:player) RETURN count(fof), count(DISTINCT fof)
+------------+----------------------+
| COUNT(fof) | COUNT(distinct fof) |
+------------+----------------------+
| 4          | 3                    |
+------------+----------------------+
```

count(NULL)

```
nebula>  RETURN count(NULL), size(NULL)
+-------------+------------+
| COUNT(NULL) | size(NULL) |
+-------------+------------+
| 0           | __NULL__   |
+-------------+------------+
```

Last update: April 13, 2021

4.5.8 collect()

`collect()` returns a list containing the values returned by an expression. Using this function aggregates data by amalgamating multiple records or values into a single list.

`collect()` is an aggregation function. Like `GROUP BY` in SQL.

Examples

This example works like `GROUP BY`.

```
nebula> UNWIND [1, 2, 1] AS a RETURN a;
+---+
| a |
+---+
| 1 |
+---+
| 2 |
+---+
| 1 |
+---+
nebula> UNWIND [1, 2, 1] AS a RETURN collect(a);
+------------+
| COLLECT(a) |
+------------+
| [1, 2, 1]  |
+------------+
nebula> UNWIND [1, 2, 1] AS a RETURN a, collect(a), size(collect(a))
+---+------------+-----------------+
| a | COLLECT(a) | size(COLLECT(a)) |
+---+------------+-----------------+
| 2 | [2]        | 1               |
+---+------------+-----------------+
| 1 | [1, 1]     | 2               |
+---+------------+-----------------+
```

You can sort reversely, limit output rows to 3, and collect the output into a list.

```
nebula> UNWIND ["c", "b", "a", "d" ] AS p \
    WITH p AS q \
    ORDER BY q DESC LIMIT 3 \
    RETURN collect(q);
+-----------------+
| COLLECT(q)      |
+-----------------+
| ["d", "c", "b"] |
+-----------------+
nebula> WITH [1, 1, 2, 2] AS coll \
    UNWIND coll AS x \
    WITH DISTINCT x \
    RETURN collect(x) AS ss
+--------+
| ss     |
+--------+
| [1, 2] |
+--------+
```

This example aggregates all players' names by their ages.

```
nebula> MATCH (n:player) RETURN collect(n.age);
+-------------------------------------------------------------+
| COLLECT(n.age)                                              |
+-------------------------------------------------------------+
| [32, 32, 34, 29, 41, 40, 33, 25, 40, 37, ...                |
+-------------------------------------------------------------+

nebula> MATCH (n:player) RETURN n.age AS age, collect(n.name);
+-----+-------------------------------------------------------------------------+
| 27  | ["Cory Joseph"]                                                         |
+-----+-------------------------------------------------------------------------+
| 28  | ["Damian Lillard", "Paul George", "Ricky Rubio"]                        |
+-----+-------------------------------------------------------------------------+
| 29  | ["Dejounte Murray", "James Harden", "Klay Thompson", "Jonathon Simmons"] |
+-----+-------------------------------------------------------------------------+
...
```

Last update: April 22, 2021

4.5.9 reduce() function

OpenCypher Compatibility

In openCypher, the function `reduce()` is not defined. nGQL implements `reduce()` function as the Cypher way.

Syntax

`reduce()` returns the value resulting from the application of an expression on each successive element in a list in conjunction with the result of the computation thus far. This function will iterate through each element `e` in the given list, run the expression on `e` — taking into account the current partial result — and store the new partial result in the accumulator. This function is analogous to the fold or reduce method in functional languages such as Lisp and Scala.

```
reduce(accumulator = initial, variable IN list | expression)
```

- Arguments:

Name	Description
accumulator	A variable that will hold the result and the partial results as the list is iterated.
initial	An expression that runs once to give a starting value to the accumulator.
list	An expression that returns a list.
variable	The closure will have a variable introduced in its context. We decide here which variable to use.
expression	This expression will run once per value in the list, and produce the result value.

- Returns:

  The type of the value returned depends on the arguments provided, along with the semantics of expression.

Example

```
nebula> RETURN reduce(totalNum = 10, n IN range(1, 3) | totalNum + n) AS r;
+----+
| r  |
+----+
| 16 |
+----+

nebula> RETURN reduce(totalNum = -4 * 5, n IN [1, 2] | totalNum + n * 2) AS r;
+-----+
| r   |
+-----+
| -14 |
+-----+
```

```
nebula> MATCH p = (n:player{name:"LeBron James"})<-[:follow]-(m) \
        RETURN      nodes(p)[0].age AS src1,  \
                    nodes(p)[1].age AS dst2,  \
                    reduce(totalAge = 100, n IN nodes(p) | totalAge + n.age) AS sum
+------+------+-----+
| src1 | dst2 | sum |
+------+------+-----+
| 34   | 31   | 165 |
+------+------+-----+
| 34   | 29   | 163 |
+------+------+-----+
| 34   | 33   | 167 |
+------+------+-----+
| 34   | 26   | 160 |
+------+------+-----+
| 34   | 34   | 168 |
+------+------+-----+
| 34   | 37   | 171 |
+------+------+-----+

nebula> LOOKUP ON player WHERE player.name == "Tony Parker" |  GO FROM $-.VertexID over follow WHERE follow.degree != reduce(totalNum = 5, n IN range(1, 3) | $
$.player.age + totalNum + n) YIELD $$.player.name AS id, $$.player.age AS age, follow.degree AS degree
+---------------------+-----+--------+
| id                  | age | degree |
+---------------------+-----+--------+
```

```
| "Tim Duncan"        | 42  | 95     |
+---------------------+-----+--------+
| "LaMarcus Aldridge" | 33  | 90     |
+---------------------+-----+--------+
| "Manu Ginobili"     | 41  | 95     |
+---------------------+-----+--------+
```

Last update: April 22, 2021

4.5.10 Hash

The `hash()` function returns the hash value of the argument. The argument can be a number, a string, a list, a boolean, null, or an expression that evaluates to a value of the preceding data types.

The source code of the `hash()` function (MurmurHash2), seed (`0xc70f6907UL`), and other parameters can be found in `MurmurHahs2.h` .

> **Note**
>
> Roughly, The chance of collision is about 1/10 in the case of 1 billion vertices. The number of edges is irrelevant to the collision possibility.

For Java, call like follows.

```
MurmurHash2.hash64("to_be_hashed".getBytes(),"to_be_hashed".getBytes().length, 0xc70f6907)
```

Legacy version compatibility

In nGQL 1.0, when nGQL does not support string VIDs, a common practice is to hash the strings first and then use the values as VIDs. But in nGQL 2.0, both string VIDs and integer VIDs are supported, you don't have to use `hash()` to make VIDs.

Hash a number

```
nebula> YIELD hash(-123);
+--------------+
| hash(-(123)) |
+--------------+
| -123         |
+--------------+
```

Hash a string

```
nebula> YIELD hash("to_be_hashed");
+----------------------+
| hash(to_be_hashed)   |
+----------------------+
| -1098333533029391540 |
+----------------------+
```

Hash a list

```
nebula> YIELD hash([1,2,3]);
+----------------+
| hash([1,2,3])  |
+----------------+
| 11093822460243 |
+----------------+
```

Hash a boolean

```
nebula> YIELD hash(true);
+------------+
| hash(true) |
+------------+
| 1          |
+------------+

nebula> YIELD hash(false);
+-------------+
| hash(false) |
+-------------+
| 0           |
+-------------+
```

Hash NULL

```
nebula> YIELD hash(NULL);
+------------+
| hash(NULL) |
+------------+
| -1         |
+------------+
```

Hash an expression

```
nebula> YIELD hash(toLower("HELLO NEBULA"));
+-------------------------------+
| hash(toLower("HELLO NEBULA")) |
+-------------------------------+
| -8481157362655072082          |
+-------------------------------+
```

Last update: April 22, 2021

4.5.11 Predicate functions

Predicate functions return true or false. They are most commonly used in `WHERE`.

Functions	Description
exists()	returns true if the specified property exists in the vertex, edge or map.
any()	returns true if the predicate holds for at least one element in the given list.
all()	returns true if the predicate holds for all elements in the given list.
none()	returns true if the predicate holds for no element in the given list.
single()	returns true if the predicate holds for exactly one of the elements in the given list.

Note

NULL is returned if the list is NULL or all of its elements are NULL.

OpenCypher compatibility

In openCypher, only function `exists()` is defined and specified. The other functions are implement-dependent.

Syntax

```
<predicate>(<variable> IN <list> WHERE <condition>)
```

Examples

```
nebula> RETURN any(n IN [1, 2, 3, 4, 5, NULL] WHERE n > 2) AS r
+------+
| r    |
+------+
| true |
+------+

nebula> RETURN single(n IN range(1, 5) WHERE n == 3) AS r
+------+
| r    |
+------+
| true |
+------+

nebula> RETURN none(n IN range(1, 3) WHERE n == 0) AS r
+------+
| r    |
+------+
| true |
+------+

nebula>  WITH [1, 2, 3, 4, 5, NULL] AS a RETURN any(n IN a WHERE n > 2)
+-------------------------+
| any(n IN a WHERE (n>2)) |
+-------------------------+
| true                    |
+-------------------------+

nebula>  MATCH p = (n:player{name:"LeBron James"})<-[:follow]-(m) \
         RETURN nodes(p)[0].name AS n1, nodes(p)[1].name AS n2, \
         all(n IN nodes(p) WHERE n.name NOT STARTS WITH "D") AS b
+---------------+-------------------+-------+
| n1            | n2                | b     |
+---------------+-------------------+-------+
| "LeBron James" | "Danny Green"     | false |
+---------------+-------------------+-------+
| "LeBron James" | "Dejounte Murray" | false |
+---------------+-------------------+-------+
| "LeBron James" | "Chris Paul"      | true  |
+---------------+-------------------+-------+
| "LeBron James" | "Kyrie Irving"    | true  |
+---------------+-------------------+-------+
| "LeBron James" | "Carmelo Anthony" | true  |
+---------------+-------------------+-------+
```

```
| "LeBron James" | "Dwyane Wade"     | false |
+----------------+-------------------+-------+

nebula> MATCH p = (n:player{name:"LeBron James"})-[:follow]->(m) \
        RETURN single(n IN nodes(p) WHERE n.age > 40) AS b
+------+
| b    |
+------+
| true |
+------+

nebula> MATCH (n:player) RETURN exists(n.id), n IS NOT NULL
+--------------+---------------+
| exists(n.id) | n IS NOT NULL |
+--------------+---------------+
| false        | true          |
+--------------+---------------+
...
```

Last update: April 22, 2021

4.5.12 User-defined functions

OpenCypher compatibility

User-defined functions are not yet supported nor designed in Nebula Graph 2.x.

Last update: March 17, 2021

4.6 General queries statements

4.6.1 MATCH

The `MATCH` statement provides the searching ability based on pattern matching.

A `MATCH` statement defines a search pattern and uses it to match data stored in Nebula Graph and to retrieve them in the form defined in the `RETURN` clause. A `WHERE` clause is often used together with the pattern as a filter to the search result.

The examples in this topic use the `basketballplayer` dataset as the sample dataset.

Syntax

The syntax of `MATCH` is relatively more flexible compared with that of other query statements such as `GO` or `LOOKUP`. But generally, it can be summarized as follows.

```
MATCH <pattern> [<WHERE clause>] RETURN <output>
```

The workflow of MATCH

1. The `MATCH` statement uses a **native index** to locate a source vertex. The vertex can be in any position in a pattern. In other words, in a valid `MATCH` statement, **there must be an indexed property or tag, or a specific VID.** For how to index a property, see Create native index.

   > **Note**
   >
   > The native index for VID is created by default, so you don't need to create an extra index if you want to match on VID.

2. The `MATCH` statement searches through the pattern to match edges and other vertices.
3. The `MATCH` statement retrieves data according to the `RETURN` clause.

> **OpenCypher compatibility**
>
> For now, nGQL DOES NOT support scanning all vertices and edges with `MATCH`. For example, `MATCH (v) RETURN v`.

Use patterns in MATCH statements

Make sure there is at least one index for the `MATCH` statement to use. If you want to create an index, but there are already vertices or edges related to the tag, edge type, or property that you want to create the index for, you have to rebuild the index after creation to make it take effect on existing vertices or edges.

> **Caution**
>
> Correct use of indexes can speed up queries, but indexes can dramatically reduce the write performance. The performance reduction can be as much as 90% or even more. **DO NOT** use indexes in production environments unless you are fully aware of their influences on your service.

```
nebula> CREATE TAG INDEX name ON player(name(20)); // Create an index on the name property.
Execution succeeded (time spent 2957/3986 us)

nebula> REBUILD TAG INDEX name; // Rebuild the index.
+------------+
| New Job Id |
+------------+
| 121        |
+------------+
```

```
Got 1 rows  (time spent 2676/3990 us)

nebula> SHOW JOB 121; // Make sure the rebuild job succeeded.
+---------------+---------------------+------------+------------+------------+
| Job Id(TaskId) | Command(Dest)       | Status     | Start Time | Stop Time  |
+---------------+---------------------+------------+------------+------------+
| 121           | "REBUILD_TAG_INDEX" | "FINISHED" | 1607073046 | 1607073046 |
+---------------+---------------------+------------+------------+------------+
| 0             | "storaged2"         | "FINISHED" | 1607073046 | 1607073046 |
+---------------+---------------------+------------+------------+------------+
| 1             | "storaged0"         | "FINISHED" | 1607073046 | 1607073046 |
+---------------+---------------------+------------+------------+------------+
| 2             | "storaged1"         | "FINISHED" | 1607073046 | 1607073046 |
+---------------+---------------------+------------+------------+------------+
Got 4 rows (time spent 1186/2998 us)
```

MATCH A VERTEX

You can use a user-defined variable in a pair of parentheses to represent a vertex in a pattern. For example: `(v)`.

MATCH ON TAG

To match on a tag, make sure there is an applicable **tag index**. For how to create a tag index, see Create tag indexes.

> **Note**
>
> Tag indexes are different from property indexes. If there is an index for a property of a tag, but no index for the tag, you cannot match on the tag.

A vertex tag is specified with `:<tag_name>` in a pattern.

```
nebula> MATCH (v:player) RETURN v
+------------------------------------------------------------+
| v                                                          |
+------------------------------------------------------------+
| ("player102" :player{age: 33, name: "LaMarcus Aldridge"})  |
+------------------------------------------------------------+
| ("player106" :player{age: 25, name: "Kyle Anderson"})      |
+------------------------------------------------------------+
| ("player115" :player{age: 40, name: "Kobe Bryant"})        |
+------------------------------------------------------------+
...
```

MATCH ON VERTEX PROPERTY

Tag properties are specified with `{<prop_name>: <prop_value>}` in a pattern after a tag.

The following example uses the `name` property to match a vertex.

```
nebula> MATCH (v:player{name:"Tim Duncan"}) RETURN v;
+-------------------------------------------------+
| v                                               |
+-------------------------------------------------+
| ("player100" :player{name: "Tim Duncan", age: 42}) |
+-------------------------------------------------+
```

The `WHERE` clause can do the same thing:

```
nebula> MATCH (v:player) WHERE v.name == "Tim Duncan" RETURN v;
+-------------------------------------------------+
| v                                               |
+-------------------------------------------------+
| ("player100" :player{name: "Tim Duncan", age: 42}) |
+-------------------------------------------------+
```

> **OpenCypher compatibility**
>
> - In nGQL, `==` is the equality operator and `=` is the assignment operator (as in C++ or Java).
> - In openCypher 9, `=` is the equality operator.

MATCH ON VID

You can use the VID to match a vertex. The `id()` function can retrieve the VID of a vertex.

```
nebula> MATCH (v) WHERE id(v) == 'player101' RETURN v;
+-------------------------------------------------+
| v                                               |
+-------------------------------------------------+
| (player101) player.name:Tony Parker,player.age:36 |
+-------------------------------------------------+
Got 1 rows (time spent 1710/2406 us)
```

To match on multiple VIDs, use `WHERE id(v) IN [vid_list]`.

```
nebula> MATCH (v:player { name: 'Tim Duncan' })--(v2) \
        WHERE id(v2) IN ["player101", "player102"] RETURN v2;
+-------------------------------------------------------+
| v2                                                    |
+-------------------------------------------------------+
| ("player101" :player{name: "Tony Parker", age: 36})   |
+-------------------------------------------------------+
| ("player102" :player{name: "LaMarcus Aldridge", age: 33}) |
+-------------------------------------------------------+
| ("player101" :player{name: "Tony Parker", age: 36})   |
+-------------------------------------------------------+
Got 3 rows (time spent 3107/3683 us)
```

MATCH CONNECTED VERTICES

You can use the `--` symbol to represent edges of both directions and match vertices connected by these edges.

> **Legacy**
>
> - In nGQL 1.x, the `--` symbol is used for inline comments.
> - Starting from nGQL 2.0, the `--` symbol represents an incoming or outgoing edge.

```
nebula> MATCH (v:player{name:"Tim Duncan"})--(v2) RETURN v2.name AS Name;
+--------------------+
| Name               |
+--------------------+
| "Tony Parker"      |
+--------------------+
| "LaMarcus Aldridge" |
+--------------------+
| "Marco Belinelli"  |
+--------------------+
| "Danny Green"      |
+--------------------+
| "Aron Baynes"      |
+--------------------+
...
Got 13 rows (time spent 6029/8976 us)
```

And you can add a `>` or `<` to the `--` symbol to specify the direction of an edge.

```
nebula> MATCH (v:player{name:"Tim Duncan"})-->(v2) RETURN v2.name AS Name;
+-----------------+
| Name            |
+-----------------+
| "Spurs"         |
+-----------------+
| "Tony Parker"   |
+-----------------+
| "Manu Ginobili" |
+-----------------+
Got 3 rows (time spent 2897/5993 us)
```

In the preceding example, `-->` represents an edge that starts from `v` and points to `v2`. To `v`, this is an outgoing edge, and to `v2` this is an incoming edge.

To extend the pattern, add more edges and vertices.

```
nebula> MATCH (v:player{name:"Tim Duncan"})-->(v2)<--(v3) RETURN v3.name AS Name;
+--------------------+
| Name               |
+--------------------+
| "Tony Parker"      |
+--------------------+
```

```
| "Tiago Splitter"    |
+---------------------+
| "Dejounte Murray"   |
+---------------------+
| "Tony Parker"       |
+---------------------+
| "LaMarcus Aldridge" |
+---------------------+
...
```

If you don't need to refer to a vertex, you can omit the variable representing it in the parentheses.

```
nebula> MATCH (v:player{name:"Tim Duncan"})-->()<--(v3) RETURN v3.name AS Name;
+---------------------+
| Name                |
+---------------------+
| "Tony Parker"       |
+---------------------+
| "LaMarcus Aldridge" |
+---------------------+
| "Rudy Gay"          |
+---------------------+
| "Danny Green"       |
+---------------------+
| "Kyle Anderson"     |
+---------------------+
...
```

MATCH PATHS

Connected vertices and edges form a path. You can use a user-defined variable as follows to name a path.

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})-->(v2) RETURN p;
+-----------------------------------------+
| p                                       |
+-----------------------------------------+
| <("player100" :player{age: 42, name: "Tim Duncan"})-[:follow@0 {degree: 95}]->("player101" :player{age: 36, name: "Tony Parker"})>   |
+-----------------------------------------+
| <("player100" :player{age: 42, name: "Tim Duncan"})-[:follow@0 {degree: 95}]->("player125" :player{age: 41, name: "Manu Ginobili"})> |
+-----------------------------------------+
| <("player100" :player{age: 42, name: "Tim Duncan"})-[:serve@0 {end_year: 2016, start_year: 1997}]->("team204" :team{name: "Spurs"})> |
+-----------------------------------------+
Got 3 rows (time spent 3717/4573 us)
```

OpenCypher compatibility

In nGQL, the `@` symbol represents the rank of an edge, but openCypher has no such a concept.

MATCH EDGES

Besides using `--`, `-->`, or `<--` to indicate a nameless edge, you can use a variable in a pair of square brackets to represent a named edge. For example: `-[e]-`.

```
nebula> MATCH (v:player{name:"Tim Duncan"})-[e]-(v2) RETURN e;
+------------------------------------------------------------------------+
| e                                                                      |
+------------------------------------------------------------------------+
| [:follow "player101"->"player100" @0 {degree: 95}]                     |
+------------------------------------------------------------------------+
| [:follow "player102"->"player100" @0 {degree: 75}]                     |
+------------------------------------------------------------------------+
| [:serve "player100"->"team204" @0 {end_year: 2016, start_year: 1997}]  |
+------------------------------------------------------------------------+
...
```

MATCH ON EDGE TYPES AND PROPERTIES

Just like tags, edge types are specified with `:<edge_type>`. For example: `-[e:serve]-`.

```
nebula> MATCH (v:player{name:"Tim Duncan"})-[e:serve]-(v2) RETURN e;
+------------------------------------------------------------------------+
| e                                                                      |
+------------------------------------------------------------------------+
| [:serve "player100"->"team204" @0 {end_year: 2016, start_year: 1997}]  |
+------------------------------------------------------------------------+
Got 1 rows (time spent 5041/5630 us)
```

And edge type properties are specified with `{<prop_name>: <prop_value>}` after the `:<edge_type>`. For example: `[e:follow{likeness:95}]`.

```
nebula> MATCH (v:player{name:"Tim Duncan"})-[e:follow{degree:95}]->(v2) RETURN e;
+-------------------------------------------------------+
| e                                                     |
+-------------------------------------------------------+
| [:follow "player100"->"player101" @0 {degree: 95}]    |
+-------------------------------------------------------+
| [:follow "player100"->"player125" @0 {degree: 95}]    |
+-------------------------------------------------------+
Got 2 rows (time spent 6080/6728 us)
```

MATCH ON MULTIPLE EDGE TYPES

The `|` symbol can help matching on multiple edge types. For example: `[e:follow|:serve]`.

```
nebula> MATCH (v:player{name:"Tim Duncan"})-[e:follow|:serve]->(v2) RETURN e;
+-------------------------------------------------------------------------+
| e                                                                       |
+-------------------------------------------------------------------------+
| [:follow "player100"->"player101" @0 {degree: 95}]                      |
+-------------------------------------------------------------------------+
| [:follow "player100"->"player125" @0 {degree: 95}]                      |
+-------------------------------------------------------------------------+
| [:serve "player100"->"team204" @0 {end_year: 2016, start_year: 1997}]   |
+-------------------------------------------------------------------------+
Got 3 rows (time spent 4264/4976 us)
```

MATCH MULTIPLE EDGES

You can expand a pattern to match multiple edges in a path.

```
nebula> MATCH (v:player{name:"Tim Duncan"})-[]->(v2)<-[e:serve]-(v3) RETURN v2, v3;
+----------------------------------+-----------------------------------------------------------+
| v2                               | v3                                                        |
+----------------------------------+-----------------------------------------------------------+
| ("player204" :team{name: "Spurs"}) | ("player101" :player{name: "Tony Parker", age: 36})       |
+----------------------------------+-----------------------------------------------------------+
| ("player204" :team{name: "Spurs"}) | ("player102" :player{name: "LaMarcus Aldridge", age: 33}) |
+----------------------------------+-----------------------------------------------------------+
| ("player204" :team{name: "Spurs"}) | ("player103" :player{age: 32, name: "Rudy Gay"})          |
+----------------------------------+-----------------------------------------------------------+
...
```

MATCH FIXED-LENGTH PATHS

To match a fixed-length path, use the `:<edge_type>*<hop>` pattern. `hop` must be a non-negative integer.

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})-[e:follow*2]->(v2) RETURN DISTINCT v2 AS Friends;
+-------------------------------------------------------+
| Friends                                               |
+-------------------------------------------------------+
| ("player100" :player{name: "Tim Duncan", age: 42})        |
+-------------------------------------------------------+
| ("player102" :player{name: "LaMarcus Aldridge", age: 33}) |
+-------------------------------------------------------+
| ("player125" :player{name: "Manu Ginobili", age: 41})     |
+-------------------------------------------------------+
Got 3 rows (time spent 4863/5591 us)
```

If `hop` is 0, the pattern matches the source vertex on the path.

```
nebula> MATCH (v:player{name:"Tim Duncan"}) -[*0]-> (v2) RETURN v2;
+-------------------------------------------------+
| v2                                              |
+-------------------------------------------------+
| ("player100" :player{age: 42, name: "Tim Duncan"}) |
+-------------------------------------------------+
Got 1 rows (time spent 2785/3377 us)
```

MATCH VARIABLE-LENGTH PATHS

You can use the `:<edge_type>*[minHop]..<maxHop>` pattern to match variable-length paths.

Parameter	Description
`minHop`	Optional. Represents the minimum length of the path. `minHop` must be a non-negative integer. The default value is 1.
`maxHop`	Required. Represents the maximum length of the path. `maxHop` must be a non-negative integer. It has no default value.

OpenCypher compatibility

- In nGQL, `maxHop` is required. And `..` cannot be omitted after `minHop`.
- In openCypher, `maxHop` is optional and default to infinity. When no bounds are given, `..` can be omitted.

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})-[e:follow*1..3]->(v2) \
        RETURN v2 AS Friends;
+-------------------------------------------------------+
| Friends                                               |
+-------------------------------------------------------+
| ("player100" :player{age: 42, name: "Tim Duncan"})    |
+-------------------------------------------------------+
| ("player101" :player{age: 36, name: "Tony Parker"})   |
+-------------------------------------------------------+
| ("player125" :player{age: 41, name: "Manu Ginobili"}) |
+-------------------------------------------------------+
| ("player102" :player{age: 33, name: "LaMarcus Aldridge"}) |
+-------------------------------------------------------+
Got 4 rows (time spent 6166/6887 us)
```

You can use the `DISTINCT` keyword to aggregate duplicate results.

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})-[e:follow*1..3]->(v2:player) \
        RETURN DISTINCT v2 AS Friends, count(v2);
+-------------------------------------------------------+-----------+
| Friends                                               | COUNT(v2) |
+-------------------------------------------------------+-----------+
| ("player125" :player{age: 41, name: "Manu Ginobili"}) | 3         |
+-------------------------------------------------------+-----------+
| ("player102" :player{age: 33, name: "LaMarcus Aldridge"}) | 1         |
+-------------------------------------------------------+-----------+
| ("player100" :player{age: 42, name: "Tim Duncan"})    | 4         |
+-------------------------------------------------------+-----------+
| ("player101" :player{age: 36, name: "Tony Parker"})   | 3         |
+-------------------------------------------------------+-----------+
Got 4 rows (time spent 5502/6556 us)
```

If `minHop` is 0, the pattern matches the source vertex. Compared to the preceding statement, the following statement uses 0 as the `minHop`, so in the following result set `"Tim Duncan"` is counted one more time than it is in the preceding result set because it is the source vertex.

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})-[e:follow*0..3]->(v2:player) \
        RETURN DISTINCT v2 AS Friends, count(v2);
+-------------------------------------------------------+-----------+
| Friends                                               | COUNT(v2) |
+-------------------------------------------------------+-----------+
| ("player125" :player{age: 41, name: "Manu Ginobili"}) | 3         |
+-------------------------------------------------------+-----------+
| ("player101" :player{age: 36, name: "Tony Parker"})   | 3         |
+-------------------------------------------------------+-----------+
| ("player102" :player{age: 33, name: "LaMarcus Aldridge"}) | 1         |
+-------------------------------------------------------+-----------+
| ("player100" :player{age: 42, name: "Tim Duncan"})    | 5         |
+-------------------------------------------------------+-----------+
Got 4 rows (time spent 5553/6275 us)
```

MATCH VARIABLE-LENGTH PATHS WITH MULTIPLE EDGE TYPES

You can specify multiple edge types in a fixed-length or variable-length pattern. In this case, `hop`, `minHop`, and `maxHop` take effect on all edge types.

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})-[e:follow|serve*2]->(v2) \
        RETURN DISTINCT v2;
```

```
+-------------------------------------------------------------+
| v2                                                          |
+-------------------------------------------------------------+
| ("player100" :player{name: "Tim Duncan", age: 42})        |
+-------------------------------------------------------------+
| ("player102" :player{name: "LaMarcus Aldridge", age: 33}) |
+-------------------------------------------------------------+
| ("player125" :player{name: "Manu Ginobili", age: 41})     |
+-------------------------------------------------------------+
| ("player204" :team{name: "Spurs"})                        |
+-------------------------------------------------------------+
| ("player215" :team{name: "Hornets"})                      |
+-------------------------------------------------------------+
Got 5 rows (time spent 3834/4571 us)
```

Common retrieving operations

This section shows how to retrieve commonly used items with `MATCH` statements.

RETRIEVE VERTEX OR EDGE INFORMATION

Use `RETURN {<vertex_name> | <edge_name>}` to retrieve all the information of a vertex or an edge.

```
nebula> MATCH (v:player{name:"Tim Duncan"}) RETURN v;
+-------------------------------------------------+
| v                                               |
+-------------------------------------------------+
| ("player100" :player{name: "Tim Duncan", age: 42}) |
+-------------------------------------------------+
Got 1 rows (time spent 1863/2545 us)
```

```
nebula> MATCH (v:player{name:"Tim Duncan"})-[e]->(v2) RETURN e;
+-----------------------------------------------------------------------------+
| e                                                                           |
+-----------------------------------------------------------------------------+
| [:follow "player100"->"player101" @0 {degree: 95}]                        |
+-----------------------------------------------------------------------------+
| [:follow "player100"->"player125" @0 {degree: 95}]                        |
+-----------------------------------------------------------------------------+
| [:serve "player100"->"team204" @0 {end_year: 2016, start_year: 1997}]     |
+-----------------------------------------------------------------------------+
Got 3 rows (time spent 3139/3773 us)
```

RETRIEVE VIDS

Use the `id()` function to retrieve VIDs.

```
nebula> MATCH (v:player{name:"Tim Duncan"}) RETURN id(v);
+-------------+
| id(v)       |
+-------------+
| "player100" |
+-------------+
Got 1 rows (time spent 2070/2747 us)
```

RETRIEVE TAGS

Use the `labels()` function to retrieve the list of tags on a vertex.

```
nebula> MATCH (v:player{name:"Tim Duncan"}) RETURN labels(v);
+------------+
| labels(v)  |
+------------+
| ["player"] |
+------------+
Got 1 rows (time spent 2198/2941 us)
```

To retrieve the nth element in the `labels(v)` list, use `labels(v)[n-1]`. The following example shows how to use `labels(v)[0]` to retrieve the first tag in the list.

```
nebula> MATCH (v:player{name:"Tim Duncan"}) RETURN labels(v)[0];
+--------------+
| labels(v)[0] |
+--------------+
| "player"     |
+--------------+
Got 1 rows (time spent 2609/3481 us)
```

RETRIEVE A SINGLE PROPERTY ON A VERTEX OR AN EDGE

Use `RETURN {<vertex_name> | <edge_name>}.<property>` to retrieve a single property.

```
nebula> MATCH (v:player{name:"Tim Duncan"}) RETURN v.age;
+-------+
| v.age |
+-------+
| 42    |
+-------+
Got 1 rows (time spent 2261/2973 us)
```

Use `AS` to specify an alias for a property.

```
nebula> MATCH (v:player{name:"Tim Duncan"}) RETURN v.age AS Age;
+-----+
| Age |
+-----+
| 42  |
+-----+
Got 1 rows (time spent 1762/2321 us)
```

RETRIEVE ALL PROPERTIES ON A VERTEX OR AN EDGE

Use the `properties()` function to retrieve all properties on a vertex or an edge.

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})-[]->(v2) RETURN properties(v2);
+------------------------------------+
| properties(v2)                     |
+------------------------------------+
| {"name":"Spurs"}                   |
+------------------------------------+
| {"name":"Tony Parker", "age":36}   |
+------------------------------------+
| {"age":41, "name":"Manu Ginobili"} |
+------------------------------------+
Got 3 rows (time spent 2943/3541 us)
```

RETRIEVE EDGE TYPES

Use the `type()` function to retrieve the types of the matched edges.

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})-[e]->() RETURN DISTINCT type(e);
+----------+
| type(e)  |
+----------+
| "follow" |
+----------+
| "serve"  |
+----------+
Got 3 rows (time spent 3776/4660 us)
```

RETRIEVE PATHS

Use `RETURN <path_name>` to retrieve all the information of the matched paths.

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})-[*3]->() RETURN p;
+-------------------------------------------------------------------------------------------------+
| p                                                                                               |
+-------------------------------------------------------------------------------------------------+
| <("player100" :player{age: 42, name: "Tim Duncan"})-[:follow@0 {degree: 95}]->("player101" :player{age: 36, name: "Tony Parker"})-[:follow@0 {degree: 90}]-
>("player102" :player{age: 33, name: "LaMarcus Aldridge"})-[:serve@0 {end_year: 2019, start_year: 2015}]->("team204" :team{name: "Spurs"})>           |
+-------------------------------------------------------------------------------------------------+
| <("player100" :player{age: 42, name: "Tim Duncan"})-[:follow@0 {degree: 95}]->("player101" :player{age: 36, name: "Tony Parker"})-[:follow@0 {degree: 90}]-
>("player102" :player{age: 33, name: "LaMarcus Aldridge"})-[:serve@0 {end_year: 2015, start_year: 2006}]->("team203" :team{name: "Trail Blazers"})>   |
+-------------------------------------------------------------------------------------------------+
| <("player100" :player{age: 42, name: "Tim Duncan"})-[:follow@0 {degree: 95}]->("player101" :player{age: 36, name: "Tony Parker"})-[:follow@0 {degree: 90}]-
>("player102" :player{age: 33, name: "LaMarcus Aldridge"})-[:follow@0 {degree: 75}]->("player101" :player{age: 36, name: "Tony Parker"})>             |
+-------------------------------------------------------------------------------------------------+
...
```

RETRIEVE VERTICES IN A PATH

Use the `nodes()` function to retrieve all vertices in a path.

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})-[]->(v2) RETURN nodes(p);
+-----------------------------------------------------------------------------------------------------------------------+
| nodes(p)                                                                                                              |
+-----------------------------------------------------------------------------------------------------------------------+
| [("player100" :star{} :player{age: 42, name: "Tim Duncan"}), ("player204" :team{name: "Spurs"})]                      |
+-----------------------------------------------------------------------------------------------------------------------+
| [("player100" :star{} :player{age: 42, name: "Tim Duncan"}), ("player101" :player{name: "Tony Parker", age: 36})]   |
```

```
+-----------------------------------------------------------------------------------------------------------------+
| [("player100" :star{} :player{age: 42, name: "Tim Duncan"}), ("player125" :player{name: "Manu Ginobili", age: 41})] |
+-----------------------------------------------------------------------------------------------------------------+
Got 3 rows (time spent 2529/3128 us)
```

RETRIEVE EDGES IN A PATH

Use the `relationships()` function to retrieve all edges in a path.

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})-[]->(v2) RETURN relationships(p);
+-------------------------------------------------------------------------+
| relationships(p)                                                        |
+-------------------------------------------------------------------------+
| [[:follow "player100"->"player101" @0 {degree: 95}]]                    |
+-------------------------------------------------------------------------+
| [[:follow "player100"->"player125" @0 {degree: 95}]]                    |
+-------------------------------------------------------------------------+
| [[:serve "player100"->"team204" @0 {end_year: 2016, start_year: 1997}]] |
+-------------------------------------------------------------------------+
Got 3 rows (time spent 2715/3363 us)
```

RETRIEVE PATH LENGTH

Use the `length()` function to retrieve the length of a path.

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})-[*..2]->(v2) \
        RETURN p AS Paths, length(p) AS Length;
+-----------------------------------------------------------------------+--------+
| Paths                                                                 | Length |
+-----------------------------------------------------------------------+--------+
| <("player100" :player{age: 42, name: "Tim Duncan"})-[:follow@0 {degree: 95}]->("player125" :player{age: 41, name: "Manu Ginobili"})-[:serve@0 {end_year: 2018,
start_year: 2002}]->("team204" :team{name: "Spurs"})>   | 2      |
+-----------------------------------------------------------------------+--------+
| <("player100" :player{age: 42, name: "Tim Duncan"})-[:follow@0 {degree: 95}]->("player125" :player{age: 41, name: "Manu Ginobili"})-[:follow@0 {degree: 90}]-
>("player100" :player{age: 42, name: "Tim Duncan"})>      | 2      |
+-----------------------------------------------------------------------+--------+
|  <("player100" :player{age: 42, name: "Tim Duncan"})-[:follow@0 {degree: 95}]->("player101" :player{age: 36, name: "Tony Parker"})-[:serve@0 {end_year: 2019,
start_year: 2018}]->("team215" :team{name: "Hornets"})>  | 2      |
+-----------------------------------------------------------------------+--------+
| <("player100" :player{age: 42, name: "Tim Duncan"})-[:follow@0 {degree: 95}]->("player101" :player{age: 36, name: "Tony Parker"})-[:serve@0 {end_year: 2018,
start_year: 1999}]->("team204" :team{name: "Spurs"})>     | 2      |
+-----------------------------------------------------------------------+--------+
| <("player100" :player{age: 42, name: "Tim Duncan"})-[:follow@0 {degree: 95}]->("player101" :player{age: 36, name: "Tony Parker"})-[:follow@0 {degree: 95}]-
>("player125" :player{age: 41, name: "Manu Ginobili"})>     | 2      |
+-----------------------------------------------------------------------+--------+
| <("player100" :player{age: 42, name: "Tim Duncan"})-[:follow@0 {degree: 95}]->("player101" :player{age: 36, name: "Tony Parker"})-[:follow@0 {degree: 90}]-
>("player102" :player{age: 33, name: "LaMarcus Aldridge"})> | 2      |
+-----------------------------------------------------------------------+--------+
| <("player100" :player{age: 42, name: "Tim Duncan"})-[:follow@0 {degree: 95}]->("player101" :player{age: 36, name: "Tony Parker"})-[:follow@0 {degree: 95}]-
>("player100" :player{age: 42, name: "Tim Duncan"})>        | 2      |
+-----------------------------------------------------------------------+--------+
| <("player100" :player{age: 42, name: "Tim Duncan"})-[:serve@0 {end_year: 2016, start_year: 1997}]->("team204" :team{name:
"Spurs"})>                                                              | 1      |
+-----------------------------------------------------------------------+--------+
| <("player100" :player{age: 42, name: "Tim Duncan"})-[:follow@0 {degree: 95}]->("player125" :player{age: 41, name: "Manu
Ginobili"})>                                                              | 1      |
+-----------------------------------------------------------------------+--------+
```

Last update: April 22, 2021

4.6.2 LOOKUP

The `LOOKUP` statement retrieves data based on indexes.

You can use `LOOKUP` for the following purposes:

- Search for the specific data based on conditions defined by the `WHERE` clause.
- List vertices with a tag: retrieve the VID of all vertices with a tag.
- List edges with an edge type: retrieve the source Vertex IDs, destination vertex IDs, and ranks of all edges with an edge type.
- Count the number of vertices or edges with a tag or an edge type.

OpenCypher compatibility

This page applies to nGQL extensions only.

Prerequisites

Before using the `LOOKUP` statement, make sure that relative indexes are created. For how to create indexes, see CREATE INDEX.

Syntax

```
LOOKUP ON {<vertex_tag> | <edge_type>} [WHERE <expression> [AND <expression> ...]] [YIELD <return_list>]

<return_list>
    <prop_name> [AS <col_alias>] [, <prop_name> [AS <prop_alias>] ...]
```

- The `WHERE` clause filters data with the specified conditions. Both `AND` and `OR` are supported between different expressions. For more information, see WHERE.
- The `YIELD` clause specifies the results to be returned and the format of the results.
- If there is a `WHERE` clause but no `YIELD` clause:
  - The Vertex ID is returned when `LOOKUP` a tag.
  - The source vertex ID, destination vertex ID, and rank of the edge is returned when `LOOKUP` an edge type.

Limitations of using `WHERE` in `LOOKUP`

The `WHERE` clause in a `LOOKUP` statement does not support the following operations:

- `$-` and `$^`.
- In relational expressions, expressions with field names on both sides of the operator are not supported, such as `tagName.prop1> tagName.prop2`.
- Nested AliasProp expressions in operation expressions and function expressions are not supported.
- Expressions that match string-type index, such as `starts with` `ends with` `contains`.
- The `OR` and `XOR` operations are not supported.

Retrieve Vertices

The following example returns vertices whose name is `Tony Parker` and tagged with *player*.

```
nebula> CREATE TAG INDEX index_player ON player(name(30), age);

nebula> REBUILD TAG INDEX index_player;
+------------+
| New Job Id |
+------------+
| 15         |
+------------+

nebula> LOOKUP ON player WHERE player.name == "Tony Parker";
```

```
============
| VertexID |
============
| 101      |
------------

nebula> LOOKUP ON player WHERE player.name == "Tony Parker" \
        YIELD player.name, player.age;
=====================================
| VertexID | player.name | player.age |
=====================================
| 101      | Tony Parker | 36         |
-------------------------------------

nebula> LOOKUP ON player WHERE player.name == "Kobe Bryant" YIELD player.name AS name \
        | GO FROM $-.VertexID OVER serve \
        YIELD $-.name, serve.start_year, serve.end_year, $$.team.name;
=============================================================
| $-.name     | serve.start_year | serve.end_year | $$.team.name |
=============================================================
| Kobe Bryant | 1996             | 2016           | Lakers       |
-------------------------------------------------------------
```

Retrieve Edges

The following example returns edges whose `degree` is 90 and the edge type is `follow`.

```
nebula> CREATE EDGE INDEX index_follow ON follow(degree);

nebula> REBUILD EDGE INDEX index_follow;
+------------+
| New Job Id |
+------------+
| 62         |
+------------+

nebula> LOOKUP ON follow WHERE follow.degree == 90;
============================
| SrcVID | DstVID | Ranking |
============================
| 100    | 106    | 0       |
----------------------------

nebula> LOOKUP ON follow WHERE follow.degree == 90 YIELD follow.degree;
=========================================
| SrcVID | DstVID | Ranking | follow.degree |
=========================================
| 100    | 106    | 0       | 90            |
-----------------------------------------

nebula> LOOKUP ON follow WHERE follow.degree == 60 YIELD follow.degree AS Degree \
        | GO FROM $-.DstVID OVER serve \
        YIELD $-.DstVID, serve.start_year, serve.end_year, $$.team.name;
=============================================================
| $-.DstVID | serve.start_year | serve.end_year | $$.team.name |
=============================================================
| 105       | 2010             | 2018           | Spurs        |
-------------------------------------------------------------
| 105       | 2009             | 2010           | Cavaliers    |
-------------------------------------------------------------
| 105       | 2018             | 2019           | Raptors      |
-------------------------------------------------------------
```

List vertices or edges with a tag or an edge type

To list vertices or edges with a tag or an edge type, at least one index must exist on the tag or the edge type, or its property.

For example, if there is a `player` tag with a `name` property and an `age` property, to retrieve the VID of all vertices tagged with `player`, there has to be an index on the `player` tag itself, the `name` property, or the `age` property.

The following example shows how to retrieve the VID of all vertices tagged with `player`.

```
nebula> CREATE TAG player(name string,age int);
Execution succeeded (time spent 3235/3865 us)

nebula> CREATE TAG INDEX player_index on player();
Execution succeeded (time spent 3486/4124 us)

nebula> REBUILD TAG INDEX player_index;
+------------+
| New Job Id |
+------------+
| 66         |
+------------+
```

```
nebula> INSERT VERTEX player(name,age) VALUES "player100":("Tim Duncan", 42), "player101":("Tony Parker", 36);
Execution succeeded (time spent 1695/2268 us)

nebula> LOOKUP ON player;
+-------------+
| _vid        |
+-------------+
| "player100" |
+-------------+
| "player101" |
+-------------+
Got 2 rows (time spent 1514/2070 us)
```

The following example shows how to retrieve the source Vertex IDs, destination vertex IDs, and ranks of all edges of the `like` edge type.

```
nebula)> CREATE EDGE like(likeness int);
Execution succeeded (time spent 3710/4483 us)

nebula)> CREATE EDGE INDEX like_index on like();
Execution succeeded (time spent 3422/4026 us)

nebula> REBUILD EDGE INDEX like_index;
+------------+
| New Job Id |
+------------+
| 88         |
+------------+

nebula)> INSERT EDGE like(likeness) values "player100"->"player101":(95);
Execution succeeded (time spent 1638/2351 us)

nebula)> LOOKUP ON like;
+-------------+----------+-------------+
| _src        | _ranking | _dst        |
+-------------+----------+-------------+
| "player100" | 0        | "player101" |
+-------------+----------+-------------+
Got 1 rows (time spent 1163/1748 us)
```

Count the numbers of vertices or edges

The following example shows how to count the number of vertices tagged with `player` and edges of the `like` edge type.

```
nebula> LOOKUP ON player | YIELD COUNT(*) AS Player_Number;
+---------------+
| Player_Number |
+---------------+
| 2             |
+---------------+
Got 1 rows (time spent 1158/1864 us)

nebula> LOOKUP ON like | YIELD COUNT(*) AS Like_Number;
+-------------+
| Like_Number |
+-------------+
| 1           |
+-------------+
Got 1 rows (time spent 1190/1970 us)
```

Last update: May 20, 2021

4.6.3 GO

OpenCypher compatibility

This page applies to nGQL extensions only.

Syntax

```
GO [[<M> TO] <N> STEPS ] FROM <vertex_list>
OVER <edge_type_list> [{REVERSELY | BIDIRECT}]
[ WHERE <expression> [ {AND | OR} expression ...]) ]
[YIELD [DISTINCT] <return_list>]
[| ORDER BY <expression> [{ASC | DESC}]]
[| LIMIT [<offset_value>,] <number_rows>]

GO [[<M> TO] <N> STEPS ] FROM <vertex_list>
OVER <edge_type_list> [{REVERSELY | BIDIRECT}]
[ WHERE <conditions>]
[| GROUP BY {col_name | expr | position} YIELD <col_name>]

<vertex_list> ::=
    <vid> [, <vid> ...]

<edge_type_list> ::=
   edge_type [, edge_type ...]
   | *

<return_list> ::=
    <col_name> [AS <col_alias>] [, <col_name> [AS <col_alias>] ...]
```

`GO` traverses in a graph with specified filters and returns results.

- `<N> STEPS` specifies the hop number. If not specified, the default value for N is one. When `N` is zero, Nebula Graph does not traverse any edges and returns nothing.
- `M TO N STEPS` traverses from M to N hops. When `M` is zero, the output is the same as that of `M` is one. That is, the output of `GO 0 TO 2` and `GO 1 TO 2` are the same.
- `<vertex_list>` is a list of vertex IDs separated by commas, or a special place holder `$-.id`. For more information, see Pipe.
- `<edge_type_list>` is a list of edge types which the traversal can go through.
- `REVERSELY | BIDIRECT` defines the direction of the query. By default, `GO` statements searches for outgoing edges. If `REVERSELY` is set, `GO` searches for incoming edges. If `BIDIRECT` is set, `GO` searches for edges of both directions.
- `WHERE <expression>` specifies the traversal filters. You can use `WHERE` for the source vertices, the edges, and the destination vertices. You can use `WHERE` together with `AND`, `OR`, and `NOT`. For more information, see WHERE.

  > **Note**
  >
  > There are some restrictions for the `WHERE` clause when you traverse along with multiple edge types. For example, `WHERE edge1.prop1 > edge2.prop2` is not supported.

- `YIELD [DISTINCT] <return_list>` specifies the desired output. For more information, see YIELD. When not specified, the destination vertex IDs are returned by default.
- `ORDER BY` sorts the outputs with the specified orders. For more information, see ORDER BY.

  > **Note**
  >
  > When the sorting method is not specified, the output orders can be different for the same query.

- `LIMIT` limits the row numbers for the output. For more information, see LIMIT.
- `GROUP BY` groups outputs into subgroups based on values of the specified properties. For more information, see GROUP BY.

Examples

```
// Returns teams that player 102 serves.
nebula> GO FROM "player102" OVER serve;
+------------+
| serve._dst |
+------------+
| "team203"  |
+------------+
| "team204"  |
+------------+
```

```
// Returns the 2 hop friends of the player 102.
nebula> GO 2 STEPS FROM "player102" OVER follow;
+-------------+
| follow._dst |
+-------------+
| "player101" |
+-------------+
| "player125" |
+-------------+
...
```

```
// Adds a filter for the traversal then duplicates the output.
nebula> GO FROM "player100", "player102" OVER serve \
WHERE serve.start_year > 1995 \
YIELD DISTINCT $$.team.name AS team_name, serve.start_year AS start_year, $^.player.name AS player_name;
+-----------------+------------+--------------------+
| team_name       | start_year | player_name        |
+-----------------+------------+--------------------+
| "Spurs"         | 1997       | "Tim Duncan"       |
+-----------------+------------+--------------------+
| "Trail Blazers" | 2006       | "LaMarcus Aldridge" |
+-----------------+------------+--------------------+
| "Spurs"         | 2015       | "LaMarcus Aldridge" |
+-----------------+------------+--------------------+
```

```
// Traverses along with multiple edge types.
nebula> GO FROM "player100" OVER follow, serve YIELD follow.degree, serve.start_year;
+---------------+-----------------+
| follow.degree | serve.start_year |
+---------------+-----------------+
| 95            | __EMPTY__       |
+---------------+-----------------+
| 95            | __EMPTY__       |
+---------------+-----------------+
| __EMPTY__     | 1997            |
+---------------+-----------------+
```

Nebula Graph displays different properties by columns. If there is no value for a property, the output is `__EMPTY__`.

```
// Returns player 100.
nebula> GO FROM "player100" OVER follow REVERSELY YIELD follow._dst AS destination;
+-------------+
| destination |
+-------------+
| "player101" |
+-------------+
| "player102" |
+-------------+
...

// This MATCH query shares the same semantics with the preceding GO query.
nebula> MATCH (v)<-[e:follow]- (v2) WHERE id(v) == 'player100' RETURN id(v2) AS destination;
+-------------+
| destination |
+-------------+
| "player101" |
+-------------+
| "player102" |
+-------------+
...
```

```
// Finds player 100's friends and the teams that they serve.
nebula> GO FROM "player100" OVER follow REVERSELY \
YIELD follow._dst AS id | \
GO FROM $-.id OVER serve \
WHERE $^.player.age > 20 \
YIELD $^.player.name AS FriendOf, $$.team.name AS Team;
+--------------------+-----------------+
| FriendOf           | Team            |
+--------------------+-----------------+
| "Tony Parker"      | "Spurs"         |
+--------------------+-----------------+
| "Tony Parker"      | "Hornets"       |
+--------------------+-----------------+
...

// This MATCH query shares the same semantics with the preceding GO query.
nebula> MATCH (v)<-[e:follow]- (v2)-[e2:serve]->(v3)  WHERE id(v) == 'player100' RETURN v2.name AS FriendOf, v3.name AS Team;
```

```
+---------------------+-----------------+
| FriendOf            | Team            |
+---------------------+-----------------+
| "Tony Parker"       | "Spurs"         |
+---------------------+-----------------+
| "Tony Parker"       | "Hornets"       |
+---------------------+-----------------+
...
```

```
nebula> GO FROM "player102" OVER follow BIDIRECT YIELD follow._dst AS both;
+-------------+
| both        |
+-------------+
| "player100" |
+-------------+
| "player101" |
+-------------+
...

// This MATCH query shares the same semantics with the preceding GO query.
nebula> MATCH (v) -[e:follow]-(v2) WHERE id(v)== "player102" RETURN id(v2) AS both;
+-------------+
| both        |
+-------------+
| "player101" |
+-------------+
| "player103" |
+-------------+
...
```

```
nebula> GO 1 TO 2 STEPS FROM "player100" OVER follow YIELD follow._dst AS destination;
+-------------+
| destination |
+-------------+
| "player101" |
+-------------+
| "player125" |
+-------------+
...

// This MATCH query shares the same semantics with the preceding GO query.
nebula> MATCH (v) -[e:follow*1..2]->(v2) WHERE id(v) == "player100" RETURN id(v2) AS destination;
+-------------+
| destination |
+-------------+
| "player100" |
+-------------+
| "player102" |
+-------------+
```

```
nebula> GO 2 STEPS FROM "player100" OVER follow \
YIELD follow._src AS src, follow._dst AS dst, $$.player.age AS age \
| GROUP BY $-.dst YIELD $-.dst AS dst, collect_set($-.src) AS src, collect($-.age) AS age
+-------------+---------------------------+----------+
| dst         | src                       | age      |
+-------------+---------------------------+----------+
| "player125" | ["player101"]             | [41]     |
+-------------+---------------------------+----------+
| "player100" | ["player125", "player101"] | [42, 42] |
+-------------+---------------------------+----------+
| "player102" | ["player101"]             | [33]     |
+-------------+---------------------------+----------+
```

```
nebula> $a = GO FROM "player100" OVER follow YIELD follow._src AS src, follow._dst AS dst; \
GO 2 STEPS FROM $a.dst OVER follow YIELD $a.src AS src, $a.dst, follow._src, follow._dst \
| ORDER BY $-.src | OFFSET 1 LIMIT 2;
+-------------+-------------+-------------+-------------+
| src         | $a.dst      | follow._src | follow._dst |
+-------------+-------------+-------------+-------------+
| "player100" | "player125" | "player100" | "player101" |
+-------------+-------------+-------------+-------------+
| "player100" | "player101" | "player100" | "player125" |
+-------------+-------------+-------------+-------------+
```

Last update: April 22, 2021

4.6.4 FETCH

The `FETCH` statement retrieves the properties of the specified vertices or edges.

OpenCypher Compatibility

This topic applies to nGQL extensions only.

Fetch vertex properties

SYNTAX

```
FETCH PROP ON {<tag_name>[, tag_name ...] | *}
<vid> [, vid ...]
[YIELD <output>]
```

The descriptions of the fields are as follows.

Field	Description
`tag_name`	The name of the tag.
`*`	Represents all the tags in the current graph space.
`vid`	The vertex ID.
`output`	Specifies the information to be returned. For more information, see `YIELD`. If there is no `YIELD` clause, `FETCH` returns all the matched information.

FETCH VERTEX PROPERTIES BY ONE TAG

Specify a tag in the `FETCH` statement to fetch the vertex properties by that tag.

```
nebula> FETCH PROP ON player "player100";
+-------------------------------------------------+
| vertices_                                       |
+-------------------------------------------------+
| ("player100" :player{age: 42, name: "Tim Duncan"}) |
+-------------------------------------------------+
Got 1 rows (time spent 913/1629 us)
```

FETCH SPECIFIC PROPERTIES OF A VERTEX

Use a `YIELD` clause to specify the properties to be returned.

```
nebula> FETCH PROP ON player "player100" \
        YIELD player.name;
+-------------+--------------+
| VertexID    | player.name  |
+-------------+--------------+
| "player100" | "Tim Duncan" |
+-------------+--------------+
Got 1 rows (time spent 2933/5931 us)
```

FETCH PROPERTIES OF MULTIPLE VERTICES

Specify multiple VIDs (vertex IDs) to fetch properties of multiple vertices. Separate the VIDs with commas.

```
nebula> FETCH PROP ON player "player101", "player102", "player103";
+-------------------------------------------------------+
| vertices_                                             |
+-------------------------------------------------------+
| ("player101" :player{age: 36, name: "Tony Parker"})   |
+-------------------------------------------------------+
| ("player102" :player{age: 33, name: "LaMarcus Aldridge"}) |
+-------------------------------------------------------+
| ("player103" :player{age: 32, name: "Rudy Gay"})      |
+-------------------------------------------------------+
Got 3 rows (time spent 1786/3135 us)
```

FETCH VERTEX PROPERTIES BY MULTIPLE TAGS

Specify multiple tags in the `FETCH` statement to fetch the vertex properties by the tags. Separate the tags with commas.

```
// Create a new tag t1.
nebula> CREATE TAG t1(a string, b int);
Execution succeeded (time spent 4153/5296 us)

// Attach t1 to vertex "player100".
nebula> INSERT VERTEX t1(a, b) VALUE "player100":("Hello", 100);
Execution succeeded (time spent 1703/2321 us)

// Fetch the properties of vertex "player100" by the tags player and t1.
nebula> FETCH PROP ON player, t1 "player100";
+-------------------------------------------------------------------------------+
| vertices_                                                                     |
+-------------------------------------------------------------------------------+
| ("player100" :t1{a: "Hello", b: 100} :player{age: 42, name: "Tim Duncan"}) |
+-------------------------------------------------------------------------------+
Got 1 rows (time spent 1788/2560 us)
```

You can combine multiple tags with multiple VIDs in a `FETCH` statement.

```
nebula> FETCH PROP ON player, t1 "player100", "player103";
+-------------------------------------------------------------------------------+
| vertices_                                                                     |
+-------------------------------------------------------------------------------+
| ("player100" :t1{a: "Hello", b: 100} :player{age: 42, name: "Tim Duncan"}) |
+-------------------------------------------------------------------------------+
| ("player103" :player{age: 32, name: "Rudy Gay"})                          |
+-------------------------------------------------------------------------------+
Got 2 rows (time spent 2971/3748 us)
```

FETCH VERTEX PROPERTIES BY ALL TAGS

Set an asterisk symbol (*) to fetch properties by all tags in the current graph space.

```
nebula> FETCH PROP ON * "player100", "player106", "team200";
+-------------------------------------------------------------------------------+
| vertices_                                                                     |
+-------------------------------------------------------------------------------+
| ("player106" :player{age: 25, name: "Kyle Anderson"})                     |
+-------------------------------------------------------------------------------+
| ("team200" :team{name: "Warriors"})                                       |
+-------------------------------------------------------------------------------+
| ("player100" :t1{a: "Hello", b: 100} :player{age: 42, name: "Tim Duncan"}) |
+-------------------------------------------------------------------------------+
Got 3 rows (time spent 2620/4863 us)
```

Fetch edge properties

SYNTAX

```
FETCH PROP ON <edge_type> <src_vid> -> <dst_vid>[@<rank>] [, <src_vid> -> <dst_vid> ...]
[YIELD <output>]
```

The descriptions of the fields are as follows.

Field	Description
`edge_type`	The name of the edge type.
`src_vid`	The VID of the source vertex. It specifies the start of an edge.
`dst_vid`	The VID of the destination vertex. It specifies the end of an edge.
`rank`	The rank of the edge. It is optional and defaults to 0. It distinguishes an edge from other edges with the same edge type, source vertex, and destination vertex.
`output`	Specifies the information to be returned. For more information, see `YIELD`. If there is no `YIELD` clause, `FETCH` returns all the matched information.

FETCH ALL PROPERTIES OF AN EDGE

The following statement fetches all the properties of the `serve` edge that connects vertex `"player100"` and vertex `"team204"`.

```
nebula> FETCH PROP ON serve "player100" -> "team204";
+------------------------------------------------------------------------+
| edges_                                                                 |
+------------------------------------------------------------------------+
| [:serve "player100"->"team204" @0 {end_year: 2016, start_year: 1997}] |
+------------------------------------------------------------------------+
Got 1 rows (time spent 1048/1632 us)
```

FETCH SPECIFIC PROPERTIES OF AN EDGE

Use a `YIELD` clause to fetch specific properties of an edge.

```
nebula> FETCH PROP ON serve "player100" -> "team204" YIELD serve.start_year;
+-------------+------------+-------------+------------------+
| serve._src  | serve._dst | serve._rank | serve.start_year |
+-------------+------------+-------------+------------------+
| "player100" | "team204"  | 0           | 1997             |
+-------------+------------+-------------+------------------+
Got 1 rows (time spent 1834/2863 us)
```

FETCH PROPERTIES OF MULTIPLE EDGES

Specify multiple edge patterns (`<src_vid> -> <dst_vid>[@<rank>]`) to fetch properties of multiple edges. Separate the edge patterns with commas.

```
nebula> FETCH PROP ON serve "player100" -> "team204", "player133" -> "team202";
+------------------------------------------------------------------------+
| edges_                                                                 |
+------------------------------------------------------------------------+
| [:serve "player100"->"team204" @0 {end_year: 2016, start_year: 1997}] |
+------------------------------------------------------------------------+
| [:serve "player133"->"team202" @0 {end_year: 2011, start_year: 2002}] |
+------------------------------------------------------------------------+
Got 2 rows (time spent 1466/2441 us)
```

Fetch properties based on edge rank

If there are multiple edges that have different ranks but the same edge type, source vertex, destination vertex, specify the rank to fetch the properties on the correct edge.

```
// Insert edges with different ranks and property values.
nebula> insert edge serve(start_year,end_year) \
        values "player100"->"team204"@1:(1998, 2017);
Execution succeeded (time spent 1679/3192 us)

nebula> insert edge serve(start_year,end_year) \
        values "player100"->"team204"@2:(1990, 2018);
Execution succeeded (time spent 1091/1608 us)

// By default, FETCH returns the edge with rank 0.
nebula> FETCH PROP ON serve "player100" -> "team204";
+------------------------------------------------------------------------+
| edges_                                                                 |
+------------------------------------------------------------------------+
| [:serve "player100"->"team204" @0 {end_year: 2016, start_year: 1997}] |
+------------------------------------------------------------------------+
Got 1 rows (time spent 2031/2739 us)

// To fetch on an edge with rank other than 0, set its rank in FETCH.
nebula> FETCH PROP ON serve "player100" -> "team204"@1;
+------------------------------------------------------------------------+
| edges_                                                                 |
+------------------------------------------------------------------------+
| [:serve "player100"->"team204" @1 {end_year: 2017, start_year: 1998}] |
+------------------------------------------------------------------------+
Got 1 rows (time spent 1049/1711 us)
```

Use FETCH in composite queries

A common way to use `FETCH` is to combine it with nGQL extensions such as GO. The following statement returns the `degree` values of outgoing `follow` edges that start from vertex `"player101"`.

```
nebula> GO FROM "player101" OVER follow \
        YIELD follow._src AS s, follow._dst AS d | \
        FETCH PROP ON follow $-.s -> $-.d \
        YIELD follow.degree;
+-------------+-------------+--------------+---------------+
| follow._src | follow._dst | follow._rank | follow.degree |
+-------------+-------------+--------------+---------------+
```

```
| "player101" | "player100" | 0            | 95           |
+-------------+-------------+--------------+--------------+
| "player101" | "player102" | 0            | 90           |
+-------------+-------------+--------------+--------------+
| "player101" | "player125" | 0            | 95           |
+-------------+-------------+--------------+--------------+
Got 3 rows (time spent 3047/3880 us)
```

Or you can use user-defined variables to construct similar queries.

```
nebula> $var = GO FROM "player101" OVER follow \
        YIELD follow._src AS s, follow._dst AS d; \
        FETCH PROP ON follow $var.s -> $var.d \
        YIELD follow.degree;
+-------------+-------------+--------------+--------------+
| follow._src | follow._dst | follow._rank | follow.degree |
+-------------+-------------+--------------+--------------+
| "player101" | "player100" | 0            | 95           |
+-------------+-------------+--------------+--------------+
| "player101" | "player102" | 0            | 90           |
+-------------+-------------+--------------+--------------+
| "player101" | "player125" | 0            | 95           |
+-------------+-------------+--------------+--------------+
Got 3 rows (time spent 1891/2509 us)
```

For more information about composite queries, see Composite queries (clause structure).

Last update: March 29, 2021

4.6.5 UNWIND

The `UNWIND` statement splits a list into separated rows.

`UNWIND` can function as an individual statement or a clause in a statement.

Syntax

```
UNWIND <list> AS <alias> <RETURN clause>
```

Split a list

The following example splits the list `[1,2,3]` into three rows.

```
nebula) [basketballplayer]> UNWIND [1,2,3] AS n RETURN n;
+---+
| n |
+---+
| 1 |
+---+
| 2 |
+---+
| 3 |
+---+
Got 3 rows (time spent 806/2126 us)
```

Return a list with distinct items

Use `UNWIND` and `WITH DISTINCT` together to return a list with distinct items.

EXAMPLE 1

The following statement:

1. Splits the list `[1,1,2,2,3,3]` into rows.
2. Removes duplicated rows.
3. Sorts the rows.
4. Transforms the rows to a list.

```
nebula> WITH [1,1,2,2,3,3] AS n \
        UNWIND n AS r \
        WITH DISTINCT r AS r \
        ORDER BY r \
        RETURN collect(r);
+------------+
| COLLECT(r) |
+------------+
| [1, 2, 3]  |
+------------+
Got 1 rows (time spent 307/1043 us)
```

Example 2

The following statement:

1. Outputs the vertices on the matched path into a list.
2. Splits the list into rows.
3. Removes duplicated rows.
4. Transforms the rows to a list.

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})--(v2) \
        WITH nodes(p) AS n \
        UNWIND n AS r \
        WITH DISTINCT r AS r \
        RETURN collect(r);
+-----------------------------------------------------------------------------------------------------------------------+
```

```
| COLLECT(r)                                                                                                          |
+---------------------------------------------------------------------------------------------------------------------+
| [("player100" :player{age: 42, name: "Tim Duncan"}), ("player101" :player{age: 36, name: "Tony Parker"}),
("team204" :team{name: "Spurs"}), ("player102" :player{age: 33, name: "LaMarcus Aldridge"}),
("player125" :player{age: 41, name: "Manu Ginobili"}), ("player104" :player{age: 32, name: "Marco Belinelli"}),
("player144" :player{age: 47, name: "Shaquile O'Neal"}), ("player105" :player{age: 31, name: "Danny Green"}),
("player113" :player{age: 29, name: "Dejounte Murray"}), ("player107" :player{age: 32, name: "Aron Baynes"}),
("player109" :player{age: 34, name: "Tiago Splitter"}), ("player108" :player{age: 36, name: "Boris Diaw"})] |
+---------------------------------------------------------------------------------------------------------------------+
Got 1 rows (time spent 6157/6833 us)
```

Last update: April 13, 2021

4.6.6 SHOW

SHOW CHARSET

The `SHOW CHARSET` statement shows the available character sets.

Currently available types are utf8 and utf8mb4. The default charset type is utf8. Nebula Graph extends the uft8 to support four-byte characters. Therefore utf8 and utf8mb4 are equivalent.

SYNTAX

```
SHOW CHARSET
```

EXAMPLE

```
nebula> SHOW CHARSET;
+---------+-----------------+-------------------+--------+
| Charset | Description     | Default collation | Maxlen |
+---------+-----------------+-------------------+--------+
| "utf8"  | "UTF-8 Unicode" | "utf8_bin"        | 4      |
+---------+-----------------+-------------------+--------+
Got 1 rows (time spent 527/1269 us)
```

The output of `SHOW CHARSET` is explained as follows:

Column	Description
`Charset`	The character set name.
`Description`	A description of the character set.
`Default collation`	The default collation for the character set.
`Maxlen`	The maximum number of bytes required to store one character.

Last update: December 31, 2020

SHOW COLLATION

The `SHOW COLLATION` statement shows the collations supported by Nebula Graph.

Currently available types are: utf8_bin, utf8_general_ci, utf8mb4_bin, and utf8mb4_general_ci. When the character set is utf8, the default collate is utf8_bin; when the character set is utf8mb4, the default collate is utf8mb4_bin. Both utf8_general_ci and utf8mb4_general_ci are case-insensitive.

SYNTAX

```
SHOW COLLATION
```

EXAMPLE

```
nebula> SHOW COLLATION;
+------------+---------+
| Collation  | Charset |
+------------+---------+
| "utf8_bin" | "utf8"  |
+------------+---------+
Got 1 rows (time spent 413/1034 us)
```

The output of `SHOW CHARSET` is described as follows:

Column	Description
`Collation`	The collation name.
`Charset`	The name of the character set with which the collation is associated.

Last update: December 31, 2020

SHOW CREATE SPACE

The `SHOW CREATE SPACE` statement shows the basic information of the specified graph space, such as the nGQL for creating the graph space, the partition number, the replica number.

For details about the graph space information, see CREATE SPACE.

SYNTAX

```
SHOW CREATE SPACE <space_name>
```

EXAMPLE

```
nebula> SHOW CREATE SPACE basketballplayer;
+--------------------+-----------------------------------------------------------------------------------------------------------------------------------------------------------------+
| Space              | Create Space                                                                                                                                                    |
+--------------------+-----------------------------------------------------------------------------------------------------------------------------------------------------------------+
| "basketballplayer" | "CREATE SPACE `basketballplayer` (partition_num = 10, replica_factor = 1, charset = utf8, collate = utf8_bin, vid_type = FIXED_STRING(32))" |
+--------------------+-----------------------------------------------------------------------------------------------------------------------------------------------------------------+
Got 1 rows (time spent 1747/2562 us)
```

Last update: April 13, 2021

SHOW CREATE TAG/EDGE

The `SHOW CREATE TAG` or `SHOW CREATE EDGE` statement shows the basic information of the specified tag or edge type.

For details about the tag or edge type information, see CREATE TAG and CREATE EDGE.

SYNTAX

```
SHOW CREATE {TAG <tag_name> | EDGE <edge_name>}
```

EXAMPLE

```
nebula> SHOW CREATE TAG player;
+----------+----------------------------------+
| Tag      | Create Tag                       |
+----------+----------------------------------+
| "player" | "CREATE TAG `player` (           |
|          |  `name` string NULL,             |
|          |  `age` int64 NULL                |
|          | ) ttl_duration = 0, ttl_col = "" |
+----------+----------------------------------+
```

Last update: March 29, 2021

SHOW HOSTS

The `SHOW HOSTS` statement lists graph/storage/meta hosts registered by the Meta Service.

SYNTAX

```
SHOW HOSTS [GRAPH/STORAGE/META]
```

EXAMPLE

```
nebula> SHOW HOSTS;
+-------------+-------+----------+--------------+---------------------------------+-----------------------------+
| Host        | Port  | Status   | Leader count | Leader distribution             | Partition distribution      |
+-------------+-------+----------+--------------+---------------------------------+-----------------------------+
| "storaged0" | 9779  | "ONLINE" | 8            | "docs:5, basketballplayer:3"    | "docs:5, basketballplayer:3" |
+-------------+-------+----------+--------------+---------------------------------+-----------------------------+
| "storaged1" | 9779  | "ONLINE" | 9            | "basketballplayer:4, docs:5"    | "docs:5, -basketballplayer:4"|
+-------------+-------+----------+--------------+---------------------------------+-----------------------------+
| "storaged2" | 9779  | "ONLINE" | 8            | "basketballplayer:3, docs:5"    | "docs:5, basketballplayer:3" |
+-------------+-------+----------+--------------+---------------------------------+-----------------------------+
Got 3 rows (time spent 866/1411 us)

nebula> SHOW HOSTS GRAPH;
+-------------+------+----------+---------+--------------+
| Host        | Port | Status   | Role    | Git Info Sha |
+-------------+------+----------+---------+--------------+
| "12.16.2.3" | 9669 | "ONLINE" | "GRAPH" | "761f22b"    |

nebula> SHOW HOSTS STORAGE;
+-------------+------+----------+-----------+--------------+
| Host        | Port | Status   | Role      | Git Info Sha |
+-------------+------+----------+-----------+--------------+
| "12.16.2.3" | 9779 | "ONLINE" | "STORAGE" | "761f22b"    |

nebula> SHOW HOSTS META;
+-------------+------+----------+--------+--------------+
| Host        | Port | Status   | Role   | Git Info Sha |
+-------------+------+----------+--------+--------------+
| "12.16.2.3" | 9559 | "ONLINE" | "META" | "761f22b"    |
```

Last update: April 13, 2021

SHOW INDEX STATUS

The `SHOW INDEX STATUS` statement shows the status of jobs that rebuild native indexes. You can find out whether a native index is successfully rebuilt or not.

SYNTAX

```
SHOW {TAG | EDGE} INDEX STATUS
```

EXAMPLE

```
nebula> SHOW TAG INDEX STATUS;
+---------------+-------------+
| Name          | Index Status |
+---------------+-------------+
| "like_index_0" | "FINISHED"   |
+---------------+-------------+
| "like1"        | "FINISHED"   |
+---------------+-------------+
Got 2 rows (time spent 1456/2122 us)
```

RELATED TOPICS

- Job manager and the JOB statements
- REBUILD NATIVE INDEX

Last update: January 22, 2021

SHOW INDEXES

The `SHOW INDEXES` statement shows the names of existing native indexes.

SYNTAX

```
SHOW {TAG | EDGE} INDEXES
```

EXAMPLE

```
nebula> SHOW TAG INDEXES;
+------------------+
| Names            |
+------------------+
| "play_age_0"     |
+------------------+
| "player_index_0" |
+------------------+
| "player_index_1" |
+------------------+
| "star"           |
+------------------+
Got 4 rows (time spent 1450/2087 us)
```

Last update: December 31, 2020

SHOW PARTS

The `SHOW PARTS` statement shows the information of a specified partition or all partitions in a graph space.

SYNTAX

```
SHOW PARTS [<part_id>]
```

EXAMPLES

Show the information of all partitions:

```
nebula> SHOW PARTS;
+-------------+------------------+------------------+-------+
| Partition ID | Leader           | Peers            | Losts |
+-------------+------------------+------------------+-------+
| 1           | "storaged1:44500" | "storaged1:44500" | ""    |
+-------------+------------------+------------------+-------+
| 2           | "storaged2:44500" | "storaged2:44500" | ""    |
+-------------+------------------+------------------+-------+
| 3           | "storaged0:44500" | "storaged0:44500" | ""    |
+-------------+------------------+------------------+-------+
| 4           | "storaged1:44500" | "storaged1:44500" | ""    |
+-------------+------------------+------------------+-------+
| 5           | "storaged2:44500" | "storaged2:44500" | ""    |
+-------------+------------------+------------------+-------+
| 6           | "storaged0:44500" | "storaged0:44500" | ""    |
+-------------+------------------+------------------+-------+
| 7           | "storaged1:44500" | "storaged1:44500" | ""    |
+-------------+------------------+------------------+-------+
| 8           | "storaged2:44500" | "storaged2:44500" | ""    |
+-------------+------------------+------------------+-------+
| 9           | "storaged0:44500" | "storaged0:44500" | ""    |
+-------------+------------------+------------------+-------+
| 10          | "storaged1:44500" | "storaged1:44500" | ""    |
+-------------+------------------+------------------+-------+
Got 10 rows (time spent 2317/3512 us)
```

Show the information of partition 1:

```
nebula> SHOW PARTS 1;
+-------------+------------------+------------------+-------+
| Partition ID | Leader           | Peers            | Losts |
+-------------+------------------+------------------+-------+
| 1           | "storaged1:44500" | "storaged1:44500" | ""    |
+-------------+------------------+------------------+-------+
Got 1 rows (time spent 1055/1678 us)
```

Last update: December 31, 2020

SHOW ROLES

The `SHOW ROLES` statement shows the roles that are assigned to a user account.

The return message differs according to the role of the user who is running this statement:

- If the user is a `GOD` or `ADMIN` and is granted access to the specified graph space, Nebula Graph shows all roles in this graph space except for `GOD`.
- If the user is a `DBA`, `USER`, or `GUEST` and is granted access to the specified graph space, Nebula Graph shows the user's own role in this graph space.
- If the user doesn't have a role, `PermissionError` is returned.

For more information about user roles, see Roles and privileges.

SYNTAX

```
SHOW ROLES IN <space_name>
```

EXAMPLE

```
nebula> SHOW ROLES in basketballplayer;
+---------+-----------+
| Account | Role Type |
+---------+-----------+
| "user1" | "ADMIN"   |
+---------+-----------+
Got 1 rows (time spent 789/1594 us)
```

Last update: April 13, 2021

SHOW SNAPSHOTS

The `SHOW SNAPSHOTS` statement shows all the snapshots.

For how to create a snapshot and backup data, see Snapshot.

ROLE REQUIREMENT

Only the root user who has the GOD role can use this statement.

SYNTAX

```
SHOW SNAPSHOTS
```

EXAMPLE

```
nebula> SHOW SNAPSHOTS;
+-------------------------------+---------+-------------------------------------------------+
| Name                          | Status  | Hosts                                           |
+-------------------------------+---------+-------------------------------------------------+
| "SNAPSHOT_2020_12_16_11_13_55" | "VALID" | "storaged0:9779, storaged1:9779, storaged2:9779"    |
+-------------------------------+---------+-------------------------------------------------+
| "SNAPSHOT_2020_12_16_11_14_10" | "VALID" | "storaged0:9779, storaged1:9779, storaged2:9779"    |
+-------------------------------+---------+-------------------------------------------------+
Got 2 rows (time spent 762/1434 us)
```

Last update: March 25, 2021

SHOW SPACES

The `SHOW SPACES` statement shows the graph spaces in Nebula Graph.

For how to create a graph space, see CREATE SPACE.

SYNTAX

```
SHOW SPACES
```

EXAMPLE

```
nebula> SHOW SPACES;
+--------------------+
| Name               |
+--------------------+
| "docs"             |
+--------------------+
| "basketballplayer" |
+--------------------+
Got 2 rows (time spent 968/1893 us)
```

Last update: April 13, 2021

SHOW STATS

The `SHOW STATS` statement shows the statistics of the graph space collected by the latest `STATS` job.

The statistics list the following information:

- The number of vertices and edges in the graph space
- The number of vertices with each tag
- The number of edges of each edge type

PREREQUISITES

You have successfully run the `SUBMIT JOB STATS` statement in the graph space you want to collect statistics. For more information, see SUBMIT JOB STATS.

> **Note**
>
> The result of the `SHOW STATS` statement is based on the last executed `SUBMIT JOB STATS` statement. If you want to update the result, run `SUBMIT JOB STATS` again.

SYNTAX

```
SHOW STATS
```

EXAMPLE

```
nebula> USE basketballplayer;
Execution succeeded (time spent 1075/1646 us)

--Start a `STATS` job.
nebula> SUBMIT JOB STATS;
+------------+
| New Job Id |
+------------+
| 98         |
+------------+
Got 1 rows (time spent 2058/2609 us)

--Make sure the job is finished.
nebula> SHOW JOB 98;
+---------------+---------------+------------+------------+------------+
| Job Id(TaskId) | Command(Dest) | Status     | Start Time | Stop Time  |
+---------------+---------------+------------+------------+------------+
| 98             | "STATS"       | "FINISHED" | 1606552675 | 1606552675 |
+---------------+---------------+------------+------------+------------+
| 0              | "storaged2"   | "FINISHED" | 1606552675 | 1606552675 |
+---------------+---------------+------------+------------+------------+
| 1              | "storaged0"   | "FINISHED" | 1606552675 | 1606552675 |
+---------------+---------------+------------+------------+------------+
| 2              | "storaged1"   | "FINISHED" | 1606552675 | 1606552675 |
+---------------+---------------+------------+------------+------------+
Got 4 rows (time spent 1233/1924 us)

--Check the statistics.
nebula> SHOW STATS;
+---------+------------+-------+
| Type    | Name       | Count |
+---------+------------+-------+
| "Tag"   | "player"   | 51    |
+---------+------------+-------+
| "Tag"   | "team"     | 30    |
+---------+------------+-------+
| "Edge"  | "like"     | 81    |
+---------+------------+-------+
| "Edge"  | "serve"    | 152   |
+---------+------------+-------+
| "Space" | "vertices" | 81    |
+---------+------------+-------+
| "Space" | "edges"    | 233   |
+---------+------------+-------+
Got 6 rows (time spent 996/1637 us)
```

Last update: April 22, 2021

SHOW TAGS/EDGES

The `SHOW TAGS` or `SHOW EDGES` statement shows all tags or edge types in the current graph space.

SYNTAX

```
SHOW {TAGS | EDGES}
```

EXAMPLES

Show tags:

```
nebula> SHOW TAGS;
+----------+
| Name     |
+----------+
| "player" |
+----------+
| "star"   |
+----------+
| "team"   |
+----------+
Got 3 rows (time spent 1461/2114 us)
```

Show edge types

```
nebula> SHOW EDGES;
+---------+
| Name    |
+---------+
| "like"  |
+---------+
| "serve" |
+---------+
Got 2 rows (time spent 1039/1687 us)
```

Last update: December 31, 2020

SHOW USERS

The `SHOW USERS` statement shows the user information.

ROLE REQUIREMENT

Only the root user who has the `GOD` role can use this statement.

SYNTAX

```
SHOW USERS
```

EXAMPLE

```
nebula> SHOW USERS;
+---------+
| Account |
+---------+
| "root"  |
+---------+
| "user1" |
+---------+
Got 2 rows (time spent 964/1691 us)
```

Last update: December 31, 2020

4.7 Clauses and options

4.7.1 GROUP BY

OpenCypher Compatibility

This page applies to nGQL extensions only.

Use `GROUP BY` in nGQL-extensions **ONLY** to aggregate data.

OpenCypher uses the count() function to aggregate data.

```
nebula>  MATCH (v:player)<-[:follow]-(:player) RETURN v.name AS Name, count(*) as cnt ORDER BY cnt DESC
+----------------------+-------------+
| Name                 | Follower_Num |
+----------------------+-------------+
| "Tim Duncan"         | 10          |
+----------------------+-------------+
| "LeBron James"       | 6           |
+----------------------+-------------+
| "Tony Parker"        | 5           |
+----------------------+-------------+
| "Manu Ginobili"      | 4           |
+----------------------+-------------+
| "Chris Paul"         | 4           |
+----------------------+-------------+
| "Tracy McGrady"      | 3           |
+----------------------+-------------+
| "Dwyane Wade"        | 3           |
+----------------------+-------------+
...
```

Syntax

The `GROUP BY` clause groups the rows with the same value into summary rows. Then operations such as counting, sorting, and calculation can be applied.

`GROUP BY` works after the pipe symbol and before a `YIELD` clause.

```
| GROUP BY <var> YIELD <var>, <aggregation_function(var)>
```

- aggregation_function can be `avg(), sum(), max(), min(), count(), collect(), std()`.

Examples

The following statement finds all the vertices connected directly to vertex `"player100"`, groups the result set by player names, and counts the times that the names show up in the result set.

```
nebula> GO FROM "player100" \
        OVER follow BIDIRECT \
        YIELD $$.player.name as Name | \
        GROUP BY $-.Name \
        YIELD $-.Name as Player, count(*) AS Name_Count;
+--------------------+------------+
| Player             | Name_Count |
+--------------------+------------+
| "Tiago Splitter"   | 1          |
+--------------------+------------+
| "Aron Baynes"      | 1          |
+--------------------+------------+
| "Boris Diaw"       | 1          |
+--------------------+------------+
| "Manu Ginobili"    | 2          |
+--------------------+------------+
| "Dejounte Murray"  | 1          |
+--------------------+------------+
| "Danny Green"      | 1          |
+--------------------+------------+
| "Tony Parker"      | 2          |
+--------------------+------------+
| "Shaquille O'Neal"  | 1         |
+--------------------+------------+
| "LaMarcus Aldridge" | 1         |
+--------------------+------------+
```

```
| "Marco Belinelli"   | 1          |
+--------------------+------------+
Got 10 rows (time spent 3527/4423 us)
```

Group and calculate with functions

The following statement finds all the players followed by `"player100"`, returns these players as `player` and the property of the follow edge as `degree`. These players are grouped and the sum of their degree values is returned.

```
nebula> GO FROM "player100" OVER follow YIELD follow._src AS player, follow.degree AS degree | GROUP BY $-.player YIELD sum($-.degree);
+---------------+
| sum($-.degree) |
+---------------+
| 190            |
+---------------+
Got 1 rows (time spent 2851/3624 us)
```

For more information about functions, see Functions.

Last update: March 25, 2021

4.7.2 LIMIT AND SKIP

The `LIMIT` clause constrains the number of rows in the output.

The Syntax in openCypher and nGQL-extension are different.

- NGQL-extension: A pipe `|` must be used. And an offset can be ignored.
- OpenCypher style: No pipes are permitted. Use `Skip` to indicate offset.

> **Note**
>
> When using `LIMIT` (in either syntax above), it is important to use an `ORDER BY` clause that constrains the output into a unique order. Otherwise, you will get an unpredictable subset of the output.

nGQL-extension syntax

In nGQL-extension, `LIMIT` works the same as in `SQL`, and must be used with pipe `|`. The `LIMIT` clause accepts one or two arguments. The values of both arguments must be non-negative integers.

```
YIELD <var>
[| LIMIT [<offset_value>,] <number_rows>]
```

- var: The columns or calculations that you wish to sort.
- number_rows: It constrains the number of rows to return. For example, `LIMIT 10` would return the first 10 rows.
- offset_value(Optional): It defines from which row to start including the rows in the output. The offset starts from zero.

EXAMPLES

```
nebula> GO FROM "player100" OVER follow REVERSELY YIELD $$.player.name AS Friend, $$.player.age AS Age | ORDER BY Age,Friend | LIMIT 1, 3;
+------------------+-----+
| Friend           | Age |
+------------------+-----+
| "Danny Green"    | 31  |
+------------------+-----+
| "Aron Baynes"    | 32  |
+------------------+-----+
| "Marco Belinelli" | 32  |
+------------------+-----+
```

OpenCypher Syntax

```
RETURN <var>
[SKIP <offset>]
[LIMIT <number_rows>]
```

Parameter	Description
`offset`	Optional. It specifies the number of rows to be skipped. The offset starts from zero.
`number_rows`	It specifies the number of rows to be returned. It can be a non-negative integer or an expression that outputs a non-negative integer.

Either `offset` or `number_rows` can accept an expression, which value must be a non-negative integer.

> **Note**
>
> Fraction expressions composed of two integers are automatically floored to integers. For example, 8/6 is floored to 1.

EXAMPLES

Return a specific number of rows. To return the top N rows from the result, use `LIMIT <N>` as follows:

```
nebula> MATCH (v:player) RETURN v.name AS Name, v.age AS Age \
        ORDER BY Age LIMIT 5;
+-------------------------+-----+
| Name                    | Age |
+-------------------------+-----+
| "Luka Doncic"           | 20  |
+-------------------------+-----+
| "Ben Simmons"           | 22  |
+-------------------------+-----+
| "Kristaps Porzingis"    | 23  |
+-------------------------+-----+
| "Giannis Antetokounmpo" | 24  |
+-------------------------+-----+
| "Kyle Anderson"         | 25  |
+-------------------------+-----+
nebula> MATCH (v:player) RETURN v.name AS Name, v.age AS Age \
        ORDER BY Age LIMIT rand32(5);
+-------------------------+-----+
| Name                    | Age |
+-------------------------+-----+
| "Luka Doncic"           | 20  |
+-------------------------+-----+
| "Ben Simmons"           | 22  |
+-------------------------+-----+
| "Kristaps Porzingis"    | 23  |
+-------------------------+-----+
| "Giannis Antetokounmpo" | 24  |
+-------------------------+-----+
```

SKIP-SYNTAX

You can use `SKIP <N>` to skip the top N rows from the result and return the rest of the result.

```
nebula> MATCH (v:player{name:"Tim Duncan"}) --> (v2) \
        RETURN v2.name AS Name, v2.age AS Age \
        ORDER BY Age DESC SKIP 1;
+-----------------+-----+
| Name            | Age |
+-----------------+-----+
| "Manu Ginobili" | 41  |
+-----------------+-----+
| "Tony Parker"   | 36  |
+-----------------+-----+
nebula> MATCH (v:player{name:"Tim Duncan"}) --> (v2) \
        RETURN v2.name AS Name, v2.age AS Age \
        ORDER BY Age DESC SKIP 1+1;
+---------------+-----+
| Name          | Age |
+---------------+-----+
| "Tony Parker" | 36  |
+---------------+-----+
```

You can use `SKIP` and `LIMIT` together to return the middle N rows.

```
nebula> MATCH (v:player{name:"Tim Duncan"}) --> (v2) \
        RETURN v2.name AS Name, v2.age AS Age \
        ORDER BY Age DESC SKIP 1 LIMIT 1;
+-----------------+-----+
| Name            | Age |
+-----------------+-----+
| "Manu Ginobili" | 41  |
+-----------------+-----+
```

Last update: April 22, 2021

4.7.3 ORDER BY

The `ORDER BY` clause specifies the order of the rows in the output.

- NGQL-extension: You must use a pipe (`|`) and an `ORDER BY` clause after `YIELD` clause.
- OpenCypher style: no pipe is permitted. `ORDER BY` follows a `RETURN` clause.

There are two order options:

- `ASC` : Ascending. `ASC` is the default order.
- `DESC` : Descending.

An order option takes effect only when the expression before it is used for sorting the results.

nGQL-extension Syntax

```
<YIELD clause>
ORDER BY <expression> [ASC | DESC] [, <expression> [ASC | DESC] ...]
```

EXAMPLES

```
nebula> FETCH PROP ON player "player100", "player101", "player102", "player103" YIELD player.age AS age, player.name AS name \
| ORDER BY age ASC, name DESC;
+-------------+-----+---------------------+
| VertexID    | age | name                |
+-------------+-----+---------------------+
| "player103" | 32  | "Rudy Gay"          |
+-------------+-----+---------------------+
| "player102" | 33  | "LaMarcus Aldridge" |
+-------------+-----+---------------------+
| "player101" | 36  | "Tony Parker"       |
+-------------+-----+---------------------+
| "player100" | 42  | "Tim Duncan"        |
+-------------+-----+---------------------+
```

OpenCypher Syntax

```
<RETURN clause>
ORDER BY <expression> [ASC | DESC] [, <expression> [ASC | DESC] ...]
```

An order option takes effect only when the expression before it is used for sorting the results.

EXAMPLES

```
nebula> MATCH (v:player) RETURN v.name AS Name, v.age AS Age  ORDER BY Name DESC;
+-----------------+-----+
| Name            | Age |
+-----------------+-----+
| "Yao Ming"      | 38  |
+-----------------+-----+
| "Vince Carter"  | 42  |
+-----------------+-----+
| "Tracy McGrady" | 39  |
+-----------------+-----+
| "Tony Parker"   | 36  |
+-----------------+-----+
| "Tim Duncan"    | 42  |
+-----------------+-----+
...
```

```
nebula> MATCH (v:player) RETURN v.age AS Age, v.name AS Name  ORDER BY Age DESC, Name ASC
+-----+------------------+
| Age | Name             |
+-----+------------------+
| 47  | "Shaquille O'Neal" |
+-----+------------------+
| 46  | "Grant Hill"     |
+-----+------------------+
| 45  | "Jason Kidd"     |
+-----+------------------+
| 45  | "Steve Nash"     |
+-----+------------------+
...
```

In the preceding example, nGQL sorts the rows by `Age` first. If multiple people are of the same age, nGQL sorts them by `Name`.

Order by NULL values

nGQL lists NULL values at the end of the output for ascending sorting, and at the start for descending sorting.

```
nebula> MATCH (v:player{name:"Tim Duncan"}) --> (v2) \
        RETURN v2.name AS Name, v2.age AS Age  \
        ORDER BY Age;
+-----------------+----------+
| Name            | Age      |
+-----------------+----------+
| "Tony Parker"   | 36       |
+-----------------+----------+
| "Manu Ginobili" | 41       |
+-----------------+----------+
| "Spurs"         | __NULL__ |
+-----------------+----------+
Got 3 rows (time spent 3089/3719 us)
```

```
nebula> MATCH (v:player{name:"Tim Duncan"}) --> (v2) \
        RETURN v2.name AS Name, v2.age AS Age  \
        ORDER BY Age DESC;
+-----------------+----------+
| Name            | Age      |
+-----------------+----------+
| "Spurs"         | __NULL__ |
+-----------------+----------+
| "Manu Ginobili" | 41       |
+-----------------+----------+
| "Tony Parker"   | 36       |
+-----------------+----------+
Got 3 rows (time spent 2851/3360 us)
```

Last update: March 29, 2021

4.7.4 RETURN

`RETURN` defines the output of an nGQL query. To return multiple fields, separate them with commas.

`RETURN` can lead a clause or a statement:

- A `RETURN` clause works in openCypher statements in nGQL, such as `MATCH` or `UNWIND`.
- A `RETURN` statement works independently to output the result of an expression.

OpenCypher compatibility

This topic applies to the openCypher syntax in nGQL only. For nGQL extensions, use `YIELD`.

`RETURN` does not support the following openCypher features yet.

- Return variables with uncommon characters, for example:

```
MATCH (`non-english_characters`:player) \
RETURN `non-english_characters`;
```

- Set a pattern in the `RETURN` clause and return all elements that this pattern matches, for example:

```
MATCH (v:player) \
RETURN (v)-[e]->(v2);
```

NGQL compatibility

- In nGQL 1.0, `RETURN` works with nGQL extensions with the syntax `RETURN <var_ref> IF <var_ref> IS NOT NULL`.
- In nGQL 2.0, `RETURN` does not work with nGQL extensions.

Return vertices

Set a vertex in the `RETURN` clause to return it.

```
nebula> MATCH (v:player) \
        RETURN v;
+------------------------------------------------------------+
| v                                                          |
+------------------------------------------------------------+
| ("player104" :player{age: 32, name: "Marco Belinelli"})    |
+------------------------------------------------------------+
| ("player107" :player{age: 32, name: "Aron Baynes"})        |
+------------------------------------------------------------+
| ("player116" :player{age: 34, name: "LeBron James"})       |
+------------------------------------------------------------+
| ("player120" :player{age: 29, name: "James Harden"})       |
+------------------------------------------------------------+
| ("player125" :player{age: 41, name: "Manu Ginobili"})      |
+------------------------------------------------------------+
...
Got 51 rows (time spent 7322/8244 us)
```

Return edges

Set an edge in the `RETURN` clause to return it.

```
nebula> MATCH (v:player)-[e]->() \
        RETURN e;
+-------------------------------------------------------------------------------+
| e                                                                             |
+-------------------------------------------------------------------------------+
| [:follow "player104"->"player100" @0 {degree: 55}]                            |
+-------------------------------------------------------------------------------+
| [:follow "player104"->"player101" @0 {degree: 50}]                            |
+-------------------------------------------------------------------------------+
| [:follow "player104"->"player105" @0 {degree: 60}]                            |
+-------------------------------------------------------------------------------+
| [:serve "player104"->"team200" @0 {end_year: 2009, start_year: 2007}]         |
+-------------------------------------------------------------------------------+
```

```
| [:serve "player104"->"team208" @0 {end_year: 2016, start_year: 2015}]       |
+-----------------------------------------------------------------------------+
...
Got 233 rows (time spent 14013/16136 us)
```

Return properties

To return a vertex or edge property, use the `{<vertex_name>|<edge_name>}.<property>` syntax.

```
nebula> MATCH (v:player) \
        RETURN v.name, v.age \
        LIMIT 3;
+------------------+-------+
| v.name           | v.age |
+------------------+-------+
| "Rajon Rondo"    | 33    |
+------------------+-------+
| "Rudy Gay"       | 32    |
+------------------+-------+
| "Dejounte Murray" | 29    |
+------------------+-------+
Got 3 rows (time spent 2663/3260 us)
```

Return all elements

To return all the elements matched on a pattern, use an asterisk (*).

```
nebula> MATCH (v:player{name:"Tim Duncan"}) \
        RETURN *;
+-------------------------------------------------+
| v                                               |
+-------------------------------------------------+
| ("player100" :player{age: 42, name: "Tim Duncan"}) |
+-------------------------------------------------+
Got 1 rows (time spent 3332/3954 us)

nebula> MATCH (v:player{name:"Tim Duncan"})-[e]->(v2) \
        RETURN *;
+-------------------------------------------------+-------------------------------------------------------------------------------
+-------------------------------------------------+
| v                                               | e                                                                             |
v2                                                |
+-------------------------------------------------+-------------------------------------------------------------------------------
+-------------------------------------------------+
| ("player100" :player{age: 42, name: "Tim Duncan"}) | [:follow "player100"->"player101" @0 {degree: 95}]                    | ("player101" :player{age: 36, name:
"Tony Parker"})   |
+-------------------------------------------------+-------------------------------------------------------------------------------
+-------------------------------------------------+
| ("player100" :player{age: 42, name: "Tim Duncan"}) | [:follow "player100"->"player125" @0 {degree: 95}]                    | ("player125" :player{age: 41, name:
"Manu Ginobili"}) |
+-------------------------------------------------+-------------------------------------------------------------------------------
+-------------------------------------------------+
| ("player100" :player{age: 42, name: "Tim Duncan"}) | [:serve "player100"->"team204" @0 {end_year: 2016, start_year: 1997}] | ("team204" :team{name:
"Spurs"})                     |
+-------------------------------------------------+-------------------------------------------------------------------------------
+-------------------------------------------------+
Got 3 rows (time spent 3957/4696 us)
```

Rename a field

Use the `AS <alias>` syntax to rename a field in the output.

```
nebula> MATCH (v:player{name:"Tim Duncan"})-[:serve]->(v2) \
        RETURN v2.name AS Team;
+---------+
| Team    |
+---------+
| "Spurs" |
+---------+
Got 1 rows (time spent 2370/3017 us)

nebula> RETURN "Amber" AS Name;
+---------+
| Name    |
+---------+
| "Amber" |
+---------+
Got 1 rows (time spent 380/1097 us)
```

Return a non-existing property

If a property matched does not exist, `NULL` is returned.

```
nebula> MATCH (v:player{name:"Tim Duncan"})-[e]->(v2) \
        RETURN v2.name, type(e), v2.age;
+-----------------+----------+----------+
| v2.name         | type(e)  | v2.age   |
+-----------------+----------+----------+
| "Tony Parker"   | "follow" | 36       |
+-----------------+----------+----------+
| "Manu Ginobili" | "follow" | 41       |
+-----------------+----------+----------+
| "Spurs"         | "serve"  | __NULL__ |
+-----------------+----------+----------+
Got 3 rows (time spent 2976/3658 us)
```

Return expression results

To return the results of expressions such as literals, functions, or predicates, set them in a `RETURN` clause.

```
nebula> MATCH (v:player{name:"Tony Parker"})-->(v2:player) \
        RETURN DISTINCT v2.name, "Hello"+" graphs!", v2.age > 35;
+--------------------+------------------+-------------+
| v2.name            | (Hello+ graphs!) | (v2.age>35) |
+--------------------+------------------+-------------+
| "Tim Duncan"       | "Hello graphs!"  | true        |
+--------------------+------------------+-------------+
| "LaMarcus Aldridge" | "Hello graphs!"  | false       |
+--------------------+------------------+-------------+
| "Manu Ginobili"    | "Hello graphs!"  | true        |
+--------------------+------------------+-------------+
Got 3 rows (time spent 2645/3237 us)

nebula> RETURN 1+1;
+-------+
| (1+1) |
+-------+
| 2     |
+-------+
Got 1 rows (time spent 319/1238 us)

nebula> RETURN 3 > 1;
+-------+
| (3>1) |
+-------+
| true  |
+-------+
Got 1 rows (time spent 205/751 us)

RETURN 1+1, rand32(1, 5);
+-------+-------------+
| (1+1) | rand32(1,5) |
+-------+-------------+
| 2     | 1           |
+-------+-------------+
Got 1 rows (time spent 258/1098 us)
```

Return unique fields

Use `DISTINCT` to remove duplicate fields in the result set.

```
// Before using DISTINCT
nebula> MATCH (v:player{name:"Tony Parker"})--(v2:player) \
        RETURN v2.name, v2.age;
+--------------------+--------+
| v2.name            | v2.age |
+--------------------+--------+
| "Tim Duncan"       | 42     |
+--------------------+--------+
| "LaMarcus Aldridge" | 33     |
+--------------------+--------+
| "Marco Belinelli"  | 32     |
+--------------------+--------+
| "Boris Diaw"       | 36     |
+--------------------+--------+
| "Dejounte Murray"  | 29     |
+--------------------+--------+
| "Tim Duncan"       | 42     |
+--------------------+--------+
| "LaMarcus Aldridge" | 33     |
+--------------------+--------+
| "Manu Ginobili"    | 41     |
+--------------------+--------+
Got 8 rows (time spent 3273/3893 us)
```

```
// After using DISTINCT
MATCH (v:player{name:"Tony Parker"})--(v2:player) RETURN DISTINCT v2.name, v2.age;
+--------------------+--------+
| v2.name            | v2.age |
+--------------------+--------+
| "Tim Duncan"       | 42     |
+--------------------+--------+
| "LaMarcus Aldridge" | 33     |
+--------------------+--------+
| "Marco Belinelli"  | 32     |
+--------------------+--------+
| "Boris Diaw"       | 36     |
+--------------------+--------+
| "Dejounte Murray"  | 29     |
+--------------------+--------+
| "Manu Ginobili"    | 41     |
+--------------------+--------+
Got 6 rows (time spent 3314/3897 us)
```

Last update: March 4, 2021

4.7.5 TTL

TTL indicates time to live. Use the TTL options to specify a timeout for a property. Once timed out, the property expires.

OpenCypher Compatibility

This topic applies to nGQL extensions only.

Precautions

- You CANNOT modify a property with TTL options on it.
- TTL options and indexes CANNOT coexist on a tag or an edge type. Not even if you try to set them on different properties.

Data expiration and deletion

VERTEX PROPERTY EXPIRATION

Vertex property expiration has the following impact.

- If a vertex has only one tag, once a property of the vertex expires, the vertex expires.
- If a vertex has multiple tags, once a property of the vertex expires, properties bound to the same tag with the expired property also expires, but the vertex does not expire and other properties of it remain untouched.

EDGE PROPERTY EXPIRATION

Since an edge can have only one edge type, once an edge property expires, the edge expires.

DATA DELETION

The expired data are still stored on the disk, but queries will filter them out.

Nebula Graph automatically deletes the expired data and reclaims the disk space during the next compaction.

> **Note**
>
> If TTL is disabled, the corresponding data deleted after the last compaction can be queried again.

TTL options

The nGQL TTL feature has the following options.

Option	Description
`ttl_col`	Specifies the property to set a timeout on. The data type of the property must be int or timestamp.
`ttl_duration`	Specifies the timeout adds-on value in seconds. The value must be a non-negative int64 number. A property expires if the sum of its value and the `ttl_duration` value is smaller than the current timestamp. If the `ttl_duration` value is 0, the property never expires.

Use TTL options

You must use the TTL options together to set a valid timeout on a property.

SET A TIMEOUT IF A TAG OR AN EDGE TYPE EXISTS

If a tag or an edge type is already created, to set a timeout on a property bound to the tag or edge type, use `ALTER` to update the tag or edge type.

```
// Create a tag.
nebula> CREATE TAG t1 (a timestamp);
Execution succeeded (time spent 4172/5377 us)

// Use ALTER to update the tag and set the TTL options.
nebula> ALTER TAG t1 ttl_col = "a", ttl_duration = 5;
Execution succeeded (time spent 2975/3700 us)

// Insert a vertex with tag t1. The vertex expires 5 seconds after the insertion.
nebula> INSERT VERTEX t1(a) values "101":(now());
Execution succeeded (time spent 1902/2642 us)
```

SET A TIMEOUT WHEN CREATING A TAG OR AN EDGE TYPE

Use TTL options in the `CREATE` statement to set a timeout when creating a tag or an edge type. For more information, see CREATE TAG or CREATE EDGE.

```
// Create a tag and set the TTL options.
nebula> CREATE TAG t2(a int, b int, c string) ttl_duration= 100, ttl_col = "a";
Execution succeeded (time spent 3173/3753 us)

// Insert a vertex with tag t2.
// The timeout timestamp is 1612778164774 (1612778164674 + 100).
nebula> INSERT VERTEX t2(a, b, c) values "102":(1612778164674, 30, "Hello");
Execution succeeded (time spent 1254/1921 us)
```

Remove a timeout

To disable TTL and remove the timeout on a property, use the following approaches.

- Set `ttl_col` to an empty string.

  ```
  nebula> ALTER TAG t1 ttl_col = "";
  ```

- Drop the property with the timeout.

  ```
  nebula> ALTER TAG t1 DROP (a);
  ```

- Set `ttl_duration` to 0. This operation keeps the TTL options and prevents the property from expiring.

  ```
  nebula> ALTER TAG t1 ttl_duration = 0;
  ```

  > **Caution**
  >
  > Even when `ttl_duration` is 0, you CANNOT alter the property because it still has TTL options.

Last update: April 22, 2021

4.7.6 WHERE

The `WHERE` clause filters the outputs by conditions.

`WHERE` works in the following queries:

- nGQL extensions such as `GO` and `LOOKUP`.
- OpenCypher syntax such as `MATCH` and `WITH`.

OpenCypher compatibility

- Using patterns in `WHERE` is not supported (TODO: planning), for example `WHERE (v)-->(v2)`.
- Filtering on edge rank is a native nGQL feature. It only applies to nGQL extensions such as `GO` and `LOOKUP` because the concept edge rank does not exist in openCypher.

Basic usage

DEFINE CONDITIONS WITH BOOLEAN OPERATORS

Use the boolean operators `NOT`, `AND`, `OR`, and `XOR` to define conditions in `WHERE` clauses. For the precedence of the operators, see Precedence.

```
nebula> MATCH (v:player) \
        WHERE v.name == "Tim Duncan" \
        XOR (v.age < 30 AND v.name == "Yao Ming") \
        OR NOT (v.name == "Yao Ming" OR v.name == "Tim Duncan") \
        RETURN v.name, v.age;
+-------------------------+-------+
| v.name                  | v.age |
+-------------------------+-------+
| "Marco Belinelli"       | 32    |
+-------------------------+-------+
| "Aron Baynes"           | 32    |
+-------------------------+-------+
| "LeBron James"          | 34    |
+-------------------------+-------+
| "James Harden"          | 29    |
+-------------------------+-------+
| "Manu Ginobili"         | 41    |
+-------------------------+-------+
...
Got 50 rows (time spent 6152/6994 us)
```

```
nebula> GO FROM "player100" \
        OVER follow \
        WHERE follow.degree > 90 \
        OR $$.player.age != 33 \
        AND $$.player.name != "Tony Parker";
+-------------+
| follow._dst |
+-------------+
| "player101" |
+-------------+
| "player125" |
+-------------+
Got 2 rows (time spent 3198/3877 us)
```

FILTER ON PROPERTIES

Use vertex or edge properties to define conditions in WHERE clauses.

- Filter on a vertex property:

```
nebula> MATCH (v:player)-[e]->(v2) \
        WHERE v2.age < 25 \
        RETURN v2.name, v2.age;
+----------------------+--------+
| v2.name              | v2.age |
+----------------------+--------+
| "Luka Doncic"        | 20     |
+----------------------+--------+
| "Kristaps Porzingis" | 23     |
+----------------------+--------+
| "Ben Simmons"        | 22     |
+----------------------+--------+
Got 3 rows (time spent 7382/8080 us)
```

```
nebula> GO FROM "player100" \
        OVER follow \
        WHERE $^.player.age >= 42;
+-------------+
| follow._dst |
+-------------+
| "player101" |
+-------------+
| "player125" |
+-------------+
Got 2 rows (time spent 1051/1668 us)
```

- Filter on an edge property:

```
nebula> MATCH (v:player)-[e]->() \
        WHERE e.start_year < 2000 \
        RETURN DISTINCT v.name, v.age;
+--------------------+-------+
| v.name             | v.age |
+--------------------+-------+
| "Shaquille O'Neal" | 47    |
+--------------------+-------+
| "Steve Nash"       | 45    |
+--------------------+-------+
| "Ray Allen"        | 43    |
+--------------------+-------+
| "Grant Hill"       | 46    |
+--------------------+-------+
| "Tony Parker"      | 36    |
+--------------------+-------+
...
Got 11 rows (time spent 7585/8154 us)
```

```
nebula> GO FROM "player100" \
        OVER follow \
        WHERE follow.degree > 90;
+-------------+
| follow._dst |
+-------------+
| "player101" |
+-------------+
| "player125" |
+-------------+
Got 2 rows (time spent 2815/3571 us)
```

FILTER ON DYNAMICALLY-CALCULATED PROPERTY

```
nebula> MATCH (v:player) \
        WHERE v[toLower("AGE")] < 21 \
        RETURN v.name, v.age;
+---------------+-------+
| v.name        | v.age |
+---------------+-------+
| "Luka Doncic" | 20    |
+---------------+-------+
```

FILTER ON THE EXISTENCE OF A PROPERTY

```
nebula> MATCH (v:player) \
        WHERE exists(v.age) \
        RETURN v.name, v.age;
+-------------------------+-------+
| v.name                  | v.age |
+-------------------------+-------+
| "Boris Diaw"            | 36    |
+-------------------------+-------+
```

```
| "DeAndre Jordan"        | 30    |
+-------------------------+-------+
```

FILTER ON EDGE RANK

In nGQL, if a group of edges has the same source vertex, destination vertex, and properties, the only thing that distinguishes them is the rank. Use rank conditions in `WHERE` to filter such edges.

The following example creates a group of edges. The differences among the edges are their ranks and properties. Then the example uses a `GO` statement with a `WHERE` clause to filter the edges on ranks.

```
nebula> CREATE SPACE test;
nebula> USE test;
nebula> CREATE EDGE e1(p1 int);
nebula> CREATE TAG person(p1 int);
nebula> INSERT VERTEX person(p1) VALUES "1":(1);
nebula> INSERT VERTEX person(p1) VALUES "2":(2);
nebula> INSERT EDGE e1(p1) VALUES "1"->"2"@0:(10);
nebula> INSERT EDGE e1(p1) VALUES "1"->"2"@1:(11);
nebula> INSERT EDGE e1(p1) VALUES "1"->"2"@2:(12);
nebula> INSERT EDGE e1(p1) VALUES "1"->"2"@3:(13);
nebula> INSERT EDGE e1(p1) VALUES "1"->"2"@4:(14);
nebula> INSERT EDGE e1(p1) VALUES "1"->"2"@5:(15);
nebula> INSERT EDGE e1(p1) VALUES "1"->"2"@6:(16);

// The return messages of the preceding statements are omitted in this example.

nebula> GO FROM "1" \
        OVER e1 \
        WHERE e1._rank>2 \
        YIELD e1._src, e1._dst, e1._rank AS Rank, e1.p1 | \
        ORDER BY Rank DESC;
==================================
| e1._src | e1._dst | Rank | e1.p1 |
==================================
| 1       | 2       | 6    | 16    |
----------------------------------
| 1       | 2       | 5    | 15    |
----------------------------------
| 1       | 2       | 4    | 14    |
----------------------------------
| 1       | 2       | 3    | 13    |
----------------------------------
```

Filter on strings

Use `STARTS WITH`, `ENDS WITH`, or `CONTAINS` in `WHERE` to match a specific part of a string. String matching is case-sensitive.

MATCH THE BEGINNING OF A STRING

Use `STARTS WITH "T"` to match a player name that starts with `T`.

```
nebula> MATCH (v:player) \
        WHERE v.name STARTS WITH "T" \
        RETURN v.name, v.age;
+------------------+-------+
| v.name           | v.age |
+------------------+-------+
| "Tracy McGrady"  | 39    |
+------------------+-------+
| "Tony Parker"    | 36    |
+------------------+-------+
| "Tim Duncan"     | 42    |
+------------------+-------+
| "Tiago Splitter" | 34    |
+------------------+-------+
Got 4 rows (time spent 5575/7203 us)
```

If you use `STARTS WITH "t"` in the preceding statement, an empty set is returned because no name in the dataset starts with the lowercase `t`.

```
nebula> MATCH (v:player) \
        WHERE v.name STARTS WITH "t" \
        RETURN v.name, v.age;
Empty set (time spent 5080/6474 us)
```

MATCH THE ENDING OF A STRING

Use `ENDS WITH "r"` to match a player name that ends with `r`.

```
nebula> MATCH (v:player) \
        WHERE v.name ENDS WITH "r" \
        RETURN v.name, v.age;
+------------------+-------+
| v.name           | v.age |
+------------------+-------+
| "Vince Carter"   | 42    |
+------------------+-------+
| "Tony Parker"    | 36    |
+------------------+-------+
| "Tiago Splitter" | 34    |
+------------------+-------+
Got 3 rows (time spent 4934/5832 us)
```

MATCH ANY PART OF A STRING

Use `CONTAINS "Pa"` to match a player name that contains `Pa`.

```
nebula> MATCH (v:player) \
        WHERE v.name CONTAINS "Pa" \
        RETURN v.name, v.age;
+---------------+-------+
| v.name        | v.age |
+---------------+-------+
| "Paul George" | 28    |
+---------------+-------+
| "Tony Parker" | 36    |
+---------------+-------+
| "Paul Gasol"  | 38    |
+---------------+-------+
| "Chris Paul"  | 33    |
+---------------+-------+
Got 4 rows (time spent 3265/4113 us)
```

NEGATIVE STRING MATCHING

Use the boolean operator `NOT` to negate a string matching condition.

```
nebula> MATCH (v:player) \
        WHERE NOT v.name ENDS WITH "R" \
        RETURN v.name, v.age;
+-------------------------+-------+
| v.name                  | v.age |
+-------------------------+-------+
| "Rajon Rondo"           | 33    |
+-------------------------+-------+
| "Rudy Gay"              | 32    |
+-------------------------+-------+
| "Dejounte Murray"       | 29    |
+-------------------------+-------+
| "Chris Paul"            | 33    |
+-------------------------+-------+
| "Carmelo Anthony"       | 34    |
+-------------------------+-------+
...
Got 51 rows (time spent 2622/3463 us)
```

Filter on lists

MATCH VALUES IN A LIST

Use the `IN` operator to check if a value is in a specific list.

```
nebula> MATCH (v:player) \
        WHERE v.age IN range(20,25) \
        RETURN v.name, v.age;
+-------------------------+-------+
| v.name                  | v.age |
+-------------------------+-------+
| "Ben Simmons"           | 22    |
+-------------------------+-------+
| "Kristaps Porzingis"    | 23    |
+-------------------------+-------+
| "Luka Doncic"           | 20    |
+-------------------------+-------+
| "Kyle Anderson"         | 25    |
+-------------------------+-------+
| "Giannis Antetokounmpo" | 24    |
+-------------------------+-------+
| "Joel Embiid"           | 25    |
+-------------------------+-------+
Got 6 rows (time spent 5815/7220 us)
```

MATCH VALUES NOT IN A LIST

Use `NOT` before `IN` to rule out the values in a list.

```
nebula> MATCH (v:player) \
        WHERE v.age NOT IN range(20,25) \
        RETURN v.name AS Name, v.age AS Age \
        ORDER BY Age;
+--------------------+-----+
| Name               | Age |
+--------------------+-----+
| "Kyrie Irving"     | 26  |
+--------------------+-----+
| "Cory Joseph"      | 27  |
+--------------------+-----+
| "Damian Lillard"   | 28  |
+--------------------+-----+
| "Paul George"      | 28  |
+--------------------+-----+
| "Ricky Rubio"      | 28  |
+--------------------+-----+
...
Got 45 rows (time spent 2954/3725 us)
```

Last update: April 1, 2021

4.7.7 YIELD

`YIELD` defines the output of an nGQL query.

`YIELD` can lead a clause or a statement:

- A `YIELD` clause works in nGQL statements such as `GO`, `FETCH`, or `LOOKUP`.
- A `YIELD` statement works in a composite query or independently.

OpenCypher Compatibility

This topic applies to nGQL extensions only. For the openCypher syntax, use `RETURN`.

`YIELD` has different functions in openCypher and nGQL.

- In openCypher, `YIELD` is used in the `CALL[…YIELD]` clause to specify the output of the procedure call.

  > **Note**
  >
  > NGQL does not support `CALL[…YIELD]` yet.

- In nGQL, `YIELD` works like `RETURN` in openCypher.

YIELD clauses

SYNTAX

```
YIELD [DISTINCT] <col> [AS <alias>] [, <col> [AS <alias>] ...]
```

The syntax is described as follows.

Keyword/Field	Description
`DISTINCT`	Aggregates the output and makes the statement return a distinct result set.
`col`	A field to be returned. If no alias is set, `col` will be a column name in the output.
`alias`	An alias for `col`. It is set after the keyword `AS` and will be a column name in the output.

USE A YIELD CLAUSE IN A STATEMENT

- Use `YIELD` with `GO`:

```
nebula> GO FROM "player100" OVER follow \
        YIELD $$.player.name AS Friend, $$.player.age AS Age;
+-----------------+-----+
| Friend          | Age |
+-----------------+-----+
| "Tony Parker"   | 36  |
+-----------------+-----+
| "Manu Ginobili" | 41  |
```

```
+-----------------+-----+
Got 2 rows (time spent 3378/4030 us)
```

- Use `YIELD` with `FETCH`:

```
nebula> FETCH PROP ON player "player100" \
        YIELD player.name;
+-------------+--------------+
| VertexID    | player.name  |
+-------------+--------------+
| "player100" | "Tim Duncan" |
+-------------+--------------+
Got 1 rows (time spent 2933/5931 us)
```

- Use `YIELD` with `LOOKUP`:

```
nebula> LOOKUP ON player WHERE player.name == "Tony Parker" \
        YIELD player.name, player.age;
======================================
| VertexID | player.name | player.age |
======================================
| 101      | Tony Parker | 36         |
--------------------------------------
Got 1 rows (time spent 2963/3778 us)
```

YIELD Statements

SYNTAX

```
YIELD [DISTINCT] <col> [AS <alias>] [, <col> [AS <alias>] ...]
[WHERE <conditions>]
```

The syntax is described as follows.

Field	Description
`DISTINCT`	Aggregates the output and makes the statement return a distinct result set.
`col`	A field to be returned. If no alias is set, `col` will be a column name in the output.
`alias`	An alias for `col`. It is set after the keyword `AS` and will be a column name in the output.
`conditions`	Conditions set in a `WHERE` clause to filter the output. For more information, see `WHERE`.

USE A YIELD STATEMENT IN A COMPOSITE QUERY

In a composite query, a `YIELD` statement accepts, filters, and reforms the result set of the preceding statement, and then outputs it.

The following query finds the players that "player100" follows and calculates their average age.

```
nebula> GO FROM "player100" OVER follow \
        YIELD follow._dst AS ID | \
        FETCH PROP ON player $-.ID \
        YIELD player.age AS Age | \
        YIELD AVG($-.Age) as Avg_age, count(*)as Num_friends;
+---------+-------------+
| Avg_age | Num_friends |
+---------+-------------+
| 38.5    | 2           |
+---------+-------------+
Got 1 rows (time spent 1846/2426 us)
```

The following query finds the players that "player101" follows and the follow degrees are greater than 90.

```
nebula> $var1 = GO FROM "player101" OVER follow \
        YIELD follow.degree AS Degree, follow._dst as ID; \
        YIELD $var1.ID AS ID \
        WHERE $var1.Degree > 90;
+-------------+
| ID          |
+-------------+
| "player100" |
+-------------+
| "player125" |
+-------------+
Got 2 rows (time spent 891/1411 us)
```

USE A STANDALONE YIELD STATEMENT

A `YIELD` statement can calculate a valid expression and output the result.

```
nebula> YIELD rand32(1, 6);
+-------------+
| rand32(1,6) |
+-------------+
| 3           |
+-------------+
Got 1 rows (time spent 144/615 us)

nebula> YIELD "Hel" + "\tlo" AS string1, ", World!" AS string2;
+-------------+------------+
| string1     | string2    |
+-------------+------------+
| "Hel    lo" | ", World!" |
+-------------+------------+
Got 1 rows (time spent 154/692 us)

nebula> YIELD hash("Tim") % 100;
+-----------------+
| (hash(Tim)%100) |
+-----------------+
| 42              |
+-----------------+
Got 1 rows (time spent 164/820 us)

nebula> YIELD \
      CASE 2+3 \
      WHEN 4 THEN 0 \
      WHEN 5 THEN 1 \
      ELSE -1 \
      END \
      AS result;
+--------+
| result |
+--------+
| 1      |
+--------+
Got 1 rows (time spent 204/935 us)
```

Last update: April 22, 2021

4.7.8 WITH

OpenCypher compatibility

The `WITH` clause can take the output from a query part, process it, and pass it to the next query part as the input.

`WITH` has a similar function with the pipe symbol in nGQL-extension, but they work in different ways.

`WITH` only works in the openCypher syntax, such as in `MATCH` or `UNWIND`.

In the nGQL-extensions such as `GO` or `FETCH`, use pipe symbols (`|`) instead.

> **Danger**
>
> Don't use pipe symbols in the openCypher syntax or use `WITH` in the nGQL extensions. Such operations may cause unpredictable results.

Combine statements and form a composite query

Use a `WITH` clause to combine statements and transfer the output of a statement as the input of another statement.

EXAMPLE 1

The following statement:

1. Matches a path.
2. Outputs all the vertices on the path to a list with the `nodes()` function.
3. Unwinds the list into rows.
4. Removes duplicated vertices and returns a set of distinct vertices.

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})--() \
        WITH nodes(p) AS n \
        UNWIND n AS n1 \
        RETURN DISTINCT n1;
+-----------------------------------------------------------------+
| n1                                                              |
+-----------------------------------------------------------------+
| ("player100" :star{} :person{} :player{age: 42, name: "Tim Duncan"}) |
+-----------------------------------------------------------------+
| ("player101" :player{age: 36, name: "Tony Parker"})             |
+-----------------------------------------------------------------+
| ("team204" :team{name: "Spurs"})                                |
+-----------------------------------------------------------------+
| ("player102" :player{age: 33, name: "LaMarcus Aldridge"})       |
+-----------------------------------------------------------------+
| ("player125" :player{age: 41, name: "Manu Ginobili"})           |
+-----------------------------------------------------------------+
| ("player104" :player{age: 32, name: "Marco Belinelli"})         |
+-----------------------------------------------------------------+
| ("player144" :player{age: 47, name: "Shaquile O'Neal"})         |
+-----------------------------------------------------------------+
| ("player105" :player{age: 31, name: "Danny Green"})             |
+-----------------------------------------------------------------+
| ("player113" :player{age: 29, name: "Dejounte Murray"})         |
+-----------------------------------------------------------------+
| ("player107" :player{age: 32, name: "Aron Baynes"})             |
+-----------------------------------------------------------------+
| ("player109" :player{age: 34, name: "Tiago Splitter"})          |
+-----------------------------------------------------------------+
| ("player108" :player{age: 36, name: "Boris Diaw"})              |
+-----------------------------------------------------------------+
Got 12 rows (time spent 3795/4487 us)
```

EXAMPLE 2

The following statement:

1. Matches a vertex with the VID "player100".
2. Outputs all the tags of the vertex into a list with the `labels()` function.
3. Unwinds the list into rows.
4. Returns the rows.

```
nebula> MATCH (v) \
        WHERE id(v)=="player100" \
        WITH labels(v) AS tags_unf \
        UNWIND tags_unf AS tags_f \
        RETURN tags_f;
+----------+
| tags_f   |
+----------+
| "star"   |
+----------+
| "player" |
+----------+
| "person" |
+----------+
Got 3 rows (time spent 1709/2495 us)
```

Filter aggregated queries

`WITH` can work as a filter in the middle of an aggregated query.

```
nebula> MATCH (v:player)-->(v2:player) \
        WITH DISTINCT v2 AS v2, v2.age AS Age \
        ORDER BY Age \
        WHERE Age<25 \
        RETURN v2.name AS Name, Age;
+----------------------+-----+
| Name                 | Age |
+----------------------+-----+
| "Luka Doncic"        | 20  |
+----------------------+-----+
| "Ben Simmons"        | 22  |
+----------------------+-----+
| "Kristaps Porzingis" | 23  |
+----------------------+-----+
Got 3 rows (time spent 7444/8467 us)
```

Process the output before using collect() on it

Use a `WITH` clause to sort and limit the output before using `collect()` to transform the output into a list.

```
nebula> MATCH (v:player) \
        WITH v.name AS Name \
        ORDER BY Name DESC \
        LIMIT 3 \
        RETURN collect(Name);
+------------------------------------------------+
| COLLECT(Name)                                  |
+------------------------------------------------+
| ["Yao Ming", "Vince Carter", "Tracy McGrady"] |
+------------------------------------------------+
Got 1 rows (time spent 3498/4222 us)
```

Use with RETURN

Set a alias using a `WITH` clause, and then output the result through a `RETURN` clause.

```
nebula> WITH [1, 2, 3] AS list  RETURN 3 IN list AS r;
+------+
| r    |
+------+
| true |
+------+

nebula> WITH 4 AS one, 3 AS two RETURN one > two AS result;
+--------+
| result |
+--------+
```

```
| true   |
+--------+
```

Last update: May 7, 2021

4.8 Space statements

4.8.1 CREATE SPACE

```
CREATE SPACE [IF NOT EXISTS] <graph_space_name>
    [(partition_num = <partition_number>,
    replica_factor = <replica_number>,
    vid_type = {FIXED_STRING(<N>) | INT64})];
```

The `CREATE SPACE` statement creates a new graph space with the given name. A `SPACE` is a region that provides physically isolated graphs in Nebula Graph. An error occurs if a graph space with the same name exists if you did not specify `IF NOT EXISTS`.

IF NOT EXISTS

You can use the `IF NOT EXISTS` keywords when creating graph spaces. These keywords automatically detect if the related graph space exists. If it does not exist, a new one is created. Otherwise, no graph space is created.

> **Note**
>
> The graph space existence detection here only compares the graph space name (excluding properties).

Graph space name

The `graph_space_name` uniquely identifies a graph space in a Nebula Graph instance.

Customized graph space options

You can set four optional options for a new graph space:

- `partition_num`

  Specifies the number of partitions in each replica. The suggested number is five times the number of the hard disks in the cluster. For example, if you have 3 hard disks in the cluster, we recommend that you set 15 partitions.

- `replica_factor`

  Specifies the number of replicas in the cluster. The default replica factor is 1. The suggested number is 3 in a production environment and 1 in a test environment. Always set the replica to an odd number for the need of quorum-based voting.

NOTICE: If the replica number is set to one, you won't be able to use the `BALANCE` statements to load balance or scale out the Nebula Graph Storage Service.

- `vid_type`

  Specifies the data type of VIDs in a graph space. Available values are `FIXED_STRING(N)` and `INT64`, where `N` represents the maximum length of the VIDs and it must be a positive integer. The default value is `FIXED_STRING(8)`.

  If you set a VID length greater than `N`, Nebula Graph throws an error. To set the integer VID for vertices, set `vid_type` to `INT64`.

If no option is given, Nebula Graph creates the graph space with the default options.

Example

```
nebula> CREATE SPACE my_space_1; -- create a graph space with default options
nebula> CREATE SPACE my_space_2(partition_num=10); -- create a graph space with customized partition number
nebula> CREATE SPACE my_space_3(replica_factor=1); -- create a graph space with customized replica factor
nebula> CREATE SPACE my_space_4(vid_type = FIXED_STRING(30)); -- create a graph space with customized VID maximum length
```

Implementation of the operation

Trying to use a newly created graph space may fail because the creation is implemented asynchronously.

Nebula Graph implements the creation in the next heartbeat cycle. To make sure the creation is successful, take one of the following approaches:

- Find the new graph space in the result of `SHOW SPACES` or `DESCRIBE SPACE`. If you can't, wait a few seconds and try again.
- Wait for two heartbeat cycles, i.e., 20 seconds.

To change the heartbeat interval, modify the `heartbeat_interval_secs` parameter in the configuration files for all services.

Check partition distribution

On some large clusters, the partition distribution is possibly unbalanced because of the different startup times. You can run the command to do a check of the machine distribution.

```
nebula> SHOW HOSTS;
+-----------+-------+--------+--------------+---------------------------------+----------------------------+
| Host      | Port  | Status | Leader count | Leader distribution             | Partition distribution     |
+-----------+-------+--------+--------------+---------------------------------+----------------------------+
| storaged0 | 9779  | ONLINE | 1            | basketballplayer:5              | basketballplayer:5         |
+-----------+-------+--------+--------------+---------------------------------+----------------------------+
| storaged1 | 9779  | ONLINE | 2            | test:1, basketballplayer:5      | basketballplayer:5, test:1 |
+-----------+-------+--------+--------------+---------------------------------+----------------------------+
| storaged2 | 9779  | ONLINE | 1            | basketballplayer:5              | basketballplayer:5         |
+-----------+-------+--------+--------------+---------------------------------+----------------------------+
```

To balance the request loads, use the following command.

```
nebula> BALANCE LEADER;
```

Last update: April 22, 2021

4.8.2 USE

```
USE <graph_space_name>
```

The `USE` statement specifies a graph space as the current working space for subsequent queries. To manage multiple graph spaces, use the `USE` statement. The `USE` statement requires some privilege.

The graph space remains the same unless another `USE` statement is executed.

```
nebula> USE space1;
-- Traverse in graph space1.
nebula> GO FROM 1 OVER edge1;
nebula> USE space2;
-- Traverse in graph space2. These vertices and edges have no relevance with space1.
nebula> GO FROM 2 OVER edge2;
-- Now you are back to space1. Hereafter, you can not read any data from space2.
nebula> USE space1;
```

> **Note**
>
> You can't use two spaces in one statement.

Different from SQL or Fabric Cypher, making a graph space as the working graph space prevents you from accessing other spaces. The only way to traverse in a new graph space is to switch by the `USE` statement.

Graph spaces are `FULLY ISOLATED` from each other. Unlike Fabric Cypher, you can only use one graph space at a time in Nebula Graph. But in Fabric Cypher, you can use two (graph) spaces in one statement.

Last update: April 22, 2021

4.8.3 SHOW SPACES

```
SHOW SPACES
```

The `SHOW SPACES` statement lists the all the graph spaces in a Nebula Graph instance.

For example:

```
nebula> SHOW SPACES;
+--------------------+
| Name               |
+--------------------+
| "basketballplayer" |
+--------------------+
```

To create graph spaces, see Create Space document.

Last update: April 13, 2021

4.8.4 DESCRIBE SPACE

```
DESC[RIBE] SPACE <graph_space_name>
```

The `DESCRIBE SPACE` statement returns information about a graph space.

The `DESCRIBE SPACE` statement is different from the `SHOW SPACES` statement. For details about `SHOW SPACES`, see SHOW SPACES.

You can use `DESC` instead of `DESCRIBE` for short.

Example

Get information about a graph space.

```
nebula> DESCRIBE SPACE basketballplayer;
+----+--------------------+------------------+---------------+---------+-----------+--------------------+-------------+-----------+
| ID | Name               | Partition Number | Replica Factor | Charset | Collate    | Vid Type           | Atomic Edge | Group     |
+----+--------------------+------------------+---------------+---------+-----------+--------------------+-------------+-----------+
| 1  | "basketballplayer" | 10               | 1             | "utf8"  | "utf8_bin" | "FIXED_STRING(32)" | "false"     | "default" |
+----+--------------------+------------------+---------------+---------+-----------+--------------------+-------------+-----------+
```

Last update: April 13, 2021

4.8.5 DROP SPACE

The `DROP SPACE` statement deletes everything in the specified graph space.

> **Danger**
>
> Before dropping a graph space, make sure that you have backed up all the important data stored in it. Otherwise, once the graph space is dropped, the data cannot be restored.

```
DROP SPACE [IF EXISTS] <graph_space_name>
```

You must have the `DROP` privilege for the related graph space.

You can use the `IF EXISTS` keywords when dropping spaces. These keywords automatically detects if the related graph space exists. If it exists, it is deleted. Otherwise, no graph space is deleted.

Other graph spaces stay unchanged.

The `DROP SPACE` statement does not immediately remove all the files and directories from disk. `Use` another `space`, and `submit job compact`.

Last update: April 22, 2021

4.9 Tag statements

4.9.1 CREATE TAG

`CREATE TAG` creates a tag with the given name in a graph space. You must have the `CREATE` privilege for the graph space. To create a tag in a specific graph space, you must use the graph space first.

OpenCypher compatibility

Tags in nGQL are similar with labels in openCypher. But they are also quite different. For example, the ways to create them are different.

- In openCypher, labels are created together with nodes by `CREATE` statements.
- In nGQL, tags are created separately by `CREATE TAG` statements. Tags in nGQL are more like tables in MySQL.

Syntax

```
CREATE TAG [IF NOT EXISTS] <tag_name>
    ([<create_definition>, ...])
    [tag_options]

<create_definition> ::=
    <prop_name> <data_type> [NULL | NOT NULL]

<tag_options> ::=
    <option> [, <option> ...]

<option> ::=
    TTL_DURATION [=] <ttl_duration>
    | TTL_COL [=] <prop_name>
    | DEFAULT <default_value>
```

Tag name

- `IF NOT EXISTS`: Creating an existent tag results in an error. You can use the `IF NOT EXISTS` option to conditionally create the tag and avoid the error.

  > **Note**
  >
  > The tag existence detection here compares only the tag names (excluding properties).

- `tag_name`: The tag name must be **unique** in a graph space. Once the tag name is set, it can not be altered. The rules for permitted tag names are the same as those for graph space names. For prohibited names, see Keywords and reserved words.

PROPERTY NAMES AND DATA TYPES

- `prop_name`

  `prop_name` is the name of the property. It must be unique for each tag.

- `data_type`

  `data_type` shows the data type of each property. For a full description of the property data types, see Data types.

- `NULL | NOT NULL`

  Specifies if the property supports `NULL | NOT NULL`. The default value is `NULL`.

- `DEFAULT`

  Specifies a default value for a property. The default value can be a literal value or an expression supported by Nebula Graph. If no value is specified, the default value is used when inserting a new vertex.

TIME-TO-LIVE (TTL)

- `TTL_DURATION`

  Specifies the life cycle for the data. Data that exceeds the specified TTL expires. The expiration threshold is the `TTL_COL` value plus the `TTL_DURATION`. The default value of `TTL_DURATION` is zero. It means the data never expires.

- `TTL_COL`

  The data type of `prop_name` must be either `int` or `timestamp`.

- single TTL definition

  Only one `TTL_COL` field can be specified in a tag.

For more information on TTL, see TTL options.

EXAMPLES

```
nebula> CREATE TAG player(name string, age int);

// Create a tag with no properties.
nebula> CREATE TAG no_property();

// Create a tag with a default value.
nebula> CREATE TAG player_with_default(name string, age int DEFAULT 20);
```

```
// Time interval is 100s, starting from the create_time field
nebula> CREATE TAG woman(name string, age int, \
   married bool, salary double, create_time timestamp) \
   TTL_DURATION = 100, TTL_COL = "create_time";

// Data expires after TTL_DURATION
nebula> CREATE TAG icec_ream(made timestamp, temperature int) \
   TTL_DURATION = 100, TTL_COL = "made";
```

Implementation of the operation

Trying to insert vertices with a newly created tag may fail, because the creation of the tag is implemented asynchronously.

Nebula Graph implements the creation in the next heartbeat cycle. To make sure the creation is successful, take one of the following approaches:

- Find the new tag in the result of `SHOW TAGS`. If you can't, wait a few seconds and try again.
- Wait for two heartbeat cycles, i.e., 20 seconds.

To change the heartbeat interval, modify the `heartbeat_interval_secs` parameter in the configuration files for all services.

Last update: April 22, 2021

4.9.2 DROP TAG

```
DROP TAG [IF EXISTS] <tag_name>
```

`DROP TAG` drops a tag with the given name in a graph space. You must have the `DROP` privilege for the graph space. To drop a tag in a specific graph space, you must use the graph space first.

> **Note**
>
> Before you drop a tag, make sure that the tag does not have any indexes. Otherwise, a conflict error (`[ERROR (-8)]: Conflict!`) is returned. To remove an index, see DROP INDEX.

A vertex can have one or more tags.

- When a vertex has only one tag, after you drop it, the vertex **CANNOT** be accessible. But its edges are available. The vertex is deleted in the next compaction.
- When a vertex has multiple tags, after you drop one of them, the vertex is still accessible. But all the properties defined by this dropped tag are not accessible.

This operation only deletes the Schema data. All the files and directories in the disk are NOT deleted directly. Data is deleted in the next compaction.

Tag name

- `IF EXISTS` : Dropping a non-existent tag results in an error. You can use the `IF EXISTS` option to conditionally drop the tag and avoid the error.

  > **Note**
  >
  > The tag existence detection here compares only the tag names (excluding properties).

- `tag_name` : Specifies the tag name that you want to drop. You can drop only one tag in one statement.

Example

```
nebula> CREATE TAG test(p1 string, p2 int);

nebula> DROP TAG test;
```

Last update: April 22, 2021

4.9.3 ALTER TAG

```
ALTER TAG <tag_name>
    <alter_definition> [, alter_definition] ...]
    [ttl_definition [, ttl_definition] ... ]

alter_definition:
| ADD    (prop_name data_type)
| DROP   (prop_name)
| CHANGE (prop_name data_type)

ttl_definition:
    TTL_DURATION = ttl_duration, TTL_COL = prop_name
```

`ALTER TAG` alters the structure of a tag with the given name in a graph space. You must have the `ALTER` privilege for the graph space. To alter a tag in a specific graph space, you must use the graph space first.

You can add or drop properties, change the data type of an existing property. You can also set TTL (Time-To-Live) for a property, or change the TTL duration. `TTL_COL` only supports the properties whose values are of the `INT` or the `TIMESTAMP` type.

Before you alter properties for a tag, make sure that the properties are not indexed. If the properties contain any indexes, a conflict error occurs when you alter them.

For information about index, see Index.

Multiple `ADD`, `DROP`, and `CHANGE` clauses are permitted in a single `ALTER` statement, separated by commas.

Tag name

- `tag_name`: Specifies the tag name that you want to alter. You can alter only one tag in one statement. Before you alter a tag, make sure that the tag exists in the graph space. If the tag does not exist, an error occurs when you alter it.

Example

```
nebula> CREATE TAG t1 (p1 string, p2 int);
nebula> ALTER TAG t1 ADD (p3 int, p4 string);
nebula> ALTER TAG t1 TTL_DURATION = 2, TTL_COL = "p2";
```

Implementation of the operation

Nebula Graph implements the alteration asynchronously in the next heartbeat cycle. Before the process finishes, the alteration does not take effect. To make sure the alteration is successful, take the following approaches:

- Use `DESCRIBE TAG` to confirm that the tag information is updated. If it is not, wait a few seconds and try again.
- Wait for two heartbeat cycles, i.e., 20 seconds.

To change the heartbeat interval, modify the `heartbeat_interval_secs` parameter in the configuration files for all services.

Last update: March 19, 2021

4.9.4 SHOW TAGS

```
SHOW TAGS
```

`SHOW TAGS` shows all tags in the current graph space. You do not need any privileges for the graph space to run this statement. But the returned results are different based on role privileges. To show tags in a specific graph space, you must use the graph space first.

Examples

```
nebula> SHOW TAGS;
+----------+
| Name     |
+----------+
| "player" |
+----------+
| "team"   |
+----------+
Got 2 rows (time spent 1461/2114 us)
```

Last update: March 19, 2021

4.9.5 DESCRIBE TAG

```
DESC[RIBE] TAG <tag_name>
```

`DESCRIBE TAG` returns information about a tag with the given name in a graph space. You must have the read Schema privilege for the graph space. To describe a tag in a specific graph space, you must use the graph space first. You can use `DESC` instead of `DESCRIBE` for short.

`DESCRIBE TAG` is different from `SHOW TAGS`. For details about `SHOW TAGS`, see SHOW TAGS.

Example

Get information about a tag named `player`.

```
nebula> DESCRIBE TAG player;

+--------+----------+-------+-----------+
| Field  | Type     | Null  | Default   |
+--------+----------+-------+-----------+
| "name" | "string" | "YES" | __EMPTY__ |
+--------+----------+-------+-----------+
| "age"  | "int64"  | "YES" | __EMPTY__ |
+--------+----------+-------+-----------+
```

Last update: March 19, 2021

4.10 Edge type statements

4.10.1 CREATE EDGE

`CREATE EDGE` creates an edge type with the given name in a graph space. You must have the `CREATE` privilege for the graph space. To create an edge type in a specific graph space, you must use the graph space first.

OpenCypher compatibility

Edge types in nGQL are similar to labels/relationship types in openCypher. But they are also quite different. For example, the ways to create them are different.

- In openCypher, relationship types are created together with relationships by `CREATE` statements.
- In nGQL, edge types are created separately by `CREATE EDGE` statements. Edge types in nGQL are more like tables in MySQL.

Syntax

```
CREATE EDGE [IF NOT EXISTS] <edge_type_name>
    ([<create_definition>, ...])
    [edge_type_options]

<create_definition> ::=
    <prop_name> <data_type>

<edge_type_options> ::=
    <option> [, <option> ...]

<option> ::=
    TTL_DURATION [=] <ttl_duration>
    | TTL_COL [=] <prop_name>
    | DEFAULT <default_value>
```

Edge type name

- `IF NOT EXISTS`: Creating an existent edge type causes an error. You can use the `IF NOT EXISTS` option to conditionally create the edge type and avoid the error.

  > **Note**
  >
  > The edge type existence detection here compares only the edge type names (excluding properties).

- `edge_type_name`: The edge type name must be **unique** in a graph space. Once the edge type name is set, it can not be altered. The rules for permitted edge type names are the same as those for graph space names. For prohibited names, see Keywords and reserved words.

PROPERTY NAMES AND DATA TYPES

- `prop_name`

  `prop_name` is the name of the property. It must be unique for each edge type.

- `data_type`

  `data_type` shows the data type of each property. For a full description of the property data types, see Data types.

- `NULL | NOT NULL`

  Specifies if the property supports `NULL | NOT NULL`. The default value is `NULL`.

- `DEFAULT`

  Specifies a default value for a property. The default value can be a literal value or an expression supported by Nebula Graph. If no value is specified, the default value is used when inserting a new vertex.

TIME-TO-LIVE (TTL)

- `TTL_DURATION`

  Specifies the life cycle for the data. Data that exceeds the specified TTL expires. The expiration threshold is the `TTL_COL` value plus the `TTL_DURATION`. The default value of `TTL_DURATION` is `0`. It means the data never expires.

- `TTL_COL`

  The data type of `prop_name` must be either `int` or `timestamp`.

- single TTL definition

  Only one `TTL_COL` field can be specified in an edge type.

For more information about TTL, see TTL options.

EXAMPLES

```
nebula> CREATE EDGE follow(degree int);

// Create an edge type with no properties.
nebula> CREATE EDGE no_property();

// Create an edge type with a default value.
nebula> CREATE EDGE follow_with_default(degree int DEFAULT 20);
```

```
// Time interval is 100s, starting from the p2 field
// Data expires after TTL_DURATION
nebula> CREATE EDGE e1(p1 string, p2 int, \
   p3 timestamp) \
   TTL_DURATION = 100, TTL_COL = "p2";
```

Implementation of the operation

Trying to insert edges of a newly created edge type may fail, because the creation of the edge type is implemented asynchronously.

Nebula Graph implements the creation in the next heartbeat cycle. To make sure the creation is successful, take the following approaches:

- Find the new edge type in the result of `SHOW EDGES`. If you can't, wait a few seconds and try again.
- Wait for two heartbeat cycles, i.e., 20 seconds.

To change the heartbeat interval, modify the `heartbeat_interval_secs` parameter in the configuration files for all services.

Last update: April 22, 2021

4.10.2 DROP EDGE

```
DROP EDGE [IF EXISTS] <edge_type_name>
```

`DROP EDGE` drops an edge type with the given name in a graph space. You must have the `DROP` privilege for the graph space. To drop an edge type in a specific graph space, you must use the graph space first.

> **Note**
>
> Before you drop an edge type, make sure that the edge type does not have any indexes. Otherwise, a conflict error (`[ERROR (-8)]: Conflict!`) is returned. To remove an index, see DROP INDEX.

An edge can have only one edge type. After you drop it, the edge **CANNOT** be accessible. The edge is deleted in the next compaction.

Edge type name

- `IF EXISTS` : Dropping a non-existent edge type causes an error. You can use the `IF EXISTS` option to conditionally drop the edge type and avoid the error.

  > **Note**
  >
  > The edge type existence detection here compares only the edge type names (excluding properties).

- `edge_type_name` : Specifies the edge type name that you want to drop. You can drop only one edge type in one statement.

Example

```
nebula> CREATE EDGE e1(p1 string, p2 int);

nebula> DROP EDGE e1;
```

Last update: April 22, 2021

4.10.3 ALTER EDGE

```
ALTER EDGE <edge_type_name>
    <alter_definition> [, alter_definition] ...]
    [ttl_definition [, ttl_definition] ... ]

alter_definition:
| ADD    (prop_name data_type)
| DROP   (prop_name)
| CHANGE (prop_name data_type)

ttl_definition:
    TTL_DURATION = ttl_duration, TTL_COL = prop_name
```

`ALTER EDGE` alters the structure of an edge type with the given name in a graph space. You must have the `ALTER` privilege for the graph space. To alter an edge type in a specific graph space, you must use the graph space first.

You can add or drop properties, change the data type of an existing property. You can also set TTL (Time-To-Live) for a property, or change the TTL duration. `TTL_COL` only supports `INT` or `TIMESTAMP` type properties.

Before you alter properties for an edge type, make sure that the properties are not indexed. If the properties contain any indexes, a conflict error occurs when you alter them.

For information about index, see Index.

Multiple `ADD`, `DROP`, and `CHANGE` clauses are permitted in a single `ALTER` statement, separated by commas.

Edge type name

`edge_type_name` specifies the edge type name that you want to alter. You can alter only one edge type in one statement. Before you alter an edge type, make sure that the edge type exists in the graph space. If the edge type does not exist, an error occurs when you alter it.

Example

```
nebula> CREATE EDGE e1(p1 string, p2 int);
nebula> ALTER EDGE e1 ADD (p3 int, p4 string);
nebula> ALTER EDGE e1 TTL_DURATION = 2, TTL_COL = "p2";
```

Implementation of the operation

Nebula Graph implements the alteration asynchronously in the next heartbeat cycle. Before the process finishes, the alteration does not take effect. To make sure the alteration is successful, take the following approaches:

- Use `DESCRIBE EDGE` to confirm that the edge information is updated. If it is not, wait a few seconds and try again.
- Wait for two heartbeat cycles, i.e., 20 seconds.

To change the heartbeat interval, modify the `heartbeat_interval_secs` parameter in the configuration files for all services.

Last update: March 19, 2021

4.10.4 SHOW EDGES

```
SHOW EDGES
```

`SHOW EDGES` shows all edge types in the current graph space. You do not need any privileges for the graph space to run this statement. But the returned results are different based on role privileges. To show edge types in a specific graph space, you must use the graph space first.

Examples

```
nebula> SHOW EDGES;
+----------+
| Name     |
+----------+
| "follow" |
+----------+
| "serve"  |
+----------+
Got 2 rows (time spent 1039/1687 us)
```

Last update: March 19, 2021

4.10.5 DESCRIBE EDGE

```
DESC[RIBE] EDGE <edge_type_name>
```

`DESCRIBE EDGE` returns information about an edge type with the given name in a graph space. You must have the read Schema privilege for the graph space. To describe an edge type in a specific graph space, you must use the graph space first. You can use `DESC` instead of `DESCRIBE` for short.

`DESCRIBE EDGE` is different from `SHOW EDGE`. For details about `SHOW EDGE`, see SHOW EDGE.

Example

Get information about an edge type named `follow`.

```
nebula> DESCRIBE EDGE follow;

+----------+---------+-------+-----------+
| Field    | Type    | Null  | Default   |
+----------+---------+-------+-----------+
| "degree" | "int64" | "YES" | __EMPTY__ |
+----------+---------+-------+-----------+
```

Last update: March 19, 2021

4.11 Vertex statements

4.11.1 INSERT VERTEX

The `INSERT VERTEX` statement inserts one or more vertices into a graph space in Nebula Graph.

When inserting a vertex with a VID that already exists, `INSERT VERTEX` **overrides** the vertex.

Syntax

```
INSERT VERTEX <tag_name> (<prop_name_list>) [, <tag_name> (<prop_name_list>), ...]
     {VALUES | VALUE} VID: (<prop_value_list>[, <prop_value_list>])

prop_name_list:
  [prop_name [, prop_name] ...]

prop_value_list:
  [prop_value [, prop_value] ...]
```

- `tag_name` denotes the tag (vertex type), which must be created before `INSERT VERTEX`.
- `prop_name_list` contains the names of the properties on the tag.
- `VID` is the vertex ID. In Nebula Graph 2.X, string and integer VID types are supported. The VID type is set when a graph space is created. For detail information on the maximum VID length, see CREATE SPACE.
- `prop_value_list` must provide the property values according to the `prop_name_list`. If the property values do not match the data type in the tag, an error is returned. When the `NOT NULL` constraint is set for a given property, an error is returned if no property is given. When the default value for a property is `NULL`, you can omit to specify the property value. For details, see CREATE TAG.

Examples

```
nebula> CREATE TAG t1();                   -- Create tag t1 with no property
nebula> INSERT VERTEX t1() VALUE "10":();    -- Insert vertex "10" with no property
```

```
nebula> CREATE TAG t2 (name string, age int);                -- Create tag t2 with two properties
nebula> INSERT VERTEX t2 (name, age) VALUES "11":("n1", 12);     -- Insert vertex "11" with two properties
nebula> INSERT VERTEX t2 (name, age) VALUES "12":("n1", "a13");  -- Failed. "a13" is not int
nebula> INSERT VERTEX t2 (name, age) VALUES "13":("n3", 12), "14":("n4", 8);    -- Insert two vertices
```

```
nebula> CREATE TAG t3(p1 int);
nebula> CREATE TAG t4(p2 string);
nebula> INSERT VERTEX  t3 (p1), t4(p2) VALUES "21": (321, "hello");   -- Insert vertex "21" with two tags.
```

A vertex can be inserted/written multiple times. Only the last written values can be read.

```
// Insert vertex "11" with the new values.
nebula> INSERT VERTEX t2 (name, age) VALUES "11":("n2", 13);
nebula> INSERT VERTEX t2 (name, age) VALUES "11":("n3", 14);
nebula> INSERT VERTEX t2 (name, age) VALUES "11":("n4", 15);

// Only the last version can be read
nebula> FETCH PROP ON t2 "11";
+---------------------------------+
| vertices_                       |
+---------------------------------+
| ("11" :t2{age: 15, name: "n4"}) |
+---------------------------------+
```

```
nebula> CREATE TAG t5(p1 fixed_string(5) NOT NULL, p2 int, p3 int DEFAULT NULL);
nebula> INSERT VERTEX t5(p1, p2, p3) VALUES "001":("Abe", 2, 3);
nebula> INSERT VERTEX t5(p1, p2, p3) VALUES "002":(NULL, 4, 5);
[ERROR (-8)]: Storage Error: The not null field cannot be null.
nebula> INSERT VERTEX t5(p1, p2) VALUES "003":("cd", 5);

// The value for p3 is the default NULL.
nebula> FETCH PROP ON t5 "003";
+-----------------------------------------+
| vertices_                               |
+-----------------------------------------+
| ("003" :t5{p1: "cd", p2: 5, p3: __NULL__}) |
```

```
+-----------------------------------------+
nebula> INSERT VERTEX t5(p1, p2) VALUES "004":("shalalalala", 4);

// The allowed maximum length for property p1 is 5.
nebula> FETCH PROP on t5 "004";
+-----------------------------------------------+
| vertices_                                     |
+-----------------------------------------------+
| ("004" :t5{p1: "shala", p2: 4, p3: __NULL__}) |
+-----------------------------------------------+
```

Last update: April 13, 2021

4.11.2 DELETE VERTEX

```
DELETE VERTEX <vid> [, <vid> ...]
```

Use `DELETE VERTEX` to delete vertices and the related incoming and outgoing edges of the vertices. The `DELETE VERTEX` statement deletes one vertex or multiple vertices at a time. You can use `DELETE VERTEX` together with pipe. For more information about pipe, see Pipe operator.

Examples

```
nebula> DELETE VERTEX "team1";
```

This query deletes the vertex whose ID is "team1".

```
nebula> GO FROM "player100" OVER serve YIELD serve._dst AS id | DELETE VERTEX $-.id;
```

This query shows that you can use `DELETE VERTEX` together with pipe.

Nebula Graph traverses the incoming and outgoing edges related to the vertices and deletes them all. Then Nebula Graph deletes information related to the vertices.

> **Note**
>
> Atomic operation is not guaranteed during the entire process for now, so please retry when a failure occurs.

Last update: April 22, 2021

4.11.3 UPDATE VERTEX

The `UPDATE VERTEX` statement updates properties on a vertex. `UPDATE VERTEX` supports compare-and-set (CAS).

> **Note**
>
> An `UPDATE VERTEX` statement can only update properties on **ONE TAG** of a vertex.

Syntax

```
UPDATE VERTEX ON <tag_name> <vid>
SET <update_prop>
[WHEN <condition>]
[YIELD <output>]
```

Field	Required	Description	Example
`ON <tag_name>`	Yes	Specifies the tag of the vertex. The properties to be updated must be on this tag.	`ON player`
`<vid>`	Yes	Specifies the ID of the vertex to be updated.	`"player100"`
`SET <update_prop>`	Yes	Specifies the properties to be updated and how they will be updated.	`SET age = age +1`
`WHEN <condition>`	No	Specifies the filter conditions. If `<condition>` evaluates to `false`, the `SET` clause does not take effect.	`WHEN name == "Tim"`
`YIELD <output>`	No	Specifies the output format of the statement.	`YIELD name AS Name`

Example

```
// Check the properties of vertex "player101".
nebula> FETCH PROP ON player "player101";
+-------------------------------------------------------+
| vertices_                                             |
+-------------------------------------------------------+
| ("player101" :player{age: 36, name: "Tony Parker"}) |
+-------------------------------------------------------+

// Update the age property and return name and the new age.
nebula> UPDATE VERTEX ON player "player101" \
        SET age = age + 2 \
        WHEN name == "Tony Parker" \
        YIELD name AS Name, age AS Age;
+---------------+-----+
| Name          | Age |
+---------------+-----+
| "Tony Parker" | 38  |
+---------------+-----+
```

Last update: May 7, 2021

4.11.4 UPSERT VERTEX

`UPSERT` is a combination of `UPDATE` and `INSERT`. Use `UPSERT VERTEX` to update properties of a vertex if it exists or insert a new vertex if it does not exist.

> **Note**
>
> An `UPSERT VERTEX` statement can only update properties on **ONE TAG** of a vertex.

The performance of `UPSERT` is much lower than that of `INSERT`, because `UPSERT` is a read-modify-write serialization operation at the partition level.

> **Danger**
>
> Don't use `UPSERT` for scenarios with highly concurrent writes. Use `UPDATE` or `INSERT` instead.

Syntax

```
UPSERT VERTEX ON <tag> <vid>
SET <update_prop>
[WHEN <condition>]
[YIELD <output>]
```

Field	Required	Description	Example
`ON <tag>`	Yes	Specifies the tag of the vertex. The properties to be updated must be on this tag.	`ON player`
`<vid>`	Yes	Specifies the ID of the vertex to be updated or inserted.	`"player100"`
`SET <update_prop>`	Yes	Specifies the properties to be updated and how they will be updated.	`SET age = age +1`
`WHEN <condition>`	No	Specifies the filter conditions.	`WHEN name == "Tim"`
`YIELD <output>`	No	Specifies the output format of the statement.	`YIELD name AS Name`

Insert a vertex if it does not exist

If a vertex does not exist, it is created no matter the conditions in the `WHEN` clause are met or not, and the `SET` clause always takes effect. The property values of the new vertex depends on:

- How the `SET` clause is defined
- The default value of the properties

For example, if:

- The vertex to be inserted will have properties `name` and `age` based on tag `player`.
- The `SET` clause specifies that `age = 30`.

Then the property values in different cases are listed as follows:

Are `WHEN` conditions met	If properties has default values	Value of `name`	Value of `age`
Yes	Yes	The default value	`30`
Yes	No	`NULL`	`30`
No	Yes	The default value	`30`
No	No	`NULL`	`30`

Here are some examples:

```
// Check if the following three vertices exists.
nebula> FETCH PROP ON * "player666", "player667", "player668";
Empty set

// The result Empty set indicates that the vertices don't exist.

nebula> UPSERT VERTEX ON player "player666" \
        SET age = 30 \
        WHEN name == "Joe" \
        YIELD name AS Name, age AS Age;
+----------+----------+
| Name     | Age      |
+----------+----------+
| __NULL__ | 30       |
+----------+----------+

nebula> UPSERT VERTEX ON player "player666" SET age = 31 WHEN name == "Joe" YIELD name AS Name, age AS Age;
+----------+-----+
| Name     | Age |
+----------+-----+
| __NULL__ | 30  |
+----------+-----+

nebula> UPSERT VERTEX ON player "player667" \
        SET age = 31 \
        YIELD name AS Name, age AS Age;
+----------+-----+
| Name     | Age |
+----------+-----+
| __NULL__ | 31  |
+----------+-----+

nebula> UPSERT VERTEX ON player "player668" \
        SET name = "Amber", age = age + 1 \
        YIELD name AS Name, age AS Age;
+---------+----------+
| Name    | Age      |
+---------+----------+
| "Amber" | __NULL__ |
+---------+----------+
```

In the last query of the preceding example, since `age` has no default value, when the vertex is created, `age` is `NULL`, and `age = age + 1` does not take effect. But if it has a default value, `age = age + 1` in the `SET` clause will take effect. For example:

```
nebula> CREATE TAG player_with_default(name string, age int DEFAULT 20);
Execution succeeded

nebula> UPSERT VERTEX ON player_with_default "player101" \
        SET age = age + 1 \
        YIELD name AS Name, age AS Age;

+----------+-----+
| Name     | Age |
+----------+-----+
| __NULL__ | 21  |
+----------+-----+
```

Update a vertex if it exists

If the vertex exists and the `WHEN` conditions are met, the vertex is updated.

```
nebula> FETCH PROP ON player "player101";
+-------------------------------------------------+
| vertices_                                       |
+-------------------------------------------------+
| ("player101" :player{age: 42, name: "Tony Parker"}) |
+-------------------------------------------------+
```

```
nebula> UPSERT VERTEX ON player "player101" \
        SET age = age + 2 \
        WHEN name == "Tony Parker" \
        YIELD name AS Name, age AS Age;
+---------------+-----+
| Name          | Age |
+---------------+-----+
| "Tony Parker" | 44  |
+---------------+-----+
```

If the vertex exists and the `WHEN` conditions are not met, the update does not take effect.

```
nebula> FETCH PROP ON player "player101";
+-------------------------------------------------+
| vertices_                                       |
+-------------------------------------------------+
| ("player101" :player{age: 44, name: "Tony Parker"}) |
+-------------------------------------------------+

nebula> UPSERT VERTEX ON player "player101" \
        SET age = age + 2 \
        WHEN name == "Someone else" \
        YIELD name AS Name, age AS Age;
+---------------+-----+
| Name          | Age |
+---------------+-----+
| "Tony Parker" | 44  |
+---------------+-----+
```

Last update: May 10, 2021

4.12 Edge statements

4.12.1 INSERT EDGE

The `INSERT EDGE` statement inserts an edge from a source vertex (given by src_vid) to a destination vertex (given by dst_vid) with a specific rank.

When inserting an edge that already exists, `INSERT VERTEX` **overrides** the edge.

Syntax

```
INSERT EDGE <edge_type> ( <prop_name_list> ) {VALUES | VALUE}
<src_vid> -> <dst_vid>[@<rank>] : ( <prop_value_list> )
[, <src_vid> -> <dst_vid>[@<rank>] : ( <prop_value_list> ), ...]

<prop_name_list> ::=
  [ <prop_name> [, <prop_name> ] ...]

<prop_value_list> ::=
  [ <prop_value> [, <prop_value> ] ...]
```

- `<edge_type>` denotes the edge type, which must be created before `INSERT EDGE`. Only one edge type can be specified in this statement.
- `<prop_name_list>` is the property name list in the given `<edge_type>`.
- `<prop_value_list>` must provide the value list according to `<prop_name_list>`. If the property values do not match the data type in the edge type, an error is returned. When the `NOT NULL` constraint is set for a given property, an error is returned if no property is given. When the default value for a property is `NULL`, you can omit to specify the property value.
- `rank` is optional. It specifies the edge rank of the same edge type. If not specified, the default value is 0. You can insert many edges with the same edge type for two vertices by using different rank values.

> **OpenCypher compatibility**
>
> OpenCypher has no such a concept as rank.

Examples

```
nebula> CREATE EDGE e1();                   -- create edge type t1 with empty property
nebula> INSERT EDGE e1 () VALUES "10"->"11":();    -- insert an edge from vertex "10" to vertex "11" with empty property
nebula> INSERT EDGE e1 () VALUES "10"->"11"@1:();  -- insert an edge from vertex "10" to vertex "11" with empty property, the edge rank is 1
```

```
nebula> CREATE EDGE e2 (name string, age int);                    -- create edge type e2 with two properties
nebula> INSERT EDGE e2 (name, age) VALUES "11"->"13":("n1", 1);          -- insert edge from "11" to "13" with two properties
nebula> INSERT EDGE e2 (name, age) VALUES \
"12"->"13":("n1", 1), "13"->"14":("n2", 2);                         -- insert two edges
nebula> INSERT EDGE e2 (name, age) VALUES "11"->"13":("n1", "a13");      -- ERROR. "a13" is not int
```

An edge can be inserted/written multiple times. Only the last written values can be read.

```
-- insert edge with the new values.
nebula> INSERT EDGE e2 (name, age) VALUES "11"->"13":("n1", 12);
nebula> INSERT EDGE e2 (name, age) VALUES "11"->"13":("n1", 13);
nebula> INSERT EDGE e2 (name, age) VALUES "11"->"13":("n1", 14);

// Only the last write can be read
nebula> FETCH PROP ON e2 "11"->"13";
+-----------------------------------------+
| edges_                                  |
+-----------------------------------------+
| [:e2 "11"->"13" @0 {age: 14, name: "n1"}] |
+-----------------------------------------+
```

Last update: April 22, 2021

4.12.2 DELETE EDGE

```
DELETE EDGE <edge_type> <src_vid> -> <dst_vid>[@<rank>] [, <edge_type> <src_vid> -> <dst_vid>[@<rank>] ...]
```

Use `DELETE EDGE` to delete edges. The `DELETE EDGE` statement deletes one edge or multiple edges at a time. You can use `DELETE EDGE` together with pipe. For more information about pipe, see Pipe operator.

Examples

```
nebula> DELETE EDGE serve "player100" -> "team200"@0;
```

This query deletes the `serve` edge from `"player100"` to `"team200"`, of which the rank value is 0.

```
nebula> GO FROM "player100" OVER follow WHERE follow._dst == "team200" YIELD follow._src AS src, follow._dst AS dst, follow._rank AS rank | \
DELETE EDGE follow $-.src->$-.dst @ $-.rank;
```

This query shows that you can use `DELETE EDGE` together with pipe. This query first traverses all the `follow` edges with different rank values from `"player100"` to `"team200"` then deletes them.

To delete all the outgoing edges for a vertex, delete the vertex. For more information, see DELETE VERTEX.

> **Note**
>
> Atomic operation is not guaranteed during the entire process for now, so please retry when a failure occurs.

Last update: April 22, 2021

4.12.3 UPDATE EDGE

The `UPDATE EDGE` statement updates properties on an edge. `UPDATE EDGE` supports compare-and-set (CAS).

Syntax

```
UPDATE EDGE ON <edge_type>
<src_vid> -> <dst_vid> [@<rank>]
SET <update_prop>
[WHEN <condition>]
[YIELD <output>]
```

Field	Required	Description	Example
`ON <edge_type>`	Yes	Specifies the type of the edge. The properties to be updated must be on this edge type.	`ON serve`
`<src_vid>`	Yes	Specifies the source vertex ID of the edge.	`"player100"`
`<dst_vid>`	Yes	Specifies the destination vertex ID of the edge.	`"team204"`
`<rank>`	No	Specifies the rank of the edge.	`10`
`SET <update_prop>`	Yes	Specifies the properties to be updated and how they will be updated.	`SET start_year = start_year +1`
`WHEN <condition>`	No	Specifies the filter conditions. If `<condition>` evaluates to `false`, the `SET` clause does not take effect.	`WHEN end_year < 2010`
`YIELD <output>`	No	Specifies the output format of the statement.	`YIELD start_year AS Start_Year`

Example

```
// Check the properties of the edge with the GO statement.
nebula> GO FROM "player100" \
        OVER serve \
        YIELD serve.start_year, serve.end_year;
+------------------+----------------+
| serve.start_year | serve.end_year |
+------------------+----------------+
| 1997             | 2016           |
+------------------+----------------+

// Update the start_year property and return end_year and the new start_year.

nebula> UPDATE EDGE on serve "player100" -> "team204"@0 \
        SET start_year = start_year + 1 \
        WHEN end_year > 2010 \
        YIELD start_year, end_year;
+------------+----------+
| start_year | end_year |
+------------+----------+
| 1998       | 2016     |
+------------+----------+
```

Last update: May 10, 2021

4.12.4 UPSERT EDGE

`UPSERT` is a combination of `UPDATE` and `INSERT`. Use `UPSERT EDGE` to update properties of an edge if it exists or insert a new edge if it does not exist.

The performance of `UPSERT` is much lower than that of `INSERT`, because `UPSERT` is a read-modify-write serialization operation at the partition level.

> **Danger**
>
> Don't use `UPSERT` for scenarios with highly concurrent writes. Use `UPDATE` or `INSERT` instead.

Syntax

```
UPSERT EDGE ON <edge_type>
<src_vid> -> <dst_vid> [@rank]
SET <update_prop>
[WHEN <condition>]
[YIELD <properties>]
```

Field	Required	Description	Example
`ON <edge_type>`	Yes	Specifies the type of the edge. The properties to be updated must be on this edge type.	`ON serve`
`<src_vid>`	Yes	Specifies the source vertex ID of the edge.	`"player100"`
`<dst_vid>`	Yes	Specifies the destination vertex ID of the edge.	`"team204"`
`<rank>`	No	Specifies the rank of the edge.	`10`
`SET <update_prop>`	Yes	Specifies the properties to be updated and how they will be updated.	`SET start_year = start_year +1`
`WHEN <condition>`	No	Specifies the filter conditions.	`WHEN end_year < 2010`
`YIELD <output>`	No	Specifies the output format of the statement.	`YIELD start_year AS Start_Year`

Insert an edge if it does not exist

If an edge does not exist, it is created no matter the conditions in the `WHEN` clause are met or not, and the `SET` clause takes effect. The property values of the new edge depends on:

- How the `SET` clause is defined
- The default value of the properties

For example, if:

- The edge to be inserted will have properties `start_year` and `end_year` based on the edge type `serve`.
- The `SET` clause specifies that `end_year = 2021`.

Then the property values in different cases are listed as follows:

Are `WHEN` conditions met	If properties has default values	Value of `start_year`	Value of `end_year`
Yes	Yes	The default value	`2021`
Yes	No	`NULL`	`2021`
No	Yes	The default value	`2021`
No	No	`NULL`	`2021`

Here are some examples:

```
// Check if the following three vertices has any outgoing serve edge.
nebula> GO FROM "player666", "player667", "player668" \
        OVER serve \
        YIELD serve.start_year, serve.end_year;
Empty set

// The result Empty set indicates that the edges don't exist.

nebula> UPSERT EDGE on serve \
        "player666" -> "team200"@0 \
        SET end_year = 2021 \
        WHEN end_year == 2010 \
        YIELD start_year, end_year;
+------------+----------+
| start_year | end_year |
+------------+----------+
| __NULL__   | 2021     |
+------------+----------+

nebula> UPSERT EDGE on serve \
        "player666" -> "team200"@0 \
        SET end_year = 2022 \
        WHEN end_year == 2010 \
        YIELD start_year, end_year;
+------------+----------+
| start_year | end_year |
+------------+----------+
| __NULL__   | 2021     |
+------------+----------+

nebula> UPSERT EDGE on serve \
        "player667" -> "team200"@0 \
        SET end_year = 2022 \
        YIELD start_year, end_year;
+------------+----------+
| start_year | end_year |
+------------+----------+
| __NULL__   | 2022     |
+------------+----------+

nebula> UPSERT EDGE on serve \
        "player668" -> "team200"@0 \
        SET start_year = 2000, end_year = end_year + 1 \
        YIELD start_year, end_year;
+------------+----------+
| start_year | end_year |
+------------+----------+
| 2000       | __NULL__ |
+------------+----------+
```

In the last query of the preceding example, since `end_year` has no default value, when the edge is created, `end_year` is `NULL`, and `end_year = end_year + 1` does not take effect. But if it has a default value, `end_year = end_year + 1` in the `SET` clause will take effect. For example:

```
nebula> CREATE EDGE serve_with_default(start_year int, end_year DEFAULT 2010);
Execution succeeded

nebula> UPSERT EDGE on serve_with_default \
        "player668" -> "team200" \
        SET end_year = end_year + 1 \
        YIELD start_year, end_year;
+------------+----------+
| start_year | end_year |
+------------+----------+
| __NULL__   | 2011     |
+------------+----------+
```

Update an edge if it exists

If the edge exists and the `WHEN` conditions are met, the edge is updated.

```
nebula> MATCH (v:player{name:"Ben Simmons"})-[e:serve]-(v2) \
        RETURN e;
+------------------------------------------------------------------------+
| e                                                                      |
+------------------------------------------------------------------------+
| [:serve "player149"->"team219" @0 {end_year: 2019, start_year: 2016}] |
+------------------------------------------------------------------------+

nebula> UPSERT EDGE on serve \
        "player149" -> "team219" \
        SET end_year = end_year + 1 \
        WHEN start_year == 2016 \
        YIELD start_year, end_year;
+------------+----------+
| start_year | end_year |
+------------+----------+
| 2016       | 2020     |
+------------+----------+
```

If the edge exists and the `WHEN` conditions are not met, the edge does not take effect.

```
nebula> MATCH (v:player{name:"Ben Simmons"})-[e:serve]-(v2) \
        RETURN e;
+------------------------------------------------------------------------+
| e                                                                      |
+------------------------------------------------------------------------+
| [:serve "player149"->"team219" @0 {end_year: 2020, start_year: 2016}] |
+------------------------------------------------------------------------+


nebula> UPSERT EDGE on serve \
        "player149" -> "team219" \
        SET end_year = end_year + 1 \
        WHEN start_year != 2016 \
        YIELD start_year, end_year;
+------------+----------+
| start_year | end_year |
+------------+----------+
| 2016       | 2020     |
+------------+----------+
```

Last update: May 10, 2021

4.13 Native index statements

4.13.1 CREATE INDEX

Use `CREATE INDEX` to add native indexes for existing tags, edge types or properties.

> **Note**
>
> For how to create text-based indexes, see Deploy full-text index.

Most graph queries start the traversal from a list of vertices or edges that are identified by their properties. Indexes make these global retrieval operations efficient on large graphs.

Prerequisites

Before you create an index, make sure that the relative tag or edge type is created. For how to create tags or edge types, see CREATE TAG and CREATE EDGE.

Must-read for using index

Correct use of indexes can speed up queries, but indexes can dramatically reduce the write performance. The performance reduction can be as much as 90% or even more. **DO NOT** use indexes in production environments unless you are fully aware of their influences on your service.

If you must use indexes, we suggest that you:

1. Import data into Nebula Graph.
2. Create indexes.
3. Rebuild the indexes.

The preceding workflow minimizes the negative influences of using indexes.

Syntax

```
CREATE {TAG | EDGE} INDEX [IF NOT EXISTS] <index_name> ON {<tag_name> | <edge_name>} ([prop_name_list])
```

- `index_name`: The name of the index. It must be unique in a graph space. A recommended way of naming is `i_tagName_propName`.
- `IF NOT EXISTS`: Creating an existent index results in an error. You can use the `IF NOT EXISTS` option to conditionally create the index and avoid the error.
- `prop_name_list`:
  - To index a **variable** string property, you must use the `prop_name(length)` syntax to specify an index length.

    > **Note**
    >
    > Long indexes decrease the scan performance of the Storage Service and use more memory. We suggest that you set the indexing length the same as that of the longest string to be indexed. The longest indexing length is 255. Strings longer than 255 are truncated.

  - To index a **fixed-length** string property, you must use the `prop_name` syntax, and the index length is the string length you set.
  - To index a tag or an edge type, ignore the `prop_name_list` in the parentheses.

    > **Note**
    >
    > When there is an index for any property of a tag or an edge type, creating another index for the tag or edge type is neither supported nor necessary.

Implementation of the operation

Nebula Graph implements the creation of the index asynchronously in the next heartbeat cycle. To make sure the creation is successful, take one of the following approaches:

- Find the new index in the result of `SHOW TAG/EDGE INDEXES`.
- Wait for two heartbeat cycles, i.e., 20 seconds.

To change the heartbeat interval, modify the `heartbeat_interval_secs` parameter in the [configuration files] for all services.

Create tag/edge type indexes

The following statement creates an index on the `player` tag.

```
nebula> CREATE TAG INDEX player_index on player();
```

The following statement creates indexes on the edge type `like`.

```
nebula> CREATE EDGE INDEX like_index on like();
```

After indexing a tag or an edge type, you can use the `LOOKUP` statement to retrieve the VID of all vertices with the tag, or the source vertex ID, destination vertex ID, and ranks of all edges with the edge type. For more information, see List vertices or edges with a tag or an edge type.

Create single-property indexes

```
nebula> CREATE TAG INDEX player_index_0 on player(name(10));
```

The preceding statement creates an index for the `name` property on all vertices carrying the `player` tag. This statement creates an index using the first 10 characters of the name property.

```
nebula> CREATE TAG var_string(p1 string);
nebula> CREATE TAG INDEX var ON var_string(p1(10));
```

```
nebula> CREATE TAG fix_string(p1 FIXED_STRING(10));
nebula> CREATE TAG INDEX fix ON fix_string(p1);
```

```
nebula> CREATE EDGE INDEX follow_index_0 on follow(degree);
```

The preceding statement creates an index for the `degree` property on all edges carrying the `follow` edge type.

Create composite property indexes

An index on multiple properties is called a composite index.

> **Note**
>
> Creating index across multiple tags is not supported.

Consider the following example:

```
nebula> CREATE TAG INDEX player_index_1 on player(name(10), age);
```

This statement creates a composite index for the `name` and `age` property on all vertices carrying the `player` tag.

Nebula Graph follows the left matching principle to select indexes. That is, columns in the `WHERE` conditions must be in the first N columns of the index. For example:

```
nebula> CREATE TAG INDEX example_index ON TAG t(p1, p2, p3);  -- Create an index for the first 3 properties of tag t
nebula> LOOKUP ON t WHERE p2 == 1 and p3 == 1; -- Not supported
nebula> LOOKUP ON t WHERE p1 == 1;  -- Supported
nebula> LOOKUP ON t WHERE p1 == 1 and p2 == 1;  -- Supported
nebula> LOOKUP ON t WHERE p1 == 1 and p2 == 1 and p3 == 1;  -- Supported
```

Using index

After the index is created and data is inserted, you can use LOOKUP or MATCH to query the data.

You do not need to specify which indexes to use in a query, Nebula Graph figures that out by itself.

Last update: April 22, 2021

4.13.2 Show INDEXES

```
SHOW {TAG | EDGE} INDEXES
```

Use `SHOW INDEXES` to list the defined tag or edge type indexes names.

Example

```
nebula> SHOW TAG INDEXES;
+------------------+
| Names            |
+------------------+
| "fix"            |
+------------------+
| "player_index_0" |
+------------------+
| "player_index_1" |
+------------------+
| "var"            |
+------------------+

nebula> SHOW EDGE INDEXES;
+------------------+
| Names            |
+------------------+
| "follow_index_0" |
+------------------+
```

Last update: December 16, 2020

4.13.3 SHOW CREATE INDEX

`SHOW CREATE INDEX` shows the statement that an index was created with. You can find the detailed information of the index, such as the property that the index is created for.

Syntax

```
SHOW CREATE {TAG | EDGE} INDEX <index_name>;
```

Examples

You can run `SHOW TAG INDEXES` to list all tag indexes, and then use `SHOW CREATE TAG INDEX` to show how a tag index was created.

```
nebula> SHOW TAG INDEXES;
+------------------+
| Names            |
+------------------+
| "player_index_0" |
+------------------+
| "player_index_1" |
+------------------+

nebula> SHOW CREATE TAG INDEX player_index_1;
+------------------+-------------------------------------------------+
| Tag Index Name   | Create Tag Index                                |
+------------------+-------------------------------------------------+
| "player_index_1" | "CREATE TAG INDEX `player_index_1` ON `player` ( |
|                  |  `name(20)`                                     |
|                  | )"                                              |
+------------------+-------------------------------------------------+
```

Edge indexes can be queried through a similar approach:

```
nebula> SHOW EDGE INDEXES;
+----------------+
| Names          |
+----------------+
| "index_follow" |
+----------------+

nebula> SHOW CREATE EDGE INDEX index index_follow;
+-----------------+-------------------------------------------------+
| Edge Index Name | Create Edge Index                               |
+-----------------+-------------------------------------------------+
| "index_follow"  | "CREATE EDGE INDEX `index_follow` ON `follow` ( |
|                 |  `degree`                                       |
|                 | )"                                              |
+-----------------+-------------------------------------------------+
```

Last update: March 29, 2021

4.13.4 DESCRIBE INDEX

```
DESCRIBE {TAG | EDGE} INDEX <index_name>
```

Use `DESCRIBE INDEX` to get information about the index. `DESCRIBE INDEX` returns the following columns:

- `Field`

  The property name. - `Type`

  The property type.

Example

```
nebula> DESCRIBE TAG INDEX player_index_0;
+--------+--------------------+
| Field  | Type               |
+--------+--------------------+
| "name" | "fixed_string(30)" |
+--------+--------------------+

nebula> DESCRIBE TAG INDEX player_index_1;
+--------+--------------------+
| Field  | Type               |
+--------+--------------------+
| "name" | "fixed_string(10)" |
+--------+--------------------+
| "age"  | "int64"            |
+--------+--------------------+
```

Last update: December 16, 2020

4.13.5 REBUILD INDEX

Danger

If data is updated or inserted before the index's creation, you must rebuild the indexes **manually** to make sure that the indexes contain the previously added data. If the index is created before any data insertion, there is no need to rebuild the index.

Danger

During the rebuilding, all queries skip the index and perform sequential scans. This means that the return results can be different because not all the data is indexed during rebuilding.

```
REBUILD {TAG | EDGE} INDEX [<index_name_list>]

<index_name_list>::=
    [index_name [, index_name] ...]
```

Use `REBUILD INDEX` to rebuild the created tag or edge type index. For details on how to create an index, see CREATE INDEX.

Multiple indexes are permitted in a single `REBUILD` statement, separated by commas. When the index name is not specified, all tag or edge indexes are rebuilt.

After rebuilding is complete, you can use the `SHOW {TAG | EDGE} INDEX STATUS` command to check if the index is successfully rebuilt. For details on index status, see SHOW INDEX STATUS.

Example

```
nebula> CREATE TAG person(name string, age int, gender string, email string);
Execution succeeded (Time spent: 10.051/11.397 ms)

nebula> CREATE TAG INDEX single_person_index ON person(name(10));
Execution succeeded (Time spent: 2.168/3.379 ms)

nebula> REBUILD TAG INDEX single_person_index;
+------------+
| New Job Id |
+------------+
| 66         |
+------------+

nebula> SHOW TAG INDEX STATUS;
```

Nebula Graph creates a job to rebuild the index. The job ID is displayed in the preceding return message. To check if the rebuilding process is complete, use the `SHOW JOB <job_id>` statement. For more information, see SHOW JOB.

Legacy version compatibility

In Nebula Graph 2.x, the `OFFLINE` options is no longer needed and not supported.

Last update: April 19, 2021

4.13.6 SHOW INDEX STATUS

```
SHOW {TAG | EDGE} INDEX STATUS
```

`SHOW INDEX STATUS` returns the created tag or edge type index status. For details on how to create index, see CREATE INDEX.

`SHOW INDEX STATUS` returns the following fields:

- `Name`

  The index name.

- `Index Status`

  Index Status includes `QUEUE`, `RUNNING`, `FINISHED`, `FAILED`, `STOPPED`, `INVALID`.

Example

```
nebula> SHOW TAG INDEX STATUS;
+----------------------+-------------+
| Name                 | Index Status |
+----------------------+-------------+
| "player_index_0"     | "FINISHED"   |
+----------------------+-------------+
| "player_index_1"     | "FINISHED"   |
+----------------------+-------------+
```

Last update: December 16, 2020

4.13.7 DROP INDEX

```
DROP {TAG | EDGE} INDEX [IF EXISTS] <index_name>
```

The `DROP INDEX` statement removes an existing index from the current graph space. Removing a nonexistent index results in an error. You can use the `IF EXISTS` option to conditionally drop the index and avoid the error. To run this statement you need some privilege. For information about the built-in roles in Nebula Graph, see Built-in roles.

Example

```
nebula> DROP TAG INDEX player_index_0;
```

This query drops a tag index names `player_index_0`.

Last update: March 19, 2021

4.14 Full-text index statements

4.14.1 Index overview

Indexes are built to fast process graph queries. Nebula Graph supports two kinds of indexes: native indexes and full-text indexes. This topic introduces the index types and helps choose the right index.

Native indexes

Native indexes allow querying data based on a given property. There are two kinds of native indexes: tag index and edge type index. Native indexes must be updated manually. You can use the `REBUILD INDEX` statement to update native indexes. Native indexes support indexing multiple properties on a tag or an edge type (composite indexes), but do not support indexing across multiple tags or edge types.

You can do partial match search by using composite indexes. Use composite indexes only for partial match searches when the declared fields in the composite index are used from left to right. For more information, see LOOKUP FAQ.

String operators like `CONTAINS` and `STARTS WITH` are not allowed in native index searching. Use full-text indexes to do fuzzy search.

OPERATIONS ON NATIVE INDEXES

You can do the following operations against native indexes:

- Create index
- Show index
- Describe index
- Rebuild index
- Show index status
- Drop index
- Query index

Full-text indexes

Full-text indexes are used to do prefix, wildcard, regexp, and fuzzy search on a string property. Full-text indexes allow indexing just one property. Only strings within a specified length (no longer than 256 bytes) are indexed. Full-text indexes do not support logical operations such as `AND`, `OR` and `NOT`. To do complete text match, use native indexes.

OPERATIONS ON FULL-TEXT INDEXES

Before doing any operations on full-text indexes, please mak sure that you deploy full-text indexes. Details on full-text indexes deployment, see Deploy Elasticsearch and Deploy Listener. At this time, full-text indexes are created automatically on the Elasticsearch cluster. And rebuilding or altering full-text indexes are not supported. To drop full-text indexes, you need to drop them on the Elasticsearch cluster manually. To query full-text indexes, see Search with full-text indexes.

Null values

Indexes do not support indexing null values at this time.

Range queries

In addition to querying single results from native indexes, you can also do range queries. Not all the native indexes support range queries. You can only do range search for numeric, date, and time type properties.

Last update: December 16, 2020

4.14.2 Full-text index restrictions

This document holds the restrictions for full-text indexes. Please read the restrictions very carefully before using the full-text indexes. For now, full-text search has the following limitations:

1. The maximum indexing string length is 256 bytes. The part of data that exceeds 256 bytes will not be indexed.
2. Full-text index can not be applied to more than one property at a time (similar to a composite index).
3. The `WHERE` clause in full-text search statement `LOOKUP` does not support logical expressions `AND` and `OR`.
4. Full-text index can not be applied to multiple tags search.
5. Sorting for the returned results of the full-text search is not supported. Data is returned in the order of data insertion.
6. Full-text index can not search the null properties.
7. Rebuilding or altering Elasticsearch indexes is not supported at this time.
8. Pipe is not supported in the `LOOKUP` statement, excluding the examples in our document.
9. Full-text search only works on single terms.
10. Full-text indexes are not deleted together with the graph space.
11. Make sure that you start the Elasticsearch cluster and Nebula Graph at the same time. If not, the data writing on the Elasticsearch cluster can be incomplete.
12. Do not contain `'` or `\` in the vertex or edge values. If not, a error is caused in the Elasticsearch cluster storage.
13. It may take a while for Elasticsearch to create indexes. If Nebula Graph warns no index is found, wait for the index to take effect.

Last update: May 7, 2021

4.14.3 Deploy full-text index

Nebula Graph full-text indexes are powered by Elasticsearch. This means that you can use Elasticsearch full-text query language to retrieve what you want. Full-text indexes are managed through built-in procedures. They can be created only for variable `STRING` and `FIXED_STRING` properties when the listener cluster and the Elasticsearch cluster are deployed.

Before you start

Before you start using the full-text index, please make sure that you know the restrictions.

Deploy Elasticsearch cluster

To deploy an Elasticsearch cluster, see the Elasticsearch documentation.

When the Elasticsearch cluster is started, add the template file for the Nebula Graph full-text index. Take the following sample template for example:

```
{
 "template": "nebula*",
  "settings": {
    "index": {
      "number_of_shards": 3,
      "number_of_replicas": 1
    }
  },
  "mappings": {
    "properties" : {
            "tag_id" : { "type" : "long" },
            "column_id" : { "type" : "text" },
            "value" :{ "type" : "keyword"}
        }
  }
}
```

Make sure that you specify the following fields in strict accordance with the preceding template format:

```
"template": "nebula*"
"tag_id" : { "type" : "long" },
"column_id" : { "type" : "text" },
"value" :{ "type" : "keyword"}
```

You can configure the Elasticsearch to meet your business needs. To customize the Elasticsearch, see Elasticsearch Document.

Sign in to the text search clients

```
SIGN IN TEXT SERVICE [(<elastic_ip:port> [,<username>, <password>]), (<elastic_ip:port>), ...]
```

When the Elasticsearch cluster is deployed, use the `SIGN IN` statement to sign in to the Elasticsearch clients. Multiple `elastic_ip:port` pairs are separated with commas. You must use the IPs and the port number in the configuration file for the Elasticsearch. For example:

```
nebula> SIGN IN TEXT SERVICE (127.0.0.1:9200);
```

Elasticsearch does not have username or password by default. If you configured a username and password, you need to specify in the `SIGN IN` statement.

Show text search clients

```
SHOW TEXT SEARCH CLIENTS
```

Use the `SHOW TEXT SEARCH CLIENTS` statement to list the text search clients. For example:

```
nebula> SHOW TEXT SEARCH CLIENTS;
+-------------+------+
| Host        | Port |
+-------------+------+
```

```
| "127.0.0.1" | 9200 |
+-------------+------+
| "127.0.0.1" | 9200 |
+-------------+------+
| "127.0.0.1" | 9200 |
+-------------+------+
```

Sign out to the text search clients

```
SIGN OUT TEXT SERVICE
```

Use the `SIGN OUT TEXT SERVICE` to sign out all the text search clients. For example:

```
nebula> SIGN OUT TEXT SERVICE;
```

Last update: December 17, 2020

4.14.4 Deploy Raft Listener for Nebula Storage service

Full-Text index data is written to the Elasticsearch cluster asynchronously. The Raft Listener (hereinafter shortened as Listener) is a separate process that fetches data from the Storage Service and writes them into the Elasticsearch cluster.

Prerequisites

- You have read and fully understand the restrictions for using Full-Text indexes.
- You have deployed a Nebula Graph cluster.
- You have prepared at least one extra Storage Server. To use the Full-Text search, you must run one or more Storage Server as the Raft Listener.

Precautions

- The Storage Service that you want to run as a Listener must have the same or later version with all the other Nebula Graph services in the cluster.
- For now, you can only add Listeners to a graph space once and for all. Trying to add listeners to a graph space that already has a listener will fail. To add multiple listeners, set them in one statement.

Step 1: Prepare the configuration file for the Listeners

You have to prepare a Listener configuration file on the machine that you want to deploy the Listeners. The file name must be `nebula-storaged-listener.conf`. A template is provided for your reference.

> **Note**
>
> Use real IP addresses in the configuration file instead of domain names or loopback IP addresses such as `127.0.0.1`.

Step 2: Start the Listeners

Run the following command to start the Listeners.

```
./bin/nebula-storaged --flagfile ${listener_config_path}/nebula-storaged-listener.conf
```

`${listener_config_path}` is the path where you store the Listener configuration file.

Step 3: Add Listeners to Nebula Graph

Connect to Nebula Graph and run `USE <space>` to enter the graph space that you want to create Full-Text indexes for. Then run the following statement to add the Listener into Nebula Graph.

> **Note**
>
> You must use real IPs for the listeners.

```
ADD LISTENER ELASTICSEARCH <listener_ip:port> [,<listener_ip:port>, ...]
```

Multiple `listener_ip:port` pairs are separated with commas. For example:

```
nebula> ADD LISTENER ELASTICSEARCH 192.168.8.5:46780,192.168.8.6:46780;
```

Show Listeners

Run the `SHOW LISTENER` statement to list the Listeners.

For example:

```
nebula> SHOW LISTENER;
+--------+-----------------+-----------------------+----------+
| PartId | Type            | Host                  | Status   |
+--------+-----------------+-----------------------+----------+
| 1      | "ELASTICSEARCH" | "[192.168.8.5:46780]" | "ONLINE" |
+--------+-----------------+-----------------------+----------+
| 2      | "ELASTICSEARCH" | "[192.168.8.5:46780]" | "ONLINE" |
+--------+-----------------+-----------------------+----------+
| 3      | "ELASTICSEARCH" | "[192.168.8.5:46780]" | "ONLINE" |
+--------+-----------------+-----------------------+----------+
```

Remove Listeners

Run the `REMOVE LISTENER ELASTICSEARCH` statement to remove all the Elasticsearch Listeners for a graph space.

For example:

```
nebula> REMOVE LISTENER ELASTICSEARCH;
```

What to do next

After deploying the Elasticsearch cluster and the Listeners, Full-Text indexes are created automatically on the Elasticsearch cluster. You can do Full-Text search now. For more information, see Full-Text search.

Last update: April 22, 2021

4.14.5 Full-text search

```
LOOKUP ON {<tag> | <edge_type>} WHERE <expression> [YIELD <return_list>]

<expression> ::=
    PREFIX | WILDCARD | REGEXP | FUZZY

<return_list>
    <prop_name> [AS <prop_alias>] [, <prop_name> [AS <prop_alias>] ...]
```

- PREFIX(schema_name.prop_name, prefix_string, row_limit, timeout)
- WILDCARD(schema_name.prop_name, wildcard_string, row_limit, timeout)
- REGEXP(schema_name.prop_name, regexp_string, row_limit, timeout)
- FUZZY(schema_name.prop_name, fuzzy_string, fuzziness, operator, row_limit, timeout)
  - `fuzziness` (optional): Maximum edit distance allowed for matching. The default value is `AUTO`. For other valid values and more information, see Elasticsearch document.
  - `operator` (optional): Boolean logic used to interpret text. Valid values are `OR` (default) and `AND`.
- `row_limit` (optional): Specifies the number of rows to return. The default value is 100.
- `timeout` (optional): Specifies the timeout time. The default value is 200ms.

Use the `LOOKUP ON` statement to do full-text search. The search string is specified in the `WHERE` clause. Before doing a full-text search, make sure that you deployed a Elasticsearch cluster and a Listener cluster. For more information, see Deploy Elasticsearch and Deploy Listener.

Before you start

Before you start using the full-text index, please make sure that you know the restrictions.

Natural language full-text search

A natural language search interprets the search string as a phrase in natural human language. The search is case-insensitive.

Examples

```
nebula> CREATE SPACE basketballplayer (partition_num=3,replica_factor=1, vid_type=fixed_string(30));
nebula> SIGN IN TEXT SERVICE (127.0.0.1:9200);
nebula> USE basketballplayer;
nebula> ADD LISTENER ELASTICSEARCH 192.168.8.5:46780;
nebula> CREATE TAG player(name string, age int);
nebula> CREATE TAG INDEX name ON player(name(20));
nebula> INSERT VERTEX player(name, age) VALUES \
  "Russell Westbrook": ("Russell Westbrook", 30), \
  "Chris Paul": ("Chris Paul", 33),\
  "Boris Diaw": ("Boris Diaw", 36),\
  "David West": ("David West", 38),\
  "Danny Green": ("Danny Green", 31),\
  "Tim Duncan": ("Tim Duncan", 42),\
  "James Harden": ("James Harden", 29),\
  "Tony Parker": ("Tony Parker", 36),\
  "Aron Baynes": ("Aron Baynes", 32),\
  "Ben Simmons": ("Ben Simmons", 22),\
  "Blake Griffin": ("Blake Griffin", 30);


nebula> LOOKUP ON player WHERE PREFIX(player.name, "B");
+-----------------+
| _vid            |
+-----------------+
| "Boris Diaw"    |
+-----------------+
| "Ben Simmons"   |
+-----------------+
| "Blake Griffin" |
+-----------------+

nebula> LOOKUP ON player WHERE WILDCARD(player.name, "*ri*") YIELD player.name, player.age;
+-----------------+-----------------+-----+
| _vid            | name            | age |
+-----------------+-----------------+-----+
| "Chris Paul"    | "Chris Paul"    | 33  |
```

```
+-----------------+-----------------+-----+
| "Boris Diaw"    | "Boris Diaw"    | 36  |
+-----------------+-----------------+-----+
| "Blake Griffin" | "Blake Griffin" | 30  |
+-----------------+-----------------+-----+

nebula> LOOKUP ON player WHERE WILDCARD(player.name, "*ri*") | YIELD count(*);
+----------+
| COUNT(*) |
+----------+
| 3        |
+----------+

nebula> LOOKUP ON player WHERE REGEXP(player.name, "R.*") YIELD player.name, player.age;
+--------------------+--------------------+-----+
| _vid               | name               | age |
+--------------------+--------------------+-----+
| "Russell Westbrook" | "Russell Westbrook" | 30  |
+--------------------+--------------------+-----+

nebula> LOOKUP ON player WHERE REGEXP(player.name, ".*");
+--------------------+
| _vid               |
+--------------------+
| "Danny Green"      |
+--------------------+
| "David West"       |
+--------------------+
| "Russell Westbrook" |
+--------------------+
...

nebula> LOOKUP ON player WHERE FUZZY(player.name, "Tim Dunncan", AUTO, OR) YIELD player.name;
+-------------+-------------+
| _vid        | name        |
+-------------+-------------+
| "Tim Duncan" | "Tim Duncan" |
+-------------+-------------+
```

Last update: April 13, 2021

4.15 Subgraph and path

4.15.1 GET SUBGRAPH

The `GET SUBGRAPH` statement retrieves information of vertices and edges reachable from the start vertices over the specified types of edges.

Syntax

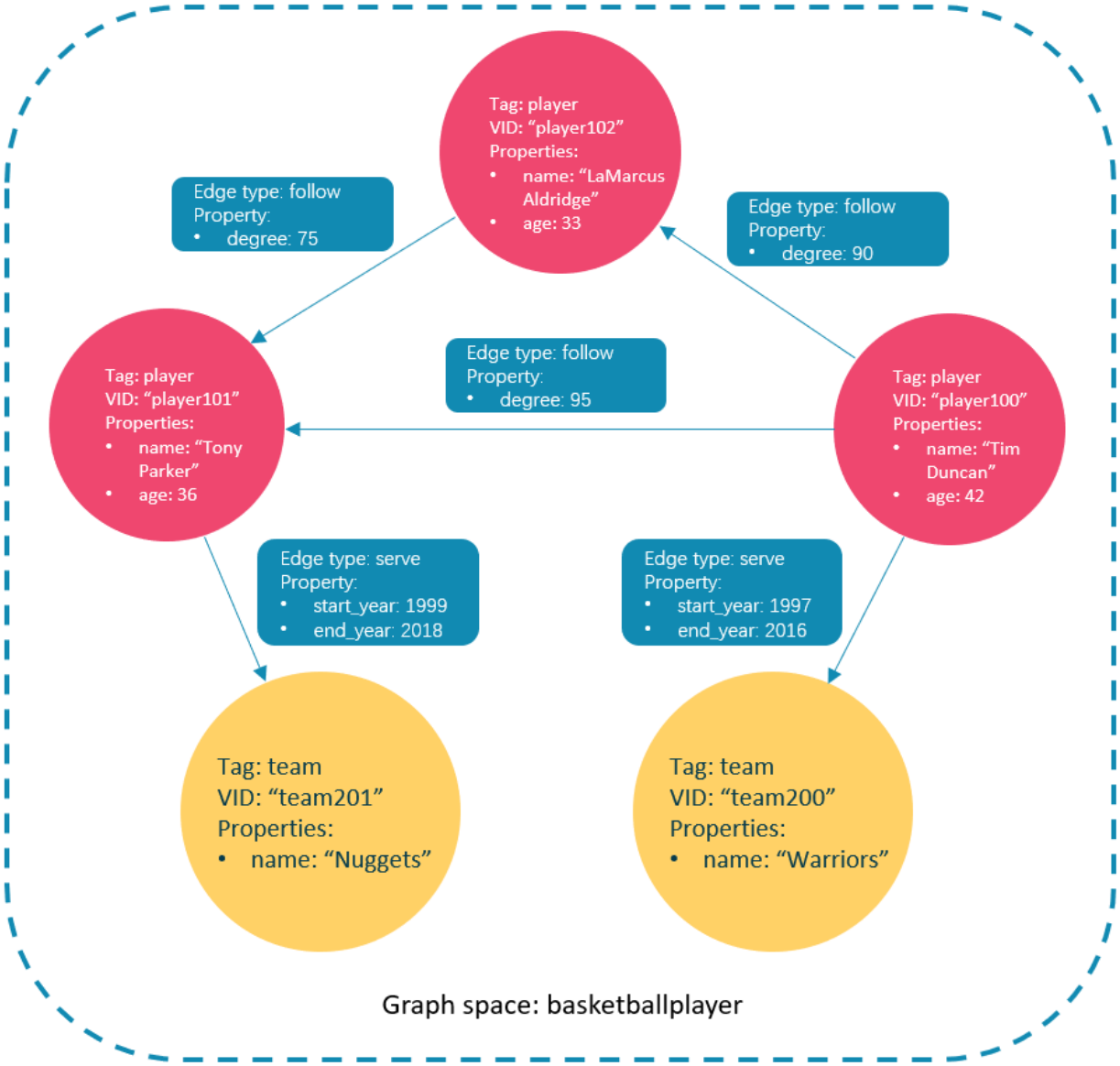
```
GET SUBGRAPH [<step_count> STEPS] FROM {<vid>, <vid>...}
[IN <edge_type>, <edge_type>...]
[OUT <edge_type>, <edge_type>...]
[BOTH <edge_type>, <edge_type>...]
```

Clause	Description
`STEPS`	Specifies the steps to go from the start vertices. A `step_count` must be a non-negative integer. Its default value is 1. When `<step_count>` is specified to `N`, the Nebula Graph returns zero to N steps subgraph.
`FROM`	Specifies the start vertices.
`IN`	Gets the subgraphs from the start vertices over the specified incoming edges (edges pointing to the start vertices).
`OUT`	Gets the subgraphs from the start vertices over the specified outgoing edges (edges pointing out from the start vertices).
`BOTH`	Gets the subgraphs from the start vertices over the specified types of edges, both incoming and outgoing.

When the traversal direction is not specified, both the incoming and outgoing edges are returned.

Examples

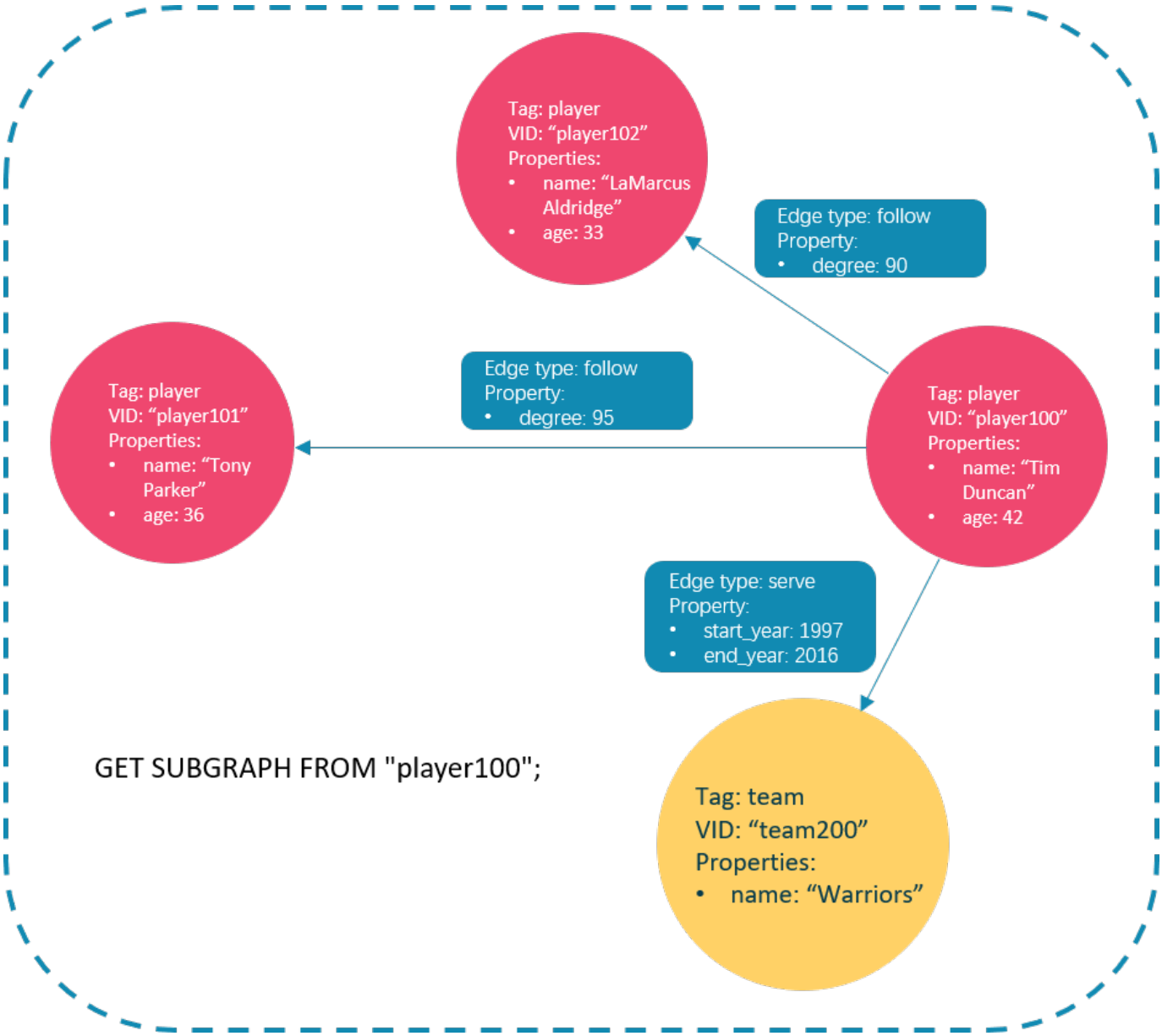
The following graph is used as the sample.



- Go one step from the vertex with VID "player100" over all types of edges and get the subgraph.

```
nebula> GET SUBGRAPH 1 STEPS FROM "player100";
+-------------------------------------------------------------------------------------------------------------------------------------------------------------+
+-------------------------------------------------------------------------------------------------------------------------------------------------------------+
| _vertices                                                                                                                                                   |
_edges                                                                                                                                                                |
+-------------------------------------------------------------------------------------------------------------------------------------------------------------+
+-------------------------------------------------------------------------------------------------------------------------------------------------------------+
| [(player100) player.name:Tim,player.age:42]                                                                                                                 | [player100-[follow]->player101@0
degree:96,player100-[follow]->player102@0 degree:90,player100-[serve]->team200@0 end_year:2016,start_year:1997] |
+-------------------------------------------------------------------------------------------------------------------------------------------------------------+
+-------------------------------------------------------------------------------------------------------------------------------------------------------------+
| [(player101) player.age:36,player.name:Tony Parker,(player102) player.age:33,player.name:LaMarcus Aldridge,(team200) team.name:Warriors] | [player102-
[follow]->player101@0 degree:75]                                                                                                                            |
+-------------------------------------------------------------------------------------------------------------------------------------------------------------+
+-------------------------------------------------------------------------------------------------------------------------------------------------------------+
Got 2 rows (time spent 6289/7423 us)
```

The returned subgraph is as follows.



- Go one step from the vertex with VID "player100" over incoming `follow` edges and get the subgraph.

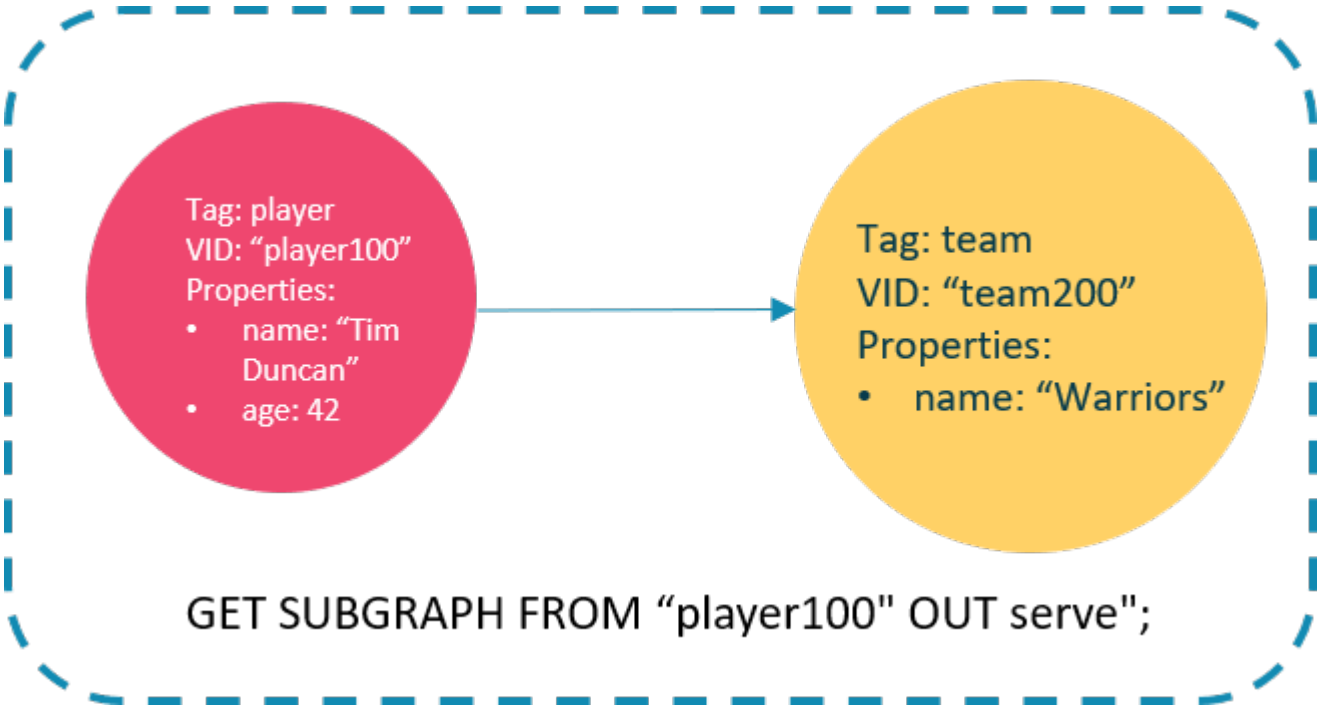
```
nebula> GET SUBGRAPH 1 STEPS FROM "player100" IN follow;
+-----------+--------+
| _vertices | _edges |
+-----------+--------+
| []        | []     |
+-----------+--------+
| []        | []     |
+-----------+--------+
Got 2 rows (time spent 2292/3091 us)
```

There is no incoming `follow` edge to "player100", so no vertex or edge is returned.

- Go one step from the vertex "player100" over outgoing `serve` edges and get the subgraph.

```
nebula> GET SUBGRAPH 1 STEPS FROM "player100" OUT serve;
+-----------------------------------------+-------------------------------------------------------------+
| _vertices                               | _edges                                                      |
+-----------------------------------------+-------------------------------------------------------------+
| [(player100) player.age:42,player.name:Tim] | [player100-[serve]->team200@0 start_year:1997,end_year:2016] |
+-----------------------------------------+-------------------------------------------------------------+
| [(team200) team.name:Warriors]          | []                                                          |
+-----------------------------------------+-------------------------------------------------------------+
Got 2 rows (time spent 2107/2547 us)
```

The returned subgraph is as follows.



Last update: April 15, 2021

4.15.2 FIND PATH

```
FIND { SHORTEST | ALL | NOLOOP } PATH FROM <vertex_id_list> TO <vertex_id_list>
OVER <edge_type_list> [REVERSELY | BIDIRECT] [UPTO <N> STEPS] [| ORDER BY $-.path] [| LIMIT <M>]

<vertex_id_list> ::=
    [vertex_id [, vertex_id] ...]
```

The `FIND PATH` statement finds the paths between the selected source vertices and destination vertices.

- `SHORTEST` finds the shortest path.
- `ALL` finds all the paths.
- `<vertex_id_list>` is a list of vertex IDs separated with commas (,). It supports `$-` and `$var`.
- `<edge_type_list>` is a list of edge types separated with commas (,). `*` is all edge types.
- `<N>` is the hop number. The default value is 5.
- `<M>` specifies the maximum number of rows to return.

Limitations

- When a list of source and/or destination vertex IDs are specified, the paths between any source vertices and the destination vertices is returned.
- There can be cycles when searching all paths.
- `FIND PATH` does not support filtering with `WHERE` clauses.
- `FIND PATH` does not support specifying a direction.
- `FIND PATH` is a single-thread procedure, so it uses much memory.
- If `NOLOOP` is not used, `FIND PATH` can retrieve paths containing cycles. If `NOLOOP` is used, `FIND PATH` can retrieve paths without cycles.

Examples

In Nebula Console, a path is shown as `vertex_id <edge_name, rank> vertex_id`.

```
nebula> FIND SHORTEST PATH FROM "player102" TO "team201" OVER *;
+------------------------------------------------------------------+
| path                                                             |
+------------------------------------------------------------------+
| ("player102")-[:follow@0]->("player101")-[:serve@0]->("team201") |
+------------------------------------------------------------------+
```

```
nebula> FIND SHORTEST PATH FROM "team200" TO "player100" OVER * REVERSELY;
+---------------------------------------+
| path                                  |
+---------------------------------------+
| ("team200")<-[:serve@0]-("player100") |
+---------------------------------------+
```

```
nebula> FIND ALL PATH FROM "player100" TO "team200" OVER *;
+---------------------------------------+
| path                                  |
+---------------------------------------+
| ("player100")-[:serve@0]->("team200") |
+---------------------------------------+
```

```
nebula> FIND NOLOOP PATH FROM "player100" TO "team200" OVER *;
+---------------------------------------+
| path                                  |
+---------------------------------------+
| ("player100")-[:serve@0]->("team200") |
+---------------------------------------+
```

Last update: April 1, 2021

4.16 Query tuning statements

4.16.1 EXPLAIN and PROFILE

`EXPLAIN` helps output the execution plan of an nGQL statement without executing the statement. `PROFILE` executes the statement, then outputs the execution plan as well as the execution profile. You can optimize the queries for better performance with the execution plan and profile.

Execution Plan

The execution plan is determined by the execution planner in the Nebula Graph query engine.

The execution planner processes the parsed nGQL statements into actions. An action is the smallest unit that can be executed. A typical action fetches all neighbors of a given vertex, gets the properties of an edge, or filters vertices or edges based on the given conditions. Each action is assigned to an operator that performs the action.

For example, a `SHOW TAGS` statement is processed into two actions and assigned to a `Start` operator and a `ShowTags` operator, while a more complex `GO` statement may be processed into more than 10 actions and assigned to 10 operators.

Syntax

- `EXPLAIN`

```
EXPLAIN [format="row" | "dot"] <your_nGQL_statement>
```

- `PROFILE`

```
PROFILE [format="row" | "dot"] <your_nGQL_statement>
```

Output formats

The output of an `EXPLAIN` or a `PROFILE` statement has two formats, the default "row" format and the "dot" format. You can use the `format` option to modify the output format. Omitting the `format` option indicates using the default "row" format.

Format "row"

The "row" format outputs the return message in a table as follows.

- `EXPLAIN`:

```
nebula> EXPLAIN format="row" SHOW TAGS;
Execution succeeded (time spent 104/705 us)
Execution Plan
+----+----------+--------------+----------------+-------------------------------------------------------------------------+
| id | name     | dependencies | profiling data | operator info                                                           |
+----+----------+--------------+----------------+-------------------------------------------------------------------------+
|  0 | ShowTags | 2            |                | outputVar: [  {"colNames":[],"name":"__ShowTags_0","type":"DATASET"}] |
|    |          |              |                | inputVar:                                                               |
+----+----------+--------------+----------------+-------------------------------------------------------------------------+
|  2 | Start    |              |                | outputVar: [  {"colNames":[],"name":"__Start_2","type":"DATASET"}]    |
+----+----------+--------------+----------------+-------------------------------------------------------------------------+
```

- `PROFILE`:

```
nebula> PROFILE format="row" SHOW TAGS;
+--------+
| Name   |
+--------+
| player |
+--------+
| team   |
+--------+
Got 2 rows (time spent 2038/2728 us)

Execution Plan

+----+----------+--------------+-------------------------------------------------+-------------------------------------------------------------------------+
| id | name     | dependencies | profiling data                                  | operator info                                                           |
+----+----------+--------------+-------------------------------------------------+-------------------------------------------------------------------------+
|  0 | ShowTags | 2            | ver: 0, rows: 1, execTime: 79us, totalTime: 1692us | outputVar: [{"colNames":[],"name":"__ShowTags_0","type":"DATASET"}] |
|    |          |              |                                                 | inputVar:                                                               |
+----+----------+--------------+-------------------------------------------------+-------------------------------------------------------------------------+
|  2 | Start    |              | ver: 0, rows: 0, execTime: 1us, totalTime: 57us | outputVar: [{"colNames":[],"name":"__Start_2","type":"DATASET"}]    |
+----+----------+--------------+-------------------------------------------------+-------------------------------------------------------------------------+
```

The descriptions of the columns are as follows:

Column	Description
`id`	Indicates the ID of the operator.
`name`	Indicates the name of the operator.
`dependencies`	Shows the ID of the operator that the current operator depends on.
`profiling data`	Shows the execution profile. `ver` is the version of the operator, which you can use to identify loops; `rows` shows the number of rows to be output by the operator; `execTime` shows the execution time only; `totalTime` contains the execution time and the system scheduling and queueing time.
`operator info`	Shows the detailed information of the operator.

Format "dot"

You can use the `format="dot"` option to output the return message in the DOT language, and then use Graphviz to generate a graph of the plan.

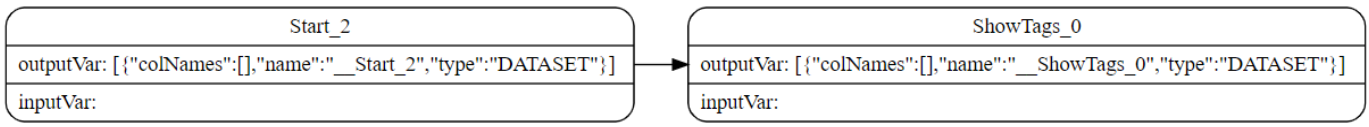
> **Note**
>
> Graphviz is open source graph visualization software.

Graphviz provides an online tool for previewing DOT language files and exporting them to other formats such as SVG or JSON. For more information, see Graphviz Online.

```
nebula> EXPLAIN format="dot" SHOW TAGS;
Execution succeeded (time spent 161/665 us)
Execution Plan
--------------------------------------------------------------------------------------------------------------------------------------------  -------------
  plan
```

```
---------------------------------------------------------------------------------------------------------------------------------------------------------------  -------------
  digraph exec_plan {
      rankdir=LR;
      "ShowTags_0"[label="ShowTags_0|outputVar: \[\{\"colNames\":\[\],\"name\":\"__ShowTags_0\",\"type\":\"DATASET\"\}\]\l|inputVar:\l",   shape=Mrecord];
      "Start_2"->"ShowTags_0";
      "Start_2"[label="Start_2|outputVar: \[\{\"colNames\":\[\],\"name\":\"__Start_2\",\"type\":\"DATASET\"\}\]\l|inputVar: \l",   shape=Mrecord];
  }
---------------------------------------------------------------------------------------------------------------------------------------------------------------  -------------
```

Transformed into a Graphviz graph, it is as follows:



Last update: April 22, 2021

4.17 Operation and maintenance statements

4.17.1 BALANCE syntax

The `BALANCE` statements support the load balancing operations of the Nebula Graph Storage services. For more information about storage load balancing and examples for using the `BALANCE` statements, see Storage load balance.

The `BALANCE` statements are listed as follows.

Syntax	Description
`BALANCE DATA`	Starts a task to balance the distribution of storage partitions in a Nebula Graph cluster.
`BALANCE DATA <balance_id>`	Shows the status of the balance task.
`BALANCE DATA STOP`	Stops the `BALANCE DATA` task.
`BALANCE DATA REMOVE <host_list>`	Scales in the Nebula Graph cluster and detaches specific storage hosts.
`BALANCE LEADER`	Balances the distribution of storage raft leaders in a Nebula Graph cluster.

Last update: March 5, 2021

4.17.2 Job manager and the JOB statements

The long-term tasks running by the Storage Service are called jobs. For example, there are jobs for `COMPACT`, `FLUSH`, and `STATS`. These jobs can be time-consuming if the data size in the graph space is large. The job manager helps you run, show, stop, and recover the jobs.

SUBMIT JOB COMPACT

The `SUBMIT JOB COMPACT` statement triggers the long-term RocksDB compact operation.

```
nebula> SUBMIT JOB COMPACT;
+------------+
| New Job Id |
+------------+
| 40         |
+------------+
```

For more information about compact configuration, see Storage Service configuration.

SUBMIT JOB FLUSH

The `SUBMIT JOB FLUSH` statement writes the RocksDB memfile in memory to the hard disk.

```
nebula> SUBMIT JOB FLUSH;
+------------+
| New Job Id |
+------------+
| 96         |
+------------+
```

SUBMIT JOB STATS

The `SUBMIT JOB STATS` statement starts a job that makes the statistics of the current graph space. Once this job succeeds, you can use the `SHOW STATS` statement to list the statistics. For more information, see SHOW STATS.

> **Note**
>
> If the data stored in the graph space changes, in order to get the latest statistics, you have to run `SUBMIT JOB STATS` again.

```
nebula> SUBMIT JOB STATS;
+------------+
| New Job Id |
+------------+
| 97         |
+------------+
```

SHOW JOB

The Meta Service parses a `SUBMIT JOB` request into tasks and assigns them to the nebula-storaged processes. The `SHOW JOB <job_id>` statement shows the information about a specific job and all its tasks.

The job ID is created when you run the `SUBMIT JOB` statement.

```
nebula> SHOW JOB 96;
+---------------+---------------+------------+-------------------------+-------------------------+
| Job Id(TaskId) | Command(Dest) | Status     | Start Time              | Stop Time               |
+---------------+---------------+------------+-------------------------+-------------------------+
| 96             | "FLUSH"       | "FINISHED" | 2020-11-28T14:14:29.000 | 2020-11-28T14:14:29.000 |
+---------------+---------------+------------+-------------------------+-------------------------+
| 0              | "storaged2"   | "FINISHED" | 2020-11-28T14:14:29.000 | 2020-11-28T14:14:29.000 |
+---------------+---------------+------------+-------------------------+-------------------------+
| 1              | "storaged0"   | "FINISHED" | 2020-11-28T14:14:29.000 | 2020-11-28T14:14:29.000 |
+---------------+---------------+------------+-------------------------+-------------------------+
| 2              | "storaged1"   | "FINISHED" | 2020-11-28T14:14:29.000 | 2020-11-28T14:14:29.000 |
+---------------+---------------+------------+-------------------------+-------------------------+
```

The description of the return message is as follows.

Column	Description
`Job Id(TaskId)`	The first row shows the job ID, and the other rows show the task IDs.
`Command(Dest)`	The first row shows the command executed, and the other rows show on which storaged processes the task is running.
`Status`	Shows the status of the job or task. For more information about job status, see Job status.
`Start Time`	Shows a timestamp indicating the time when the job or task enters the `RUNNING` phase.
`Stop Time`	Shows a timestamp indicating the time when the job or task gets `FINISHED`, `FAILED`, or `STOPPED`.

JOB STATUS

The description of the job status is as follows.

Status	Description
QUEUE	The job or task is waiting in a queue. The `Start Time` is empty in this phase.
RUNNING	The job or task is running. The `Start Time` shows the beginning of this phase.
FINISHED	The job or task is successfully finished. The `Stop Time` shows the time when the job or task enters this phase.
FAILED	The job or task failed. The `Stop Time` shows the time when the job or task enters this phase.
STOPPED	The job or task is stopped without running. The `Stop Time` shows the time when the job or task enters this phase.
REMOVED	The job or task is removed.

Status switching is described as follows.

```
Queue -- running -- finished -- removed
     \          \               /
      \          \ -- failed -- /
       \          \            /
        \ ---------- stopped -/
```

SHOW JOBS

The `SHOW JOBS` statement lists all the unexpired jobs. The default job expiration interval is one week. You can change it by modifying the `job_expired_secs` parameter of the Meta Service. For how to modify `job_expired_secs`, see Meta Service configuration.

```
nebula> SHOW JOBS;
+--------+---------------------+------------+-------------------------+-------------------------+
| Job Id | Command             | Status     | Start Time              | Stop Time               |
+--------+---------------------+------------+-------------------------+-------------------------+
| 97     | "STATS"             | "FINISHED" | 2020-11-28T14:48:52.000 | 2020-11-28T14:48:52.000 |
+--------+---------------------+------------+-------------------------+-------------------------+
| 96     | "FLUSH"             | "FINISHED" | 2020-11-28T14:14:29.000 | 2020-11-28T14:14:29.000 |
+--------+---------------------+------------+-------------------------+-------------------------+
| 95     | "STATS"             | "FINISHED" | 2020-11-28T13:02:11.000 | 2020-11-28T13:02:11.000 |
+--------+---------------------+------------+-------------------------+-------------------------+
| 86     | "REBUILD_EDGE_INDEX" | "FINISHED" | 2020-11-26T13:38:24.000 | 2020-11-26T13:38:24.000 |
+--------+---------------------+------------+-------------------------+-------------------------+
```

STOP JOB

The `STOP JOB` statement stops jobs that are not finished.

```
nebula> STOP JOB 22;
+---------------+
| Result        |
+---------------+
| "Job stopped" |
+---------------+
```

RECOVER JOB

The `RECOVER JOB` statement re-executes the failed jobs and returns the number of recovered jobs.

```
nebula> RECOVER JOB;
+-------------------+
| Recovered job num |
+-------------------+
| 5 job recovered   |
+-------------------+
```

FAQ

HOW TO TROUBLESHOOT JOB PROBLEMS

The `SUBMIT JOB` operations use the HTTP port. Please check if the HTTP ports on the machines where the Storage Service is running are working well. You can use the following command to debug.

```
curl "http://{storaged-ip}:19779/admin?space={space_name}&op=compact"
```

Last update: April 22, 2021

4.18 Appendix

4.18.1 Comments

Legacy version compatibility

- In Nebula Graph 1.0, four comment styles: `#`, `--`, `//`, `/* */`.
- In Nebula Graph 2.0, `--` represents an edge, and can not be used as comments.

Examples

```
nebula> # Do nothing this line
nebula> RETURN 1+1;     # This comment continues to the end of line
nebula> RETURN 1+1;     // This comment continues to the end of line
nebula> RETURN 1 /* This is an in-line comment */ + 1 == 2;
nebula> RETURN 11 +            \
/* Multiple-line comment       \
Use backslash as line break.   \
*/ 12;
```

The backslash `\` in a line indicates a line break.

OpenCypher Compatibility

You must add a `\` at the end of every line, even in multi-line comments `\* *\`.

```
/* The openCypher style:
The following comment
spans more than
one line */
MATCH (n:label)
RETURN n
```

```
/* The ngql style:    \
The following comment \
spans more than       \
one line */           \
MATCH (n:tag)             \
RETURN n
```

Last update: March 29, 2021

4.18.2 Identifer Case Sensitivity

Identifiers are Case-Sensitive

The following statements would not work because they refer to two different spaces, i.e. `my_space` and `MY_SPACE`:

```
nebula> CREATE SPACE my_space;
nebula> use MY_SPACE;
[ERROR (-8)]: SpaceNotFound:
# my_space and MY_SPACE are two different spaces
```

Keywords and Reserved Words are Case-Insensitive

The following statements are equivalent:

```
nebula> show spaces;  # show and spaces are keywords.
nebula> SHOW SPACES;
nebula> SHOW spaces;
nebula> show SPACES;
```

Last update: March 29, 2021

4.18.3 Keywords and Reserved Words

Keywords have significance in nGQL. Certain keywords are reserved and require special treatment for use as identifiers.

Non-reserved keywords are permitted as identifiers without quoting. Non-reserved keywords are case-insensitive. To use reserved keywords as identifiers, quote them with back quotes such as `AND`.

```
nebula> CREATE TAG TAG(name string);
[ERROR (-7)]: SyntaxError: syntax error near `TAG'

// SPACE is an unreserved keyword.
nebula> CREATE TAG SPACE(name string);
Execution succeeded
```

`TAG` is a reserved keyword. To use `TAG` as an identifier, you must quote it with a backtick. `SPACE` is a non-reserved keyword. You can use `SPACE` as an identifier without quoting it.

> **Note**
>
> There is a small pitfall when you use the non-reserved keyword. Unquoted non-reserved keyword will be converted to **lower-case** words. For example, `SPACE` or `Space` will become `space`.

```
// TAG is a reserved keyword here.
nebula> CREATE TAG `TAG` (name string);
Execution succeeded
```

Reserved Words

The following list shows reserved words in nGQL.

```
ADD
ALTER
AND
AS
ASC
BALANCE
BOOL
BY
CASE
CHANGE
COMPACT
CREATE
DATE
DATETIME
DELETE
DESC
DESCRIBE
DISTINCT
DOUBLE
DOWNLOAD
DROP
EDGE
EDGES
EXISTS
EXPLAIN
FETCH
FIND
FIXED_STRING
FLOAT
FLUSH
FORMAT
FROM
GET
GO
GRANT
IF
IN
INDEX
INDEXES
INGEST
INSERT
INT
INT16
INT32
INT64
INT8
INTERSECT
```

```
IS
LIMIT
LOOKUP
MATCH
MINUS
NO
NOT
NULL
OF
OFFSET
ON
OR
ORDER
OVER
OVERWRITE
PROFILE
PROP
REBUILD
RECOVER
REMOVE
RETURN
REVERSELY
REVOKE
SET
SHOW
STEP
STEPS
STOP
STRING
SUBMIT
TAG
TAGS
TIME
TIMESTAMP
TO
UNION
UPDATE
UPSERT
UPTO
USE
VERTEX
WHEN
WHERE
WITH
XOR
YIELD
```

Non-Reserved Keywords

```
ACCOUNT
ADMIN
ALL
ANY
ATOMIC_EDGE
AUTO
AVG
BIDIRECT
BIT_AND
BIT_OR
BIT_XOR
BOTH
CHARSET
CLIENTS
COLLATE
COLLATION
COLLECT
COLLECT_SET
CONFIGS
CONTAINS
COUNT
COUNT_DISTINCT
DATA
DBA
DEFAULT
ELASTICSEARCH
ELSE
END
ENDS
FALSE
FORCE
FUZZY
GOD
GRAPH
GROUP
GROUPS
GUEST
HDFS
HOST
HOSTS
INTO
JOB
```

```
JOBS
LEADER
LISTENER
MAX
META
MIN
NOLOOP
NONE
OPTIONAL
OUT
PART
PARTITION_NUM
PARTS
PASSWORD
PATH
PLAN
PREFIX
REGEXP
REPLICA_FACTOR
RESET
ROLE
ROLES
SEARCH
SERVICE
SHORTEST
SIGN
SINGLE
SKIP
SNAPSHOT
SNAPSHOTS
SPACE
SPACES
STARTS
STATS
STATUS
STD
STORAGE
SUBGRAPH
SUM
TEXT
TEXT_SEARCH
THEN
TRUE
TTL_COL
TTL_DURATION
UNWIND
USER
USERS
UUID
VALUE
VALUES
VID_TYPE
WILDCARD
ZONE
ZONES
```

Last update: April 22, 2021

4.18.4 Vertex identifier and partition ID

VID

`VID` is short for vertex identifier.

In Nebula Graph, vertices are identified with vertex identifiers (i.e. `VID`s). The VID can be an int64 or a fixed length string. When inserting a vertex, you must specify a `VID` for it.

You can also call `hash()` to generate an int64 VID if the graph has less than one billion vertices.

> `VID` must be unique in a graph space.

That is, in the same graph space, two vertices that have the same `VID` are considered as the same vertex.

In addition, one `VID` can have multiple `TAG`s. E.g., One person (`VID`) can have two roles (`tags`).

Two `VID`s in two different graph spaces are totally independent of each other.

Partition ID

When inserting into Nebula Graph, vertices and edges are distributed across different partitions. And the partitions are located on different machines. If you want certain vertices to locate on the same partition (i.e., on the same machine), you can control the generation of the `VID`s by using the following formula / code.

```
// If the length of the id is 8, we will treat it as int64_t to be compatible
// with the version 1.0
uint64_t vid = 0;
if (id.size() == 8) {
    memcpy(static_cast<void*>(&vid), id.data(), 8);
} else {
    MurmurHash2 hash;
    vid = hash(id.data());
}
PartitionID pId = vid % numParts + 1;
```

Roughly say, after hashing a fixed string to int64, (the hashing of int64 is the number itself), do modulo and then plus one.

```
pId = vid % numParts + 1;
```

In the preceding formula,

- `%` is the modulo operation.
- `numParts` is the number of partition for the graph space where the `VID` is located, namely the value of `partition_num` in the CREATE SPACE statement.
- `pId` is the ID for the partition where the `VID` is located.

For example, if there are 100 partitions, the vertices with `VID` 1, 101, 1001 will be stored on the same partition.

But, the mapping between the `partition ID` and the machine address is random. Therefore, you can't assume that any two partitions are located on the same machine.

Last update: March 29, 2021

5. Deployment and installation

5.1 Prepare resources for compiling, installing, and running Nebula Graph

This topic describes the requirements and suggestions for compiling and installing Nebula Graph, as well as how to estimate the resource you need to reserve for running a Nebula Graph cluster.

5.1.1 Reading guide

If you are reading this topic with the questions listed below, click them to jump to their answers.

- What do I need to compile Nebula Graph?
- What do I need to run Nebula Graph in a test environment?
- What do I need to run Nebula Graph in a production environment?
- How much memory and disk space do I need to reserve for my Nebula Graph cluster?

5.1.2 Requirements for compiling the Nebula Graph source code

Hardware requirements for compiling Nebula Graph

Item	Requirement
CPU architecture	x86_64
Memory	4 GB
Disk	10 GB, SSD

Supported operating systems for compiling Nebula Graph

For now, we can only compile Nebula Graph in the Linux system. We recommend that you use any Linux system with kernel version 2.6.32 or above.

Software requirements for compiling Nebula Graph

You must have the correct version of the software listed below to compile Nebula Graph. If they are not as required or you are not sure, follow the steps in Prepare software for compiling Nebula Graph to get them ready.

Software	Version	Note
glibc	2.12 or above	You can run `ldd --version` to check the glibc version.
make	Any stable version	-
m4	Any stable version	-
git	Any stable version	-
wget	Any stable version	-
unzip	Any stable version	-
xz	Any stable version	-
readline-devel	Any stable version	-
ncurses-devel	Any stable version	-
zlid-devel	Any stable version	-
gcc	7.1.0 or above	You can run `gcc -v` to check the gcc version.
gcc-c++	Any stable version	-
cmake	3.5.0 or above	You can run `cmake --version` to check the cmake version.
gettext	Any stable version	-
curl	Any stable version	-
redhat-lsb-core	Any stable version	-
libstdc++-static	Any stable version	Only needed in CentOS 8+, RedHat 8+, and Fedora systems.
libasan	Any stable version	Only needed in CentOS 8+, RedHat 8+, and Fedora systems.

Other third-party software will be automatically downloaded and installed to the build directory at the configure (cmake) stage.

Prepare software for compiling Nebula Graph

This section guides you through the downloading and installation of software required for compiling Nebula Graph.

1. Install dependencies.

   - For CentOS, RedHat, and Fedora users, run the following commands.

   ```bash
   $ yum update
   $ yum install -y make \
                    m4 \
                    git \
                    wget \
                    unzip \
                    xz \
                    readline-devel \
                    ncurses-devel \
                    zlib-devel \
                    gcc \
                    gcc-c++ \
                    cmake \
                    gettext \
                    curl \
                    redhat-lsb-core
   // For CentOS 8+, RedHat 8+, and Fedora, install libstdc++-static, libasan as well
   $ yum install -y libstdc++-static libasan
   ```

   - For Debian and Ubuntu users, run the following commands.

   ```bash
   $ apt-get update
   $ apt-get install -y make \
                        m4 \
                        git \
                        wget \
                        unzip \
                        xz-utils \
                        curl \
                        lsb-core \
                        build-essential \
                        libreadline-dev \
                        ncurses-dev \
                        cmake \
                        gettext
   ```

2. Check if the GCC and cmake on your host are in the right version. See Software requirements for compiling Nebula Graph for the required versions.

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

   If your GCC and CMake are in the right version, then you are all set. If they are not, follow the sub-steps as follows.

   1. Clone the nebula-common repository to your host.

      ```bash
      $ git clone https://github.com/vesoft-inc/nebula-common.git
      ```

      The source code of Nebula Graph versions such as v2.0.0 is stored in particular branches. You can use the `--branch` or `-b` option to specify the branch to be cloned. For example, for 2.0.0, run the following command.

      ```bash
      $ git clone --branch v2.0.0 https://github.com/vesoft-inc/nebula-common.git
      ```

   2. Make `nebula-common` the current working directory.

      ```bash
      $ cd nebula-common
      ```

   3. Run the following commands to install and enable CMake and GCC.

      ```bash
      // Install CMake.
      $ ./third-party/install-cmake.sh cmake-install

      // Enable CMake
      $ source cmake-install/bin/enable-cmake.sh

      // Install GCC. Installing GCC to /opt requires root privilege, you can change it to other locations.
      $ sudo ./third-party/install-gcc.sh --prefix=/opt

      // Enable GCC.
      ```

```
$ source /opt/vesoft/toolset/gcc/7.5.0/enable
```

5.1.3 Requirements and suggestions for installing Nebula Graph in test environments

Hardware requirements for test environments

Item	Requirement
CPU architecture	x86_64
Number of CPU core	4
Memory	8 GB
Disk	100 GB, SSD

Supported operating systems for test environments

For now, we can only install Nebula Graph in the Linux system. To install Nebula Graph in a test environment, we recommend that you use any Linux system with kernel version 3.9 or above.

Suggested service architecture for test environments

Process	Suggested number
metad (the metadata service process)	1
storaged (the storage service process)	1 or more
graphd (the query engine service process)	1 or more

For example, for a single-machine environment, you can deploy 1 metad, 1 storaged, and 1 graphd processes in the machine.

For a more common environment, such as a cluster of 3 machines (named as A, B, and C), you can deploy Nebula Graph as follows:

Machine name	Number of metad	Number of storaged	Number of graphd
A	1	1	1
B	None	1	1
C	None	1	1

5.1.4 Requirements and suggestions for installing Nebula Graph in production environments

Hardware requirements for production environments

Item	Requirement
CPU architecture	x86_64
Number of CPU core	48
Memory	96 GB
Disk	2 * 900 GB, NVMe SSD

Supported operating systems for production environments

For now, we can only install Nebula Graph in the Linux system. To install Nebula Graph in a production environment, we recommend that you use any Linux system with kernel version 3.9 or above.

You can adjust some of the kernel parameters to better accommodate the need for running Nebula Graph. For more information, see kernel configuration.

Suggested service architecture for production environments

Process	Suggested number
metad (the metadata service process)	3
storaged (the storage service process)	3 or more
graphd (the query engine service process)	3 or more

Each metad process automatically creates and maintains a copy of the metadata. Usually, you only need 3 metad processes. The number of storaged processes does not affect the number of graph space copies.

You can deploy multiple processes on a single machine. For example, on a cluster of 5 machines (named as A, B, C, D, and E), you can deploy Nebula Graph as follows:

> **Caution**
>
> Deploying a cluster across IDCs is not supported yet.

Machine name	Number of metad	Number of storaged	Number of graphd
A	1	1	1
B	1	1	1
C	1	1	1
D	None	1	1
E	None	1	1

5.1.5 Capacity requirements for running a Nebula Graph cluster

You can estimate the memory, disk space, and partition number needed for a Nebula Graph cluster of 3 replicas as follows.

Resource	Unit	How to estimate	Description
Disk space for a cluster	Bytes	`the_sum_of_edge_number_and_vertex_number` * `average_bytes_of_attributes` * 6 * 120%	-
Memory for a cluster	Bytes	[`the_sum_of_edge_number_and_vertex_number` * 15 + `the_number_of_RocksDB_instances` * (`write_buffer_size` * `max_write_buffer_number`) + `rocksdb_block_cache`] * 120%	`write_buffer_size` and `max_write_buffer_number` are RocksDB parameters, for more information, see MemTable. For details about `rocksdb_block_cache`, see Memory usage in RocksDB.
Number of partitions for a graph space	-	`the_number_of_disks_in_the_cluster` * `disk_partition_num_multiplier`	`disk_partition_num_multiplier` is an integer between 2 and 10 (both including). It's value depends on the disk performance. Use 2 for HDD.

- Question 1: Why do we multiply the disk space and memory by 120%?

  Answer: The extra 20% is for buffer.

- Question 2: How to get the number of RocksDB instances?

  Answer: Each directory in the `--data_path` item in the `etc/nebula-storaged.conf` file corresponds to a RocksDB instance. Count the number of directories to get the RocksDB instance number.

> **Note**
>
> You can decrease the memory size occupied by the bloom filter by adding `--enable_partitioned_index_filter=true` in `etc/nebula-storaged.conf`. But it may decrease the read performance in some random-seek cases.

5.1.6 About storage devices

Nebula Graph is designed and implemented for NVMe SSD. All default parameters are optimized for the SSD devices.

Due to the poor IOPS capability and long random seek latency, HDD is not recommended. You may encounter many problems when using HDD.

And remote storage devices, such as NAS or SAN, are not recommended/tested as well.

Use local SSD device.

Last update: April 22, 2021

5.2 Compile and install Nebula Graph

5.2.1 Install Nebula Graph by compiling the source code

Installing Nebula Graph from the source code allows you to customize the compiling and installation settings and test the latest features.

Prerequisites

- You have prepared the necessary resources described in Prepare resources for compiling, installing, and running Nebula Graph.
- You can access the Internet from the host you plan to install Nebula Graph.
- The console is not complied or packaged with Nebula Graph server binaries. You can install nebula-console by yourself.

How to install

1. Use Git to clone the source code of Nebula Graph to your host.
   - To install the latest developing version, run the following command to download the source code from the `master` branch.

     ```
     $ git clone https://github.com/vesoft-inc/nebula-graph.git
     ```

   - To install a specific release version, use the `--branch` option to specify the correct branch. For example, to install 2.0.0, run the following command.

     ```
     $ git clone --branch v2.0.0 https://github.com/vesoft-inc/nebula-graph.git
     ```

2. Make the `nebula-graph` directory the current working directory.

   ```
   $ cd nebula-graph
   ```

3. Create a `build` directory and make it the current working directory.

   ```
   $ mkdir build && cd build
   ```

4. Generate the Makefile with CMake.

   > **Note**
   >
   > - The installation path is `/usr/local/nebula` by default. To customize it, add the `-DCMAKE_INSTALL_PREFIX=/your/install/path/` CMake variable in the following command.
   > - For more information about CMake variables, see CMake variables.

   - If you are installing the latest developing version and has cloned the `master` branch in step 1, run the following command.

     ```
     $ cmake -DENABLE_BUILD_STORAGE=on -DENABLE_TESTING=OFF -DCMAKE_BUILD_TYPE=Release ..
     ```

   - If you are installing a specific release version and has cloned the corresponding branch in step 1, use the `-DNEBULA_COMMON_REPO_TAG` and `-DNEBULA_STORAGE_REPO_TAG` options to specify the correct branches of the nebula-common and nebula-storage repositories. For example, to install release version 2.0.0, run the following command.

     ```
     $ cmake -DENABLE_BUILD_STORAGE=on -DENABLE_TESTING=OFF -DCMAKE_BUILD_TYPE=Release \
     -DNEBULA_COMMON_REPO_TAG=v2.0.0 -DNEBULA_STORAGE_REPO_TAG=v2.0.0 ..
     ```

5. Compile Nebula Graph.

   To speed up the compiling, use the `-j` option to set a concurrent number `N`. It should be `min(MEM/2, CPU)`, where `MEM` is the memory size in GB, and `CPU` is the core number.

```
$ make -j{N} # E.g., make -j4
```

This step will take about 20 minutes on a VM with four cores of `Intel(R) Xeon(R) Platinum 8260M CPU @ 2.30GHz`.

6. Install Nebula Graph.

```
$ sudo make install-all
```

7. [Optional] Update the source code of the `master` branch. (It changes frequently.)
   1. In the `nebula-graph/` directory, you can use `git pull upstream master` to update the source code.
   2. In `nebula-graph/modules/common/` and `nebula-graph/modules/storage/`, run `git pull upstream master` separately.
   3. In `nebula-graph/build/`, `make` and `make install[-all]` again.

CMake variables

Usage of CMake variables:

```
$ cmake -D<variable>=<value> ...
```

The following CMake variables can be used at the configure (cmake) stage to adjust the compiling settings.

ENABLE_BUILD_STORAGE

Starting from the 2.0 pre-release, Nebula Graph uses two separated github repositories of compute and storage. The `ENABLE_BUILD_STORAGE` variable is set to `OFF` by default so that the storage service is not installed together with the graph service.

If you are deploying Nebula Graph on a single host for testing, you can set `ENABLE_BUILD_STORAGE` to `ON` to download and install the storage service automatically.

CMAKE_INSTALL_PREFIX

`CMAKE_INSTALL_PREFIX` specifies the path where the service modules, scripts, configuration files are installed. The default path is `/usr/local/nebula`.

ENABLE_WERROR

`ENABLE_WERROR` is `ON` by default and it makes all warnings into errors. You can set it to `OFF` if needed.

ENABLE_TESTING

`ENABLE_TESTING` is `ON` by default and unit tests are built with the Nebula Graph services. If you just need the service modules, set it to `OFF`.

ENABLE_ASAN

`ENABLE_ASAN` is `OFF` by default and the building of ASan (AddressSanitizer), a memory error detector, is disabled. To enable it, set `ENABLE_ASAN` to `ON`. This variable is intended for Nebula Graph developers.

CMAKE_BUILD_TYPE

Nebula Graph supports the following building types:

- `Debug`, the default value of `CMAKE_BUILD_TYPE`, indicates building Nebula Graph with the debug info but not the optimization options.
- `Release`, indicates building Nebula Graph with the optimization options but not the debug info.
- `RelWithDebInfo`, indicates building Nebula Graph with the optimization options and the debug info.
- `MinSizeRel`, indicates building Nebula Graph with the optimization options for controlling the code size but not the debug info.

CMAKE_C_COMPILER/CMAKE_CXX_COMPILER

Usually, CMake locates and uses a C/C++ compiler installed in the host automatically. But if your compiler is not installed at the standard path, or if you want to use a different one, run the command as follows to specify the installation path of the target compiler:

```
$ cmake -DCMAKE_C_COMPILER=<path_to_gcc/bin/gcc> -DCMAKE_CXX_COMPILER=<path_to_gcc/bin/g++> ..
$ cmake -DCMAKE_C_COMPILER=<path_to_clang/bin/clang> -DCMAKE_CXX_COMPILER=<path_to_clang/bin/clang++> ..
```

ENABLE_CCACHE

`ENABLE_CCACHE` is `ON` by default and ccache is used to speed up the compiling of Nebula Graph.

To disable `ccache`, set `ENABLE_CCACHE` to `OFF`. On some platforms, the `ccache` installation hooks up or precedes the compiler. In such a case, you have to set an environment variable `export CCACHE_DISABLE=true` or add a line `disable=true` in `~/.ccache/ccache.conf` as well.

For more information, see the ccache official documentation.

NEBULA_THIRDPARTY_ROOT

`NEBULA_THIRDPARTY_ROOT` specifies the path where the third party software is installed. By default it is `/opt/vesoft/third-party`.

Last update: May 28, 2021

5.2.2 Install Nebula Graph with RPM or DEB package

RPM and DEB are common package formats on Linux systems. This topic shows how to quickly install Nebula Graph with the RPM or DEB package.

Prerequisites

Prepare the right resources.

> **Note**
>
> The console is not complied or packaged with Nebula Graph server binaries. You can install nebula-console by yourself.

Download the package from cloud service

- Download the released version.

  URL

  ```
  //Centos 6
  https://oss-cdn.nebula-graph.io/package/<release_version>/nebula-graph-<release_version>.el6.x86_64.rpm

  //Centos 7
  https://oss-cdn.nebula-graph.io/package/<release_version>/nebula-graph-<release_version>.el7.x86_64.rpm

  //Centos 8
  https://oss-cdn.nebula-graph.io/package/<release_version>/nebula-graph-<release_version>.el8.x86_64.rpm

  //Ubuntu 1604
  https://oss-cdn.nebula-graph.io/package/<release_version>/nebula-graph-<release_version>.ubuntu1604.amd64.deb

  //Ubuntu 1804
  https://oss-cdn.nebula-graph.io/package/<release_version>/nebula-graph-<release_version>.ubuntu1804.amd64.deb

  //Ubuntu 2004
  https://oss-cdn.nebula-graph.io/package/<release_version>/nebula-graph-<release_version>.ubuntu2004.amd64.deb
  ```

  For example, download release package `2.0.0` for `Centos 7.5`

  ```
  wget https://oss-cdn.nebula-graph.io/package/2.0.0/nebula-graph-2.0.0.el7.x86_64.rpm
  wget https://oss-cdn.nebula-graph.io/package/2.0.0/nebula-graph-2.0.0.el7.x86_64.rpm.sha256sum.txt
  ```

  download release package `2.0.0` for `Ubuntu 1804`

```
wget https://oss-cdn.nebula-graph.io/package/2.0.0/nebula-graph-2.0.0.ubuntu1804.amd64.deb
wget https://oss-cdn.nebula-graph.io/package/2.0.0/nebula-graph-2.0.0.ubuntu1804.amd64.deb.sha256sum.txt
```

- Download the nightly version.

> **Danger**
>
> Nightly versions are usually used to test new features. Don't use it for production.

URL

```
//Centos 6
https://oss-cdn.nebula-graph.io/package/v2-nightly/<yyyy.mm.dd>/nebula-graph-<yyyy.mm.dd>-nightly.el6.x86_64.rpm

//Centos 7
https://oss-cdn.nebula-graph.io/package/v2-nightly/<yyyy.mm.dd>/nebula-graph-<yyyy.mm.dd>-nightly.el7.x86_64.rpm

//Centos 8
https://oss-cdn.nebula-graph.io/package/v2-nightly/<yyyy.mm.dd>/nebula-graph-<yyyy.mm.dd>-nightly.el8.x86_64.rpm

//Ubuntu 1604
https://oss-cdn.nebula-graph.io/package/v2-nightly/<yyyy.mm.dd>/nebula-graph-<yyyy.mm.dd>-nightly.ubuntu1604.amd64.deb

//Ubuntu 1804
https://oss-cdn.nebula-graph.io/package/v2-nightly/<yyyy.mm.dd>/nebula-graph-<yyyy.mm.dd>-nightly.ubuntu1804.amd64.deb

//Ubuntu 2004
https://oss-cdn.nebula-graph.io/package/v2-nightly/<yyyy.mm.dd>/nebula-graph-<yyyy.mm.dd>-nightly.ubuntu2004.amd64.deb
```

For example, download the `Centos 7.5` package developed and built in `2021.03.28`

```
wget https://oss-cdn.nebula-graph.io/package/v2-nightly/2021.03.28/nebula-graph-2021.03.28-nightly.el7.x86_64.rpm
wget https://oss-cdn.nebula-graph.io/package/v2-nightly/2021.03.28/nebula-graph-2021.03.28-nightly.el7.x86_64.rpm.sha256sum.txt
```

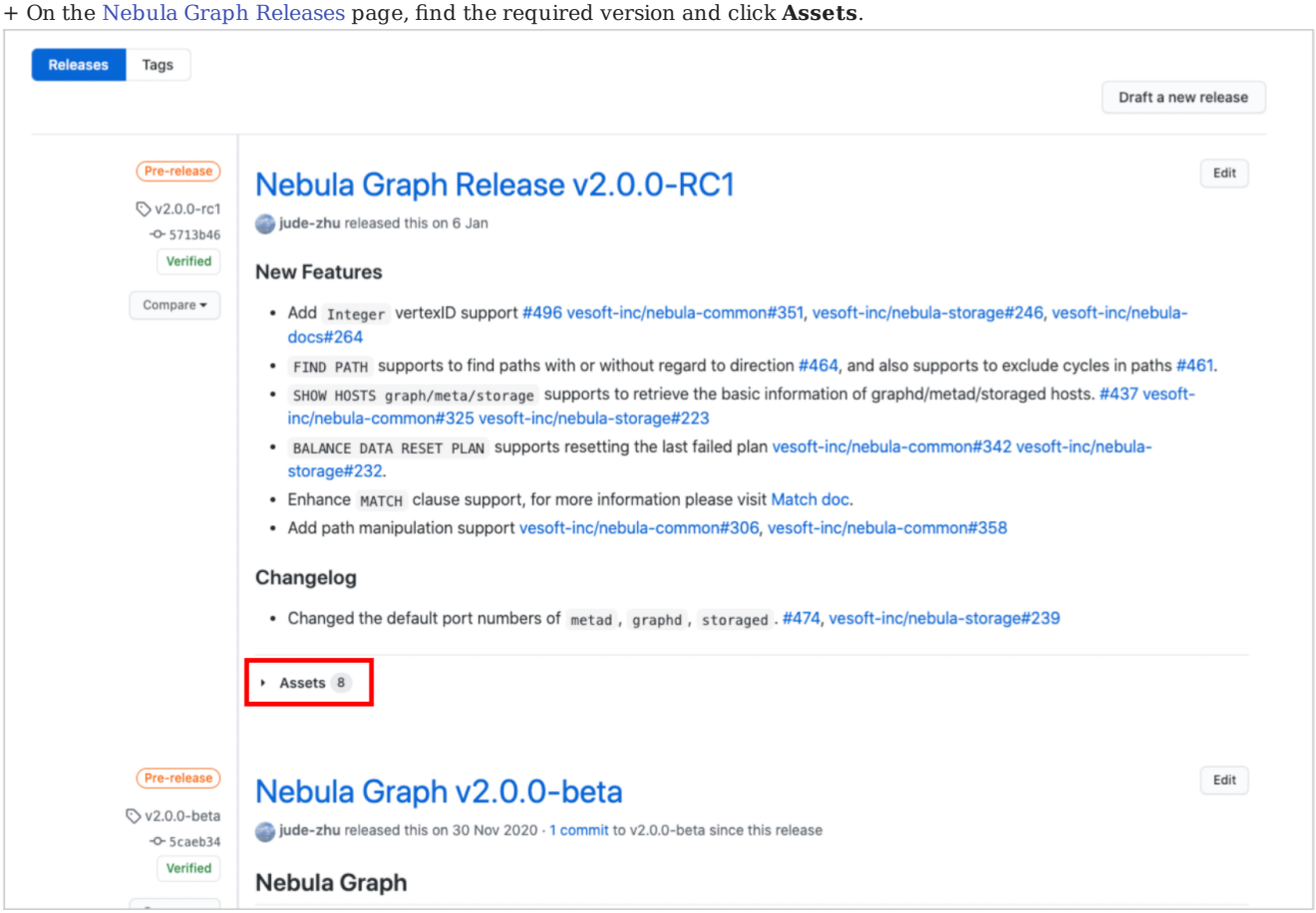
For example, download the `Ubuntu 1804` package developed and built in `2021.03.28`

```
wget https://oss-cdn.nebula-graph.io/package/v2-nightly/2021.03.28/nebula-graph-2021.03.28-nightly.ubuntu1804.amd64.deb
wget https://oss-cdn.nebula-graph.io/package/v2-nightly/2021.03.28/nebula-graph-2021.03.28-nightly.ubuntu1804.amd64.deb.sha256sum.txt
```

Download the package from GitHub

- Download the release version.

+ On the Nebula Graph Releases page, find the required version and click **Assets**.

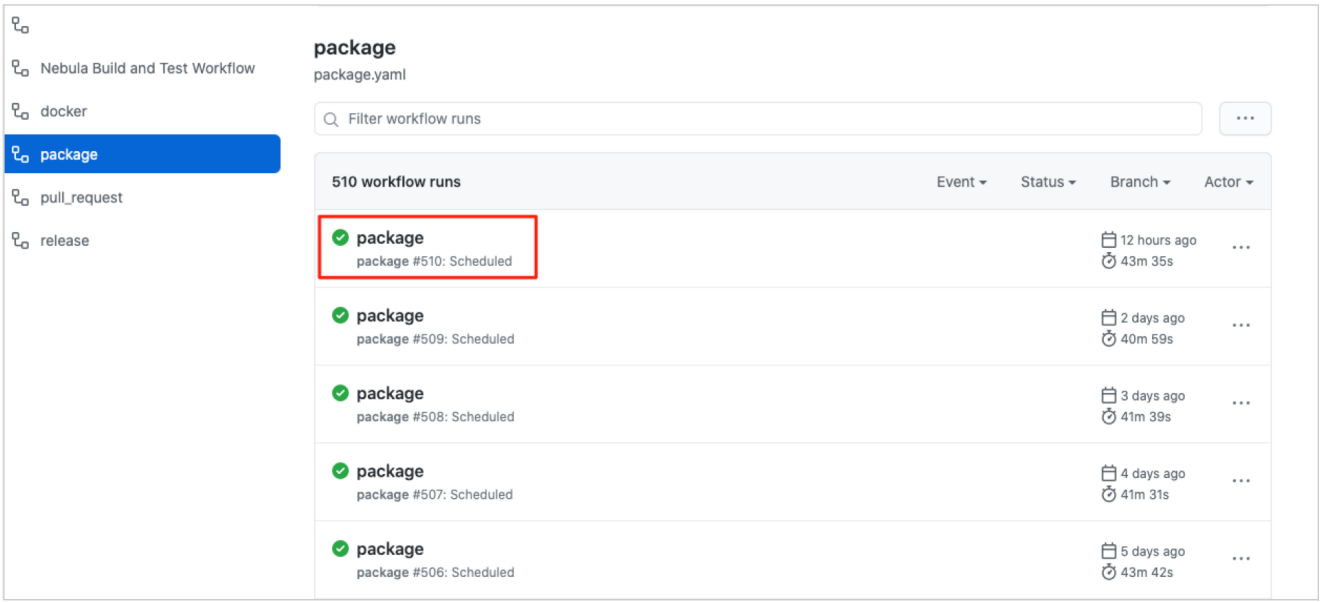


+ In the **Assets** area, click the package to download it.

- Download the nightly version.

> **Danger**
>
> Nightly versions are usually used to test new features. Don't use it for production.

+ On the Nebula Graph package page, click the latest package on the top of the package list.



+ In the **Artifacts** area, click the package to download it.

Install Nebula Graph

- Use the following syntax to install with an RPM package.

```
sudo rpm -ivh --prefix=<installation_path> <package_name>
```

- Use the following syntax to install with a DEB package.

```
sudo dpkg -i --instdir==<installation_path> <package_name>
```

> **Note**
>
> The default installation path is `/usr/local/nebula/`.

Last update: April 22, 2021

5.3 Deploy Nebula Graph cluster

This topic describes how to manually deploy a Nebula Graph cluster.

> **Note**
>
> For now, Nebula Graph does not have an official deployment tool.

5.3.1 Prerequisites

Prepare hardware for deploying the cluster.

5.3.2 Step 1: Install Nebula Graph

Install Nebula Graph on each machine in the cluster. Available approaches of installation are as follows.

- Install Nebula Graph with RPM or DEB package
- Install Nebula Graph by compiling the source code

5.3.3 Step 2: Modify the configurations

To deploy Nebula Graph according to your requirements, you have to modify the configuration files. All the configuration files for Nebula Graph, including `nebula-graphd.conf`, `nebula-metad.conf`, and `nebula-storaged.conf`, are stored in the `etc` directory in the installation path.

You only need to modify the configuration for the corresponding service on the machines. For example, modify `nebula-graphd.conf` on the machines where you want to deploy the Graph Service.

For how to prepare the configuration files, see:

- Meta Service configurations
- Graph Service configurations
- Storage Service configurations

5.3.4 Step 3: Start the cluster

Start the corresponding service on each machine. The command to start the Nebula Graph services is as follows.

```
sudo /usr/local/nebula/scripts/nebula.service start <metad|graphd|storaged|all>
```

`/usr/local/nebula` is the default installation path for Nebula Graph. Use the actual path if you have customized the path.

For more information about how to start and stop the services, see Manage Nebula Graph services.

5.3.5 Connect to the cluster

Connect to the Graph Service with a Nebula Graph client, such as Nebula Console. For more information, see Connect to Nebula Graph.

5.3.6 Check the cluster status

After connecting to the Nebula Graph cluster, run `SHOW HOSTS` to check the cluster status.

Last update: April 22, 2021

5.4 Upgrade Nebula Graph to v2.0.0

This topic describes how to upgrade Nebula Graph to v2.0.0.

5.4.1 Limitations

- Rolling Upgrade is not supported. You must stop the Nebula Graph services before the upgrade.
- There is no upgrade script. You have to manually upgrade each server in the cluster.
- Supported versions:
  - From Nebula Graph v1.2.0 to Nebula Graph v2.0.0.
  - From Nebula Graph v2.0.0-RC1 to Nebula Graph 2.0.0.
- This topic does not apply to scenarios where Nebula Graph is deployed with Docker, including Docker Swarm, Docker Compose, and Kubernetes.
- You must upgrade the old Nebula Graph services on the same machines they are deployed. **DO NOT** change the IP addresses, configuration files of the machines, and **DO NOT** change the cluster topology.
- The hard disk space of each machine should be three times as much as the space taken by the original data directories.
- Known issues that could cause data loss are listed on GitHub known issues. The issues are all related to altering schema or default values.
- To connect to Nebula Graph 2.0.0, you must upgrade all the Nebula Graph clients. The communication protocols of the old versions and the latest versions are not compatible.
- The upgrade takes about 30 minutes in this test environment.
- **DO NOT** use soft links to switch the data directories.
- You must have the sudo privileges to complete the steps in this topic.

5.4.2 Installation paths

Old installation path

By default, old versions of Nebula Graph are installed in `/usr/local/nebula/`, hereinafter referred to as `${nebula-old}`. The default configuration file path is `${nebula-old}/etc/`.

The data of the old Nebula Graph are stored by the Storage Service and the Meta Service. You can find the data paths as follows.

- Storage data path is defined by the `--data_path` option in the `${nebula-old}/etc/nebula-storaged.conf` file. The default path is `data/storage`.
- Meta data path is defined by the `--data_path` option in the `${nebula-old}/etc/nebula-metad.conf` file. The default path is `data/meta`.

> **Note**
>
> The actual paths in your environment may be different from those described in this topic. You can run the Linux command `ps -ef | grep nebula` to locate them.

New installation path

`${nebula-new}` represents the installation path of the new Nebula Graph version. An example for `${nebula-new}` is `/usr/local/nebula-new/`.

5.4.3 Steps

1. Stop all client connections. You can run the following commands on each Graph server to turn off the Graph Service and avoid dirty write.

```
> ${nebula-old}/scripts/nebula.service stop graphd
[INFO] Stopping nebula-graphd...
[INFO] Done
```

2. Run the following commands to stop all services of the old version Nebula Graph.

```
> ${nebula-old}/scripts/nebula.service stop all
[INFO] Stopping nebula-metad...
[INFO] Done
[INFO] Stopping nebula-graphd...
[INFO] Done
[INFO] Stopping nebula-storaged...
[INFO] Done
```

   The Storage Service needs about 1 minute to flush data. Wait 1 minute and then run `ps -ef | grep nebula` to check and make sure that all the Nebula Graph services are stopped.

> **Note**
>
> If the services are not fully stopped in 20 minutes, stop upgrading and go to the Nebula Graph community for help.

3. Install the new version of Nebula Graph on each machine.

   - To install with RPM/DEB packages, run the following command. For detailed steps, see Install Nebula Graph with RPM or DEB package.

     ```
     > sudo rpm --force -i --prefix=${nebula-new}  ${nebula-package-name.rpm} # for CentOS/RedHat
     > sudo dpkg -i --instdir==${nebula-new} ${nebula-package-name.deb} # for Ubuntu
     ```

   - To install with the source code, follow the substeps. For detailed steps, see Install Nebula Graph by compiling the source code

     1. Clone the source code.

        ```
        > git clone --branch v2.0.0 https://github.com/vesoft-inc/nebula-graph.git
        ```

     2. Configure CMake.

        ```
        > cmake -DCMAKE_INSTALL_PREFIX=${nebula-new} -DENABLE_BUILD_STORAGE=on -DENABLE_TESTING=OFF -DCMAKE_BUILD_TYPE=Release -DNEBULA_COMMON_REPO_TAG=v2.0.0  -
        DNEBULA_STORAGE_REPO_TAG=v2.0.0 ..
        ```

4. Copy the configuration files from the old path to the new path.

```
> cp -rf ${nebula-old}/etc ${nebula-new}/
```

5. Follow the substeps to prepare the Meta servers (usually 3 of them in a cluster).

> **Note**
>
> You must make sure that this step is applied on every Meta server.

   a. Locate the old Meta data path and copy the data files to the new path.

```
> mkdir -p ${nebula-new}/data/meta/
> cp -r ${nebula-old}/data/meta/* ${nebula-new}/data/meta/
```

   b. Modify the new Meta configuration files:

```
> vim ${nebula-new}/nebula-metad.conf
```

[Optional] Add the following parameters in the Meta configuration files if you need them.

- `--null_type=false`: Disables the support for using `NULL` as schema properties after the upgrade. The default value is `true`. When set to `false`, you must specify a default value when altering tags or edge types, otherwise, data reading fails.
- `--string_index_limit=32`: Specifies the index length for string values as 32. The default length is 64.

6. Prepare the Storage configuration files on each Storage server.
   - If the old Storage data path is not the default setting `--data_path=data/storage`, Modify the Storage configuration file and change the value of `--data_path` as the new data path.

     ```
     > vim ${nebula-new}/nebula-storaged.conf
     ```

   - Create the new Storage data directories.

     ```
     > mkdir -p ${nebula-new}/data/storage/
     ```

   Note

   If the `--data_path` default value has been modified, create the Storage data directories according to the modification.

7. Start the new Meta Service.
   a. Run the following command on each Meta server.

   ```
   $ sudo ${nebula-new}/scripts/nebula.service start metad
   [INFO] Starting nebula-metad...
   [INFO] Done
   ```

   b. Check if every nebula-metad process is started normally.

   ```
   $ ps -ef |grep nebula-metad
   ```

   c. Check if there is any error information in the Meta logs in `${nebula-new}/logs`. If any nebula-metad process cannot start normally, stop upgrading, start the Nebula Graph services from the old directories, and take the error logs to the Nebula Graph community for help.

8. Run the following commands to upgrade the Storage data format.

```
$ sudo ${nebula-new}/bin/db_upgrader  \
--src_db_path=<old_storage_directory_path> \
--dst_db_path=<new_storage_directory_path>  \
--upgrade_meta_server=<meta_server_ip1>:<port1>[,<meta_server_ip2>:<port2>,...] \
--upgrade_version=<old_nebula_version> \
```

The parameters are described as follows.

- `--src_db_path`: Specifies the absolute path of the **OLD** Storage data directories. Separate multiple paths with commas, without spaces.
- `--dst_db_path`: Specifies the absolute path of the **NEW** Storage data directories. Separate multiple paths with commas, without spaces. The paths must correspond to the paths set in `--src_db_path` one by one.

Danger

Don't mix up the preceding two parameters, otherwise, the old data will be damaged during the upgrade.

- `--upgrade_meta_server`: Specifies the addresses of the new Meta servers that you started in step 7.
- `--upgrade_version`: If the old Nebula Graph version is v1.2.0, set the parameter value to `1`. If the old version is v2.0.0-RC1, set the value to 2.

  Danger

  Don't set the value to other numbers.

Example of upgrading from v1.2.0:

```
$ sudo /usr/local/nebula_new/bin/db_upgrader \
--src_db_path=/usr/local/nebula/data/storage/data1/,/usr/local/nebula/data/storage/data2/ \
--dst_db_path=/usr/local/nebula_new/data/storage/data1/,/usr/local/nebula_new/data/storage/data2/\
--upgrade_meta_server=192.168.8.14:45500,192.168.8.15:45500,192.168.8.16:45500 \
--upgrade_version=1
```

Example of upgrading from v2.0.0-RC1:

```
$ sudo /usr/local/nebula_new/bin/db_upgrader \
--src_db_path=/usr/local/nebula/data/storage/ \
--dst_db_path=/usr/local/nebula_new/data/storage/ \
--upgrade_meta_server=192.168.8.14:9559,192.168.8.15:9559,192.168.8.16:9559 \
--upgrade_version=2
```

Note

Make sure that all the Storage servers have finished the upgrade. If anything goes wrong:

a. Stop upgrading.

b. Stop all the Meta servers.

c. Start the Nebula Graph services from the old directories.

d. Go to the Nebula Graph community for help.

9. Start the new Storage Service on each Storage server.

```
$ sudo ${nebula-new}/scripts/nebula.service start storaged
$ sudo ${nebula-new}/scripts/nebula.service status storaged
```

Note

If this step goes wrong on any server:

a. Stop upgrading.

b. Stop all the Meta servers and Storage servers.

c. Start the Nebula Graph services from the old directories.

d. Take the logs in `${nebula-new}/logs/` to the Nebula Graph community for help.

10. Start the new Graph Service on each Graph server.

```
$ sudo ${nebula-new}/scripts/nebula.service start graphd
$ sudo ${nebula-new}/scripts/nebula.service status graphd
```

Note

If this step goes wrong on any server:

1. Stop upgrading. 2. Stop all the Meta servers, Storage servers, and Graph servers. 3. Start the Nebula Graph services from the old directories. 4. Take the logs in `${nebula-new}/logs/` to the Nebula Graph community for help.

11. Connect to Nebula Graph with the new version (v2.0.0 or later) of Nebula Console. Verify if the Nebula Graph services are available and if the data can be accessed normally.

    The command for connection, including the IP address and port of the Graph Service, is the same as the old one.

    The following statements may help in this step.

```
nebula> SHOW HOSTS;
nebula> SHOW SPACES;
nebula> USE <space_name>
nebula> SHOW PARTS;
nebula> SUBMIT JOB STATS;
nebula> SHOW STATS;
```

> **Danger**
>
> Don't use Nebula Console versions prior to v2.0.0.

12. Upgrade other Nebula Graph clients.

    You must upgrade all other clients to corresponding v2.0.0 versions. The clients include but are not limited to the following ones. Find the v2.0.0 branch for each client.

    - studio
    - python
    - java
    - go
    - c++
    - flink-connector
    - spark-util
    - benchmark

> **Note**
>
> - Communication protocols of the v2.0.0 versions are not compatible with that of the historical versions. To upgrade the clients, you must compile the v2.0.0 source code of the clients or download corresponding binaries.
> - Tip for maintenance: The data path after the upgrade is `${nebula-new}/`. Modify relative paths for hard disk monitor systems or log ELK.

5.4.4 Upgrade failure and rollback

If the upgrade fails, stop all Nebula Graph services of the new version, and start the services of the old version.

All Nebula Graph clients in use must be switched to the old version.

5.4.5 Appendix 1: Test Environment

The test environment for this topic is as follows:

- Machine specifications: 32 CPU cores, 62 GB memory, and SSD.
- Data size: 100 GB of Nebula Graph 1.2.0 LDBC test data, with 1 graph space, 24 partitions, and 92 GB of data directory size.
- Concurrent configuration:

Parameter	Default value	Applied value in the Tests
--max_concurrent	5	5
--max_concurrent_parts	10	24
--write_batch_num	100	100

The upgrade cost 21 minutes in all, including 21 minutes of compaction.

5.4.6 Appendix 2: Nebula Graph V2.0.0 code address and commit ID

Code address	Commit ID
Graph Service	7923a45
Storage and Meta Services	761f22b
Common	b2512aa

5.4.7 FAQ

Can I write through the client during the upgrade?

A: No. The state of the data written during this process is undefined.

Can I upgrade other old versions except for v1.2.0 or v2.0.0-RC1 to v2.0.0?

A: Upgrading from other old versions is not tested. Theoretically, versions between v1.0.0 and v1.2.0 could adopt the upgrade approach for v1.2.0. V2.x nightly versions cannot apply the solutions in this topic.

How to upgrade clients after the server upgrade?

A: See step 12 in this topic.

How to upgrade if a machine has only the Graph Service, but not the Storage Service?

A: Upgrade the Graph Service with the corresponding binary or rpm package.

How to resolve the error `Permission denied`?

A: Try again with the sudo privileges.

Is there any change in gflags?

A: Yes. For more information, see known gflags changes.

What are the differences between deleting data then installing the new version and upgrading according to this topic?

A: The default configurations for v2.x and v1.x are different, including the ports used. The upgrade solution keeps the old configurations, and the delete-and-install solution uses the new configurations.

Is there a tool or solution for verifying data consistency after the upgrade?

A: No.

Last update: May 12, 2021

5.5 Uninstall Nebula Graph

This topic describes how to uninstall Nebula Graph.

> **Caution**
>
> Before re-installing Nebula Graph on a machine, follow this topic to completely uninstall the old Nebula Graph, in case the remaining data interferes with the new services.

5.5.1 Prerequisites

You have stopped the Nebula Graph services. For more information, see Manage Nebula Graph services.

5.5.2 Step 1: Delete data files of the Storage and Meta Services

If you have modified the `data_path` in the configuration files for the Meta Service and Storage Service, the directories where Nebula Graph stores data may not be in the installation path of Nebula Graph. Check the configuration files to confirm the data paths, and then manually delete the directories to clear all data.

For a Nebula Graph cluster, delete the data files of all Storage and Meta servers.

> **Caution**
>
> Make sure that you have backed up all important data.

1. Find the data paths in the Storage Service disk settings and Meta Service storage settings. For example:

```
########## Disk ##########
# Root data path. Split by comma. e.g. --data_path=/disk1/path1/,/disk2/path2/
# One path per Rocksdb instance.
--data_path=/nebula/data/storage
```

2. Delete the directories found in step 1.

5.5.3 Step 2: Delete the installation directories

The default installation path is `/usr/local/nebula`. It can be modified while installing Nebula Graph. Delete all installation directories, including the `cluster.id` files in them.

Uninstall Nebula Graph deployed with source code

Find the installation path of Nebula Graph, and delete the directories.

Uninstall Nebula Graph deployed with RPM packages

1. Run the following command to get the Nebula Graph version.

```
$ rpm -qa | grep "nebula"
```

The return message is as follows.

```
nebula-graph-2.0.1-1.x86_64
```

2. Run the following command to uninstall Nebula Graph.

```
sudo rpm -e <nebula_version>
```

For example:

```
sudo rpm -e nebula-graph-2.0.1-1.x86_64
```

3. Delete the installation directories.

Uninstall Nebula Graph deployed with DEB packages

1. Run the following command to get the Nebula Graph version.

```
$ dpkg -l | grep "nebula"
```

The return message is as follows.

```
ii  nebula-graph  2.0.1  amd64     Nebula Package built using CMake
```

2. Run the following command to uninstall Nebula Graph.

```
sudo dpkg -r <nebula_version>
```

For example:

```
sudo dpkg -r nebula-graph
```

3. Delete the installation directories.

Uninstall Nebula Graph deployed with Docker Compose

1. In the `nebula-docker-compose` directory, run the following command to stop the Nebula Graph services.

```
docker-compose down -v
```

2. Delete the `nebula-docker-compose` directory.

Last update: May 10, 2021

6. Configurations and logs

6.1 Configurations

6.1.1 Configurations

This document gives some introduction to configurations in Nebula Graph.

For the path and usage of local configuration files for Nebula Graph services, see:

- Meta configuration
- Graph configuration
- Storage configuration

Get configurations

Most configurations are gflags. You can get all the gflags and the explanations by the following command.

```
<binary> --help
```

For example:

```
$ ./nebula-metad  --help
$ ./nebula-graphd --help
$ ./nebula-storaged --help
$ ./nebula-console --help
```

Besides, you can get the values of running flags by `curl`-ing from the services.

For example:

```
$ curl 127.0.0.1:19559/flags # From Meta
$ curl 127.0.0.1:19669/flags # From Graph
$ curl 127.0.0.1:19779/flags # From Storage
```

Modify configurations

We suggest that you change configurations from local configure files. To change configurations from local files, follow these steps:

1. Add `--local_config=true` to each configuration file. The configuration files are stored in `/usr/local/nebula/etc/` by default. If you have customized your Nebula Graph installation directory, the path to your configuration files is `$pwd/nebula/etc/`.
2. Save your modification to the files.
3. Restart the Nebula Graph services.

> **Caution**
>
> Remember to add `--local_config=true` to each configuration file.

To make your modifications take effect, restart all the Nebula Graph services.

Legacy version compatibility

The `curl` commands and parameters in Nebula Graph v2.x. are different from Nebula Graph v1.x. Those `curl` commands in v1.x are deprecated now.

Last update: April 22, 2021

6.1.2 Meta Service configuration

Nebula Graph provides two initial configuration files for the Meta Service: `nebula-metad.conf.default` and `nebula-metad.conf.production`. You can use them in different scenarios. The default file path is `/usr/local/nebula/etc/`.

How to use the configuration files

The Meta Service gets its configuration from the `nebula-metad.conf` file. You have to remove the suffix `.default` or `.production` from an initial configuration file for the Meta Service to apply the configuration defined in it.

If you have modified the configuration in the file and want new configuration to take effect, add `--local_conf=true` at the top of the file. Otherwise, Nebula Graph reads the cached configuration.

About parameter values

If a parameter is not set in the configuration file, Nebula Graph uses the default value.

> **Note**
>
> The default value of a parameter in Nebula Graph may be different from the predefined value in the `.default` and `.production` files.

The predefined parameters in `nebula-metad.conf.default` and `nebula-metad.conf.production` are different. And not all parameters are predefined. This topic uses the parameters in `nebula-metad.conf.default`.

Nebula Graph provides two initial configuration files for the Meta Service: `nebula-metad.conf.default` and `nebula-metad.conf.production`. You can use them in different scenarios. The default file path is `/usr/local/nebula/etc/`.

Basic configurations

Name	Predefine Value	Descriptions
`daemonize`	`true`	When set to `true`, the process is a daemon process.
`pid_file`	`pids/nebula-metad.pid`	File to host the process ID.
`timezone_name`	-	Specifies the Nebula Graph time zone. This parameter is not predefined in the initial configuration files. You can manually set it if you need it. The system default value is `UTC+00:00:00`. For the format of the parameter value, see Specifying the Time Zone with TZ. For example, `--timezone_name=CST-8` represents the GMT+8 time zone.

> **Note**
>
> - While inserting time-type property values except timestamps, Nebula Graph transforms them to a UTC time according to the time zone specified with the `timezone_name` parameter in the configuration files. The time-type values returned by nGQL queries are all UTC time.
> - `timezone_name` is only used to transform the data stored in Nebula Graph. Other time-related data of the Nebula Graph processes still uses the default time zone of the host, such as the log printing time.

Logging configurations

Name	Predefine Value	Descriptions
`log_dir`	`logs`	Directory to the Meta Service log. We recommend that you put logs on a different hard disk from the `data_path`.
`minloglevel`	`0`	Specifies the minimum log level. Available values are `0` (INFO), `1` (WARNING), `2` (ERROR), and `3` (FATAL). We suggest that you set `minloglevel` to `0` for debugging and `1` for production. When you set it to `4`, Nebula Graph does not print any logs.
`v`	`0`	Specifies the verbose log level. Available values are 0-4. The larger the value, the more verbose the log.
`logbufsecs`	`0`	Specifies the maximum time to buffer the logs. The configuration is measured in seconds.
`stdout_log_file`	`metad-stdout.log`	Specifies the filename for the stdout log.
`stderr_log_file`	`metad-stderr.log`	Specifies the filename for the stderr log.
`stderrthreshold`	`2`	Specifies the minimum level to copy the log messages to stderr.

Networking configurations

Name	Predefine Value	Descriptions
`meta_server_addrs`	`127.0.0.1:9559`	Specifies the IP addresses and ports of all Meta Services. Separate multiple addresses with commas.
`local_ip`	`127.0.0.1`	Specifies the local IP for the Meta Service.
`port`	`9559`	Specifies RPC daemon listening port. The external port for the Meta Service is predefined to `9559`. The internal port is predefined to `port + 1`, i.e., `9560`. Nebula Graph uses the internal port for multi-replica interactions.
`ws_ip`	`0.0.0.0`	Specifies the IP address for the HTTP service.
`ws_http_port`	`19559`	Specifies the port for the HTTP service.
`ws_h2_port`	`19560`	Specifies the port for the HTTP2 service.
`heartbeat_interval_secs`	`10`	Specifies the default heartbeat interval in seconds. Make sure the `heartbeat_interval_secs` values for all services are the same, otherwise Nebula Graph **CANNOT** work normally.

> **Note**
>
> We recommend that you use the real IP address in your configuration because sometimes `127.0.0.1` can not be parsed correctly.

Storage configurations

Name	Predefine Value	Descriptions
`data_path`	data/meta (i.e. /usr/local/nebula/data/meta/)	Directory for cluster metadata persistence

Misc configurations

Name	Predefine Value	Descriptions
`default_parts_num`	`100`	Specifies the default partition number when you create a new graph space.
`default_replica_factor`	`1`	Specifies the default replica factor number when you create a new graph space.

RocksDB options

Name	Predefine Value	Descriptions
`rocksdb_wal_sync`	`true`	Enable or disable RocksDB WAL synchronization. Available values are `true` (enable) and `false` (disable).

Last update: April 22, 2021

6.1.3 Graph Service configuration

Nebula Graph provides two initial configuration files for the Graph Service: `nebula-graphd.conf.default` and `nebula-graphd.conf.production`. You can use them in different scenarios. The default file path is `/usr/local/nebula/etc/`.

How to use the configuration files

The Graph Service gets its configuration from the `nebula-graphd.conf` file. You have to remove the suffix `.default` or `.production` from an initial configuration file for the Graph Service to apply the configuration defined in it.

If you have modified the configuration in the file and want new configuration to take effect, add `--local_conf=true` at the top of the file. Otherwise, Nebula Graph reads the cached configuration.

About parameter values

If a parameter is not set in the configuration file, Nebula Graph uses its default value.

> **Note**
>
> The default value of a parameter in Nebula Graph may be different from the predefined value in the `.default` and `.production` files.

The predefined parameters in `nebula-graphd.conf.default` and `nebula-graphd.conf.production` are different. And not all parameters are predefined. This topic uses the parameters in `nebula-graphd.conf.default`.

Basic configurations

Name	Predefine Value	Descriptions
`daemonize`	`true`	When set to `true`, the process is a daemon process.
`pid_file`	`pids/nebula-graphd.pid`	File to host the process ID.
`enable_optimizer`	`true`	When set to `true`, the optimizer is enabled.
`timezone_name`	-	Specifies the Nebula Graph time zone. This parameter is not predefined in the initial configuration files. You can manually set it if you need it. The system default value is `UTC+00:00:00`. For the format of the parameter value, see Specifying the Time Zone with TZ. For example, `--timezone_name=CST-8` represents the GMT+8 time zone.

> **Note**
>
> - While inserting time-type property values except timestamps, Nebula Graph transforms them to a UTC time according to the time zone specified with the `timezone_name` parameter in the configuration files. The time-type values returned by nGQL queries are all UTC time.
> - `timezone_name` is only used to transform the data stored in Nebula Graph. Other time-related data of the Nebula Graph processes still uses the default time zone of the host, such as the log printing time.

Logging configurations

Name	Predefine Value	Descriptions
`log_dir`	`logs`	Directory to the Graph Service log. We recommend that you put logs on a different hard disk from the `data_path`.
`minloglevel`	`0`	Specifies the minimum log level. Available values are `0` (INFO), `1` (WARNING), `2` (ERROR), and `3` (FATAL). We suggest that you set `minloglevel` to `0` for debugging and `1` for production. When you set it to `4`, Nebula Graph does not print any logs.
`v`	`0`	Specifies the verbose log level. Available values are 0-4. The larger the value, the more verbose the log.
`logbufsecs`	`0`	Specifies the maximum time to buffer the logs. The configuration is measured in seconds.
`redirect_stdout`	`true`	When set to `true`, stdout and stderr are redirected.
`stdout_log_file`	`graphd-stdout.log`	Specifies the filename for the stdout log.
`stderr_log_file`	`graphd-stderr.log`	Specifies the filename for the stderr log.
`stderrthreshold`	`2`	Specifies the minimum level to copy the log messages to stderr.

Networking configurations

Name	Predefine Value	Descriptions
`meta_server_addrs`	`127.0.0.1:9559`	Specifies the IP addresses and ports of all Meta Services. Separate multiple addresses with commas.
`local_ip`	`127.0.0.1`	Specifies the local IP for the Graph Service.
`listen_netdev`	`any`	Specifies the network device to listen on.
`port`	`9669`	Specifies RPC daemon listening port. The external port for the Graph Service is `9669`.
`reuse_port`	`false`	When set to `false`, the SO_REUSEPORT is closed.
`listen_backlog`	`1024`	Specifies the backlog for the listen socket. You must modify this configuration together with the `net.core.somaxconn`.
`client_idle_timeout_secs`	`0`	Specifies the time to close an idle connection. This configuration is measured in seconds.
`session_idle_timeout_secs`	`0`	Specifies the time to expire an idle session. This configuration is measured in seconds.
`num_accept_threads`	`1`	Specifies the thread number to accept incoming connections.
`num_netio_threads`	`0`	Specifies the networking IO threads number. `0` is the number of CPU cores.
`num_worker_threads`	`0`	Specifies the thread number to execute user queries. `0` is the number of CPU cores.
`ws_ip`	`0.0.0.0`	Specifies the IP address for the HTTP service.
`ws_http_port`	`19669`	Specifies the port for the HTTP service.
`ws_h2_port`	`19670`	Specifies the port for the HTTP2 service.
`heartbeat_interval_secs`	`10`	Specifies the default heartbeat interval in seconds. Make sure the `heartbeat_interval_secs` values for all services are the same, otherwise Nebula Graph **CANNOT** work normally.
`storage_client_timeout_ms`	-	Specifies the RPC connection timeout threshold between the Graph Service and the Storage Service. This parameter is not predefined in the initial configuration files. You can manually set it if you need it. The system default value is `60000` ms.

Note

We recommend that you use the real IP address in your configuration because sometimes `127.0.0.1` can not be parsed correctly.

Charset and collate configurations

Name	Predefine Value	Descriptions
`default_charset`	`utf8`	Specifies the default charset when you create a new graph space.
`default_collate`	`utf8_bin`	Specifies the default collate when you create a new graph space.

Authorization and authentication configurations

Name	Predefine Value	Descriptions
`enable_authorize`	`false`	When set to `false`, the system authentication is not enabled. For more information, see Authentication.
`auth_type`	`password`	Specifies the login method. Available values are `password`, `ldap`, and `cloud`.

If you have set `enable_authorize` to `true`, you can only log in with the root account. For example:

```
/usr/local/nebula/bin/nebula -u root -p nebula --addr=127.0.0.1 --port=9669
```

If you have set `enable_authorize` to `false`, you can log in with any account and password. For example:

```
/usr/local/nebula/bin/nebula -u any -p 123 --addr=127.0.0.1 --port=9669
```

Last update: April 22, 2021

6.1.4 Storage Service configurations

Nebula Graph provides two initial configuration files for the Storage Service: `nebula-storaged.conf.default` and `nebula-storaged.conf.production`. You can use them in different scenarios. The default file path is `/usr/local/nebula/etc/`.

> **Note**
>
> Raft Listener is different from the Storage Service. For more information, see Raft Listener.

How to use the configuration files

The Storage Service gets its configuration from the `nebula-storaged.conf` file. You have to remove the suffix `.default` or `.production` from an initial configuration file for the Storage Service to apply the configuration defined in it.

If you have modified the configuration in the file and want the new configuration to take effect, add `--local_conf=true` at the top of the file. Otherwise, Nebula Graph reads the cached configuration.

About parameter values

If a parameter is not set in the configuration file, Nebula Graph uses its default value.

> **Note**
>
> The default value of a parameter in Nebula Graph may be different from the predefined value in the `.default` and `.production` files.

The predefined parameter in `nebula-storaged.conf.default` and `nebula-storaged.conf.production` are different. And not all parameters are predefined. This topic uses the parameters in `nebula-storaged.conf.default`.

Basic configurations

Name	Predefine Value	Descriptions
`daemonize`	`true`	When set to `true`, the process is a daemon process.
`pid_file`	`pids/nebula-storaged.pid`	File to host the process ID.
`timezone_name`	-	Specifies the Nebula Graph time zone. This parameter is not predefined in the initial configuration files. You can manually set it if you need it. The system default value is `UTC+00:00:00`. For the format of the parameter value, see Specifying the Time Zone with TZ. For example, `--timezone_name=CST-8` represents the GMT+8 time zone.

> **Note**
>
> - While inserting time-type property values except timestamps, Nebula Graph transforms them to a UTC time according to the time zone specified with the `timezone_name` parameter in the configuration files. The time-type values returned by nGQL queries are all UTC time.
> - `timezone_name` is only used to transform the data stored in Nebula Graph. Other time-related data of the Nebula Graph processes still uses the default time zone of the host, such as the log printing time.

Logging configurations

Name	Predefine Value	Descriptions
`log_dir`	`logs`	Directory to the Storage Service log. We recommend that you put logs on a different hard disk from the `data_path`.
`minloglevel`	`0`	Specifies the minimum log level. Available values are 0-3. `0`, `1`, `2`, and `3` are `INFO`, `WARNING`, `ERROR`, and `FATAL`. We suggest that you set `minloglevel` to `0` for debug, `1` for production. When you set it to `4`, Nebula Graph does not print any logs.
`v`	`0`	Specifies the verbose log level. Available values are 0-4. The larger the value, the more verbose the log.
`logbufsecs`	`0`	Specifies the maximum time to buffer the logs. The configuration is measured in seconds.
`redirect_stdout`	`true`	When set to `true`, stdout and stderr are redirected.
`stdout_log_file`	`storaged-stdout.log`	Specifies the filename for the stdout log.
`stderr_log_file`	`storaged-stderr.log`	Specifies the filename for the stderr log.
`stderrthreshold`	`2`	Specifies the minimum level to copy the log messages to stderr. Available values are 0-3. `0`, `1`, `2`, and `3` are `INFO`, `WARNING`, `ERROR`, and `FATAL`.

Networking configurations

Name	Predefine Value	Descriptions
`meta_server_addrs`	`127.0.0.1:9559`	Specifies the IP addresses and ports of all Meta Services. Separate multiple addresses with commas.
`local_ip`	`127.0.0.1`	Specifies the local IP for the Storage Service.
`port`	`9779`	Specifies RPC daemon listening port. The external port for Storage Service is predefined to `9779`. The internal ports are predefined to `port -2`, `port -1`, and `port + 1`, i.e., `9777`, `9778`, and `9780`. Nebula Graph uses the internal ports for multi-replica interactions.
`ws_ip`	`0.0.0.0`	Specifies the IP address for the HTTP service.
`ws_http_port`	`19779`	Specifies the port for the HTTP service.
`ws_h2_port`	`19780`	Specifies the port for the HTTP2 service.
`heartbeat_interval_secs`	`10`	Specifies the default heartbeat interval in seconds. Make sure the `heartbeat_interval_secs` values for all services are the same, otherwise Nebula Graph **CANNOT** work normally.

Note

We recommend that you use the real IP address in your configuration because sometimes `127.0.0.1` can not be parsed correctly.

Raft configurations

Name	Predefine Value	Descriptions
`raft_heartbeat_interval_secs`	`30`	Specifies the timeout for the Raft election. The configuration is measured in seconds.
`raft_rpc_timeout_ms`	`500`	Specifies the timeout for the Raft RPC. The configuration is measured in milliseconds.
`wal_ttl`	`14400`	Specifies the recycle RAFT wal time. The configuration is measured in seconds.

Disk configurations

Name	Predefine Value	Descriptions
`data_path`	`data/storage`	Specifies the root data path. Separate multiple paths with commas.
`rocksdb_batch_size`	`4096`	Specifies the block cache for a batch operation. The configuration is mea
`rocksdb_block_cache`	`4`	Specifies the block cache for BlockBasedTable. The configuration is mea megabytes.
`engine_type`	`rocksdb`	Specifies the engine type.
`rocksdb_compression`	`lz4`	Specifies the compression algorithm for RocksDB. Available values are `lz4`, `lz4hc`, `zlib`, `bzip2`, and `zstd`.
`rocksdb_compression_per_level`	\	Specifies compression for each level.
`enable_rocksdb_statistics`	`false`	When set to `false`, RocksDB statistics is disabled.
`rocksdb_stats_level`	`kExceptHistogramOrTimers`	Specifies the stats level for RocksDB. Available values are `kExceptHisto` `kExceptTimers`, `kExceptDetailedTimers`, `kExceptTimeForMutex`, and `kAll`.
`enable_rocksdb_prefix_filtering`	`false`	When set to `true`, the prefix bloom filter for RocksDB is enabled. Enabl filter reduces memory usage.
`rocksdb_filtering_prefix_length`	`12`	Specifies the prefix length for each key. Available values are `12` and `16`

RocksDB options

The format of the RocksDB options is `{"<option_name>":"<option_value>"}`. Multiple options are separated with commas.

Name	Predefine Value	Descriptions
`rocksdb_db_options`	`{}`	Specifies the RocksDB options.
`rocksdb_column_family_options`	`{"write_buffer_size":"67108864", "max_write_buffer_number":"4", "max_bytes_for_level_base":"268435456"}`	Specifies the RocksDB column family options.
`rocksdb_block_based_table_options`	`{"block_size":"8192"}`	Specifies the RocksDB block based table options.

Available `rocksdb_db_options` and `rocksdb_column_family_options` are listed as follows.

- `rocksdb_db_options`

```
max_total_wal_size
delete_obsolete_files_period_micros
max_background_jobs
stats_dump_period_sec
compaction_readahead_size
writable_file_max_buffer_size
bytes_per_sync
wal_bytes_per_sync
delayed_write_rate
avoid_flush_during_shutdown
max_open_files
stats_persist_period_sec
stats_history_buffer_size
strict_bytes_per_sync
enable_rocksdb_prefix_filtering
enable_rocksdb_whole_key_filtering
rocksdb_filtering_prefix_length
num_compaction_threads
rate_limit
```

- `rocksdb_column_family_options`

```
write_buffer_size
max_write_buffer_number
level0_file_num_compaction_trigger
level0_slowdown_writes_trigger
level0_stop_writes_trigger
target_file_size_base
target_file_size_multiplier
max_bytes_for_level_base
max_bytes_for_level_multiplier
disable_auto_compactions
```

For more information about RocksDB configuration, see RocksDB official documentation

For super-Large vertices

For super vertex with a large number of edges, currently there are two truncation strategies:

1. Truncate directly. Set the `enable_reservoir_sampling` parameter to `false`. A certain number of edges specified in the `Max_edge_returned_per_vertex` parameter are truncated by default.
2. Truncate with the reservoir sampling algorithm. Based on the algorithm, a certain number of edges specified in the `Max_edge_returned_per_vertex` parameter are truncated with equal probability from the total n edges. Equal probability sampling is useful in some business scenarios. However, the performance is affected compared to direct truncation due to the probability calculation.

Storage configuration for large dataset

When you have a large dataset (in the RocksDB directory) and your memory is tight, we suggest that you set the `enable_partitioned_index_filter` parameter to `true`. For example, 100 vertices + 100 edges require 300 key-values. Each key takes 10bit in memory. Then you can calculate your own memory usage.

Last update: May 10, 2021

6.1.5 Kernel configurations

This document gives some introductions to the Kernel configurations in Nebula Graph.

ulimit

ULIMIT -C

`ulimit -c` limits the size of the core dumps. We recommend that you set it to `unlimited`. The command is:

```
ulimit -c unlimited
```

ULIMIT -N

`ulimit -n` limits the number of open files. We recommend that you set it to more than 100,000. For example:

```
ulimit -n 130000
```

Memory

VM.SWAPPINESS

`vm.swappiness` is the percentage of the free memory before starting swap. The greater the value, the more likely the swap occurs. We recommend that you set it to 0. When set to 0, the page cache is removed first. Note that when `vm.swappiness` is 0, it does not mean that there is no swap.

VM.MIN_FREE_KBYTES

`vm.min_free_kbytes` is used to force the Linux VM to keep a minimum number of kilobytes free. If you have a large system memory, we recommend that you increase this value. For example, if your physical memory 128GB, set it to 5GB. If the value is not big enough, the system cannot apply for enough continuous physical memory.

VM.MAX_MAP_COUNT

`vm.max_map_count` limits the maximum number of vma (virtual memory area) for a process. The default value is `65530`. It is enough for most applications. If your memory application fails because the memory consumption is large, increase the `vm.max_map_count` value.

VM.OVERCOMMIT_MEMORY

`vm.overcommit_memory` contains a flag that enables memory overcommitment. We recommend that you set the default value 0 or 1. DO NOT set it to 2.

VM.DIRTY_*

These values control the aggressiveness of the dirty page cache for the system. For write-intensive scenarios, you can make adjustments based on your needs (throughput priority or delay priority). We recommend that you use the system default value.

TRANSPARENT HUGE PAGE

For better delay performance, you must delete the transparent huge pages (THP). The options are `/sys/kernel/mm/transparent_hugepage/enabled` and `/sys/kernel/mm/transparent_hugepage/defrag`. For example:

```
echo never > /sys/kernel/mm/transparent_hugepage/enabled
echo never > /sys/kernel/mm/transparent_hugepage/defrag
swapoff -a && swapon -a
```

Networking

NET.IPV4.TCP_SLOW_START_AFTER_IDLE

The default value for this parameter is `1`. If set, the congestion window is timed out after an idle period. We recommend that you set it to 0, especially for long fat links (high latency and large bandwidth).

NET.CORE.SOMAXCONN

`net.core.somaxconn` is the maximum number of the backlogged sockets. The default value is `128`. For scenarios with a large number of burst connections, we recommend that you set it to greater than `1024`.

NET.IPV4.TCP_MAX_SYN_BACKLOG

The maximum number of remembered connection requests. The setting rule for this parameter is the same as that of `net.core.somaxconn`.

NET.CORE.NETDEV_MAX_BACKLOG

It determines the maximum number of packets. We recommend that you increase it to greater than 10,000, especially for 10G network adapters. The default value is `1000`.

NET.IPV4.TCP_KEEPALIVE_*

Keep alive parameters for the TCP connections. For applications that use a 4-layer transparent load balancer, if the idle connection is disconnected unexpectedly, decrease `tcp_keepalive_time` and `tcp_keepalive_intvl`.

NET.IPV4.TCP_RMEM/WMEM

The minimum, default, and maximum size of the TCP socket receive buffer. For long fat links, we recommend that you increase the default value to bandwidth * RTT.

SCHEDULER

For SSD devices, we recommend that you set `/sys/block/DEV_NAME/queue/scheduler` to noop or none.

Other parameters

KERNEL.CORE_PATTERN

we recommend that you set it to core and set `kernel.core_uses_pid` to 1.

Parameter usage guide

SYSCTL

- `sysctl conf_name` checks the current parameter value.
- `sysctl -w conf_name=value` modifies the parameter value. And your modification takes effect immediately.
- `sysctl -p` loads parameter values from related configuration files.

INTRODUCTION TO ULIMIT

`ulimit` sets the resource threshold for the current shell session. Please note that:

- Changes made by the `ulimit` command are valid only for the current session (and child processes).
- `ulimit` cannot adjust the (soft) threshold of a resource to a value greater than the current hard value.
- Ordinary users cannot adjust the hard threshold (even by using `sudo`) through this command.
- To modify on the system level, or adjust the hard threshold, edit the `/etc/security/limits.conf` file. But this method needs to re-log in to take effect.

PRLIMIT

`prlimit` gets and sets process resource limits. You can modify the hard threshold by using it and the `sudo` command. Together with the `sudo` command, the hard threshold can be modified. For example, `prlimit --nofile = 130000 --pid = $$` adjusts the maximum number of open files permitted by the current process to `14000`. And the modification takes effect immediately. Note that this command is only available in RedHat 7u or later OS versions.

Last update: February 8, 2021

6.2 Log management

6.2.1 Logs

Nebula Graph uses glog to print logs, uses gflag to control the severity level of the log, and provides an HTTP interface to dynamically change the log level at runtime to facilitate tracking.

Log Directory

The default log directory is `/usr/local/nebula/logs/`.

> **Note**
>
> If you deleted the log directory during runtime, the runtime log would not continue to be printed. However, this operation will not affect the services. Restart the services to recover the logs.

Parameter Description

TWO MOST COMMONLY USED FLAGS IN GLOG

- minloglevel: The scale of minloglevel is 0-4. The numbers of severity levels INFO(DEBUG), WARNING, ERROR, and FATAL are 0, 1, 2, and 3, respectively. Usually specified as 0 for debug, 1 for production. If you set the minloglevel to 4, no logs are printed.
- v: The scale of v is 0-3. When the value is set to 0, you can further set the severity level of the debug log. The greater the value is, the more detailed the log is.

CONFIGURATION FILES

The default severity level for the metad, graphd, and storaged logs can be found in the configuration files (usually in `/usr/local/nebula/etc/`).

Check and Change the Severity Levels Dynamically

Check all the flag values (log values included) of the current gflags with the following command. Not all flags are listed because changing some flags can be dangerous. Read the response explanation and the source code before you change these not documented parameters. To get all the available flags for a process, use this command:

```
> curl ${ws_ip}:${ws_port}/flags
```

In the command:

- `ws_ip` is the IP address for the HTTP service, which can be found in the configuration files above. The default value is `127.0.0.1`.
- `ws_port` is the port for the HTTP service, the default values for `metad`, `storaged`, and `graphd` are `19559`, `19779`, and `19669`, respectively.

> **Note**
>
> If you changed the runtime log level, then restart the services, the log level changes to the configuration file specifications. For more information, see Storage Service configurations.

For example, check the minloglevel for the `storaged` service:

```
> curl 127.0.0.1:19559/flags | grep minloglevel
```

To change the log level for a process, use these commands. For example, you can change the log severity level the **the most detailed**.

```
$ curl -X PUT -H "Content-Type: application/json" -d '{"minloglevel":0,"v":3}' "127.0.0.1:19779/flags" # storaged
$ curl -X PUT -H "Content-Type: application/json" -d '{"minloglevel":0,"v":3}' "127.0.0.1:19669/flags" # graphd
$ curl -X PUT -H "Content-Type: application/json" -d '{"minloglevel":0,"v":3}' "127.0.0.1:19559/flags" # metad
```

To change the severity of the storage log, replace the port in the preceding command with `storage` port.

> **Note**
>
> Nebula Graph only supports modifying the graph and storage log severity by using the console. And the severity level of meta logs can only be modified with the `curl` command.

Close all logs print (FATAL only) with the following command.

```
$ curl -X PUT -H "Content-Type: application/json" -d '{"minloglevel":3,"v":0}' "127.0.0.1:19779/flags" # storaged
$ curl -X PUT -H "Content-Type: application/json" -d '{"minloglevel":3,"v":0}' "127.0.0.1:19669/flags" # graphd
$ curl -X PUT -H "Content-Type: application/json" -d '{"minloglevel":3,"v":0}' "127.0.0.1:19559/flags" # metad
```

Last update: April 22, 2021

7. Monitor and metrics

7.1 Query Nebula Graph metrics

Nebula Graph supports querying the monitoring metrics through HTTP ports.

7.1.1 Metrics

Each metric of Nebula Graph consists of three fields: name, type, and time range. The fields are separated by periods, for example, `num_queries.sum.600`. The detailed description is as follows.

Field	Example	Description
Metric name	`num_queries`	Indicates the function of the metric.
Metric type	`sum`	Indicates how the metrics are collected. Supported types are SUM, COUNT, AVG, RATE, and the P-th sample quantiles such as P75, P95, P99, and P99.9.
Time range	`600`	The time range in seconds for the metric collection. Supported values are 5, 60, 600, and 3600, representing the last 5 seconds, 1 minute, 10 minutes, and 1 hour.

Different Nebula Graph services (Graph, Storage, or Meta) support different metrics, for more information, see Metric list (TODO: doc).

7.1.2 Query metrics over HTTP

Syntax

```
$ curl -G "http://<ip>:<port>/stats?stats=<metric_name_list>[&format=json]"
```

Parameter	Description
`ip`	The IP address of the server. You can find it in the configuration file in the installation directory.
`port`	The HTTP port of the server. You can find it in the configuration file in the installation directory. The default ports are 19559 (Meta), 19669 (Graph), and 19779 (Storage).
`metric_name_list`	The metrics names. Multiple metrics are separated by commas (,).
`&format=json`	Optional. Returns the result in the JSON format.

> **Note**
>
> If Nebula Graph is deployed with Docker Compose, run `docker-compose ps` to check the ports that are mapped from the service ports inside of the container and then query through them.

Example

- Query a single metric

  Query the query number in the last 10 minutes in the Graph Service.

```
$ curl -G "http://192.168.8.40:19669/stats?stats=num_queries.sum.600"
num_queries.sum.600=400
```

- Query multiple metrics

  Query the following metrics together: * The average heartbeat latency in the last 1 minute. * The average latency of the slowest 1% heartbeats, i.e., the P99 heartbeats, in the last 10 minutes.

```
$ curl -G "http://192.168.8.40:19559/stats?stats=heartbeat_latency_us.avg.60,heartbeat_latency_us.p99.600"
heartbeat_latency_us.avg.60=281
heartbeat_latency_us.p99.600=985
```

- Return a JSON result.

  Query the number of new vertices in the Storage Service in the last 10 minutes and return the result in the JSON format.

```
$ curl -G "http://192.168.8.40:19779/stats?stats=num_add_vertices.sum.600&format=json"
[{"value":1,"name":"num_add_vertices.sum.600"}]
```

- Query all metrics in a service.

  If no metric is specified in the query, Nebula Graph returns all metrics in the service.

```
$ curl -G "http://192.168.8.40:19559/stats"
heartbeat_latency_us.avg.5=304
heartbeat_latency_us.avg.60=308
heartbeat_latency_us.avg.600=299
heartbeat_latency_us.avg.3600=285
heartbeat_latency_us.p75.5=652
heartbeat_latency_us.p75.60=669
heartbeat_latency_us.p75.600=651
heartbeat_latency_us.p75.3600=642
heartbeat_latency_us.p95.5=930
heartbeat_latency_us.p95.60=963
heartbeat_latency_us.p95.600=933
heartbeat_latency_us.p95.3600=929
heartbeat_latency_us.p99.5=986
heartbeat_latency_us.p99.60=1409
heartbeat_latency_us.p99.600=989
heartbeat_latency_us.p99.3600=986
num_heartbeats.rate.5=0
num_heartbeats.rate.60=0
num_heartbeats.rate.600=0
num_heartbeats.rate.3600=0
num_heartbeats.sum.5=2
num_heartbeats.sum.60=40
num_heartbeats.sum.600=394
num_heartbeats.sum.3600=2364
```

Last update: April 22, 2021

8. Data security

8.1 Authentication and authorization

8.1.1 Authentication

Nebula Graph replies on local authentication or LDAP authentication to implement access control.

Nebula Graph creates a session when a client connects to it. The session stores information about the connection, including the user information.

By default, authentication is disabled and Nebula Graph allows connections with any username and password. If the authentication system is enabled, Nebula Graph checks a session according to the authentication configuration, and decides whether the session should be allowed or denied.

Local authentication

Local authentication indicates that usernames and passwords are stored locally on the server, with the passwords encrypted.

ENABLE LOCAL AUTHENTICATION

1. In the `/usr/local/nebula/etc/nebula-graphd.conf` file, set `--enable_authorize=true` and save the modification.

> **Note**
>
> `/usr/local/nebula/` is the default installation path for Nebula Graph. If you have changed it, use the actual path.

2. Restart the Nebula Graph services. For how to restart, see Manage Nebula Graph services.

> **Note**
>
> You can use the username `root` and password `nebula` to log into Nebula Graph after enabling local authentication. This account has the build-in God role. For more information about roles, see Roles and privileges.

LDAP authentication

Lightweight Directory Access Protocol (LDAP), is a lightweight client-server protocol for accessing directories and building a centralized account management system.

LDAP authentication and local authentication can be enabled at the same time, but LDAP authentication has a higher priority. If the local authentication server and the LDAP server both have the information of user `Amber`, Nebula Graph reads from the LDAP server first.

ENABLE LDAP AUTHENTICATION

The Nebula Graph Enterprise Edition supports LDAP authentication. For how to enable LDAP, see Authenticate with an LDAP server (TODO: doc).

Last update: April 22, 2021

8.1.2 User management

This topic describes how to manage users and roles.

By default, Nebula Graph allows connections with any username and password. After enabling authentication, only valid users can connect to Nebula Graph and access the resources according to the user roles.

CREATE USER

The `root` user with the GOD role can run `CREATE USER` to create a new user.

- Syntax

```
CREATE USER [IF NOT EXISTS] <user_name> [WITH PASSWORD '<password>'];
```

- Example

```
nebula> CREATE USER user1 WITH PASSWORD 'nebula';
```

GRANT ROLE

Users with the GOD role or the ADMIN role can run `GRANT ROLE` to assign a built-in role in a graph space to a user. For more information about Nebula Graph built-in roles, see Roles and privileges

> **Note**
>
> If the target user is connected to Nebula Graph when running `GRANT ROLE`, the new role takes effect when the user logs out and logs in again.

- Syntax

```
GRANT ROLE <role_type> ON <space_name> TO <user_name>;
```

- Example

```
nebula> GRANT ROLE USER ON basketballplayer TO user1;
```

REVOKE ROLE

Users with the GOD role or the ADMIN role can run `REVOKE ROLE` to revoke a user's role in a graph space.

> **Note**
>
> If the target user is connected to Nebula Graph when running `REVOKE ROLE`, the old role still takes effect until the user logs out.

- Syntax

```
REVOKE ROLE <role_type> ON <space_name> FROM <user_name>;
```

- Example

```
nebula> REVOKE ROLE USER ON basketballplayer FROM user1;
```

CHANGE PASSWORD

With the correct username and password, users can run `CHANGE PASSWORD` to set a new password for a user.

- Syntax

```
CHANGE PASSWORD <user_name> FROM '<old_password>' TO '<new_password>';
```

- Example

```
nebula> CHANGE PASSWORD user1 FROM 'nebula' TO 'nebula123';
```

ALTER USER

The `root` user with the GOD role can run `ALTER USER` to set a new password for a user.

- Syntax

```
ALTER USER <user_name> WITH PASSWORD '<password>';
```

- Example

```
nebula> ALTER USER user1 WITH PASSWORD 'nebula';
```

DROP USER

The `root` user with the GOD role can run `DROP USER` to remove a user.

> **Note**
>
> Removing a user does not close the user's current session, and the user role still takes effect in the session until the session is closed.

- Syntax

```
DROP USER [IF EXISTS] <user_name>;
```

- Example

```
nebula> DROP USER user1;
```

SHOW USERS

The `root` user with the GOD role can run `SHOW USERS` to list all the users.

- Syntax

```
SHOW USERS;
```

- Example

```
nebula> SHOW USERS;
+-----------+
| Account   |
+-----------+
| "test1"   |
+-----------+
| "test2"   |
+-----------+
| "test3"   |
+-----------+
```

Last update: April 22, 2021

8.1.3 Roles and privileges

A role is a collection of privileges. You can assign a role to a user for access control.

Built-in roles

Nebula Graph does not support custom roles, but it has multiple built-in roles:

- GOD
  - GOD is the original role with all privileges not limited to graph spaces. It is similar to `root` in Linux and `administrator` in Windows.
  - When the Meta Service is initialized, the one and only GOD role user `root` is automatically created with the password `nebula`.

  > **Note**
  >
  > Modify the password for `root` as soon as possible for security.

  - The default username `root` is immutable.
  - If authentication is disabled, you can use any username and password to connect to Nebula Graph. This user is regarded as the GOD role.
- ADMIN
  - An ADMIN role can read and write both the Schema and the data in a specific graph space.
  - An ADMIN role of a graph space can grant DBA, USER, and GUEST roles in the graph space to other users.
- DBA
  - A DBA role can read and write both the Schema and the data in a specific graph space.
  - A DBA role of a graph space CANNOT grant roles to other users.
- USER
  - A USER role can read and write data in a specific graph space.
  - The Schema information is read-only to the USER roles in a graph space.
- GUEST
  - A GUEST role can only read the Schema and the data in a specific graph space.

> **Note**
>
> A user can have only one role in a graph space.

Role privileges and allowed nGQL

The privileges of roles and the nGQL statements that each role can use are listed as follows.

Privilege	God	Admin	DBA	User	Guest	Allowed nGQL
Read space	Y	Y	Y	Y	Y	`USE`, `DESCRIBE SPACE`
Write space	Y					`CREATE SPACE`, `DROP SPACE`, `CREATE SNAPSHOT`, `DROP SNAPSHOT`, `BALANCE`, `ADMIN`, `CONFIG`, `INGEST`, `DOWNLOAD`
Read schema	Y	Y	Y	Y	Y	`DESCRIBE TAG`, `DESCRIBE EDGE`, `DESCRIBE TAG INDEX`, `DESCRIBE EDGE INDEX`
Write schema	Y	Y	Y			`CREATE TAG`, `ALTER TAG`, `CREATE EDGE`, `ALTER EDGE`, `DROP TAG`, `DROP EDGE`, `CREATE TAG INDEX`, `CREATE EDGE INDEX`, `DROP TAG INDEX`, `DROP EDGE INDEX`
Write user	Y					`CREATE USER`, `DROP USER`, `ALTER USER`
Write role	Y	Y				`GRANT`, `REVOKE`
Read data	Y	Y	Y	Y	Y	`GO`, `SET`, `PIPE`, `MATCH`, `ASSIGNMENT`, `LOOKUP`, `YIELD`, `ORDER BY`, `FETCH VERTICES`, `Find`, `FETCH EDGES`, `FIND PATH`, `LIMIT`, `GROUP BY`, `RETURN`
Write data	Y	Y	Y	Y		`BUILD TAG INDEX`, `BUILD EDGE INDEX`, `INSERT VERTEX`, `UPDATE VERTEX`, `INSERT EDGE`, `UPDATE EDGE`, `DELETE VERTEX`, `DELETE EDGES`
Show operations	Y	Y	Y	Y	Y	`SHOW`, `CHANGE PASSWORD`

Note

- The results of `SHOW` operations are limited to the role of a user. For example, all users can run `SHOW SPACES`, but the results only include the graph spaces that the users have privileges.
- Only the GOD role can run `SHOW USERS` and `SHOW SNAPSHOTS`.

Last update: April 22, 2021

8.2 Backup and restore data with snapshots

Nebula Graph supports using snapshots to backup and restore data.

8.2.1 Authentication and snapshots

Nebula Graph authentication is disabled by default. In this case, All users can use the snapshot feature.

If authentication is enabled, only the GOD-role user can use the snapshot function. For more information about roles, see Roles and privileges.

8.2.2 Precautions

- To prevent data loss, create a snapshot as soon as the system structure changes, for example, after operations such as `ADD HOST`, `DROP HOST`, `CREATE SPACE`, `DROP SPACE`, and `BALANCE` are performed.
- Nebula Graph cannot automatically delete the invalid files created by a failed snapshot task, you have to manually delete them by using `DROP SNAPSHOT`.
- Customizing the storage path for the snapshots is not supported for now.

8.2.3 Snapshot form and path

Nebula Graph snapshots are in the form of directories with names like `SNAPSHOT_2021_03_09_08_43_12`. The suffix `2021_03_09_08_43_12` is generated automatically based on the creation time.

When a snapshot is created, snapshot directories will be automatically created in the `checkpoints` directory on the leader Meta server and each Storage server.

To fast locate the path where the snapshots are stored, you can use the Linux command `find`. For example:

```
$ find |grep 'SNAPSHOT_2021_03_11_07_30_36'
./data/meta2/nebula/0/checkpoints/SNAPSHOT_2021_03_11_07_30_36
./data/meta2/nebula/0/checkpoints/SNAPSHOT_2021_03_11_07_30_36/data
./data/meta2/nebula/0/checkpoints/SNAPSHOT_2021_03_11_07_30_36/data/000081.sst
...
```

Note

For how to get the snapshot name, see View snapshots.

8.2.4 Create a snapshot

Run `CREATE SNAPSHOT` to create a snapshot for all the graph spaces based on the current time for Nebula Graph.

Note

Creating a snapshot for a specific graph space is not supported yet.

If the creation fails, delete the snapshot and try again. If it still fails, go to the Nebula Graph community for help.

8.2.5 View snapshots

To view all existing snapshots, run `SHOW SNAPSHOTS`.

```
nebula> SHOW SNAPSHOTS;
+---------------------------------+---------+------------------+
| Name                            | Status  | Hosts            |
```

```
+--------------------------------+---------+------------------+
| "SNAPSHOT_2021_03_09_08_43_12" | "VALID" | "127.0.0.1:9779" |
+--------------------------------+---------+------------------+
| "SNAPSHOT_2021_03_09_09_10_52" | "VALID" | "127.0.0.1:9779" |
+--------------------------------+---------+------------------+
```

The parameters in the return information are described as follows.

Parameter	Description
`Name`	Name of the snapshot directory.
`Status`	Status of the snapshot. `VALID` indicates that the creation succeeded and `INVALID` indicates that it failed.
`Hosts`	IP addresses and ports of all Storage servers at the time the snapshot was created.

8.2.6 Delete a snapshot

To delete a snapshot, use the following syntax:

```
DROP SNAPSHOT <snapshot_name>;
```

Example:

```
nebula> DROP SNAPSHOT SNAPSHOT_2021_03_09_08_43_12;
nebula> SHOW SNAPSHOTS;
+--------------------------------+---------+------------------+
| Name                           | Status  | Hosts            |
+--------------------------------+---------+------------------+
| "SNAPSHOT_2021_03_09_09_10_52" | "VALID" | "127.0.0.1:9779" |
+--------------------------------+---------+------------------+
```

8.2.7 Restore data with a snapshot

1. Find the snapshot directories you want to use for data restoration.
2. Choose an approach to restore the data files:
   - Change the `data_path` in the Meta configuration and Storage configuration to the snapshot path.
   - Copy the snapshot directories to other locations, and change the `data_path` to these locations.
   - Copy all the content in the snapshot directories into the directories where the `checkpoints` directories are located, and cover the existing files that have duplicate names with them. For example, cover `/usr/local/nebula/data/meta/nebula/0/data` with `/usr/local/nebula/data/meta/nebula/0/checkpoints/SNAPSHOT_2021_03_09_09_10_52/data`.
3. Restart Nebula Graph.

8.2.8 Another way to backup and restore data

You can also use Backup&Restore to backup and restore Nebula Graph data. (TODO: coding)

Last update: April 22, 2021

9. Service Tuning

9.1 Compaction

This document gives some information about compaction.

9.1.1 Introduction to compaction

In Nebula Graph, compaction is the most important background process. Compaction has an important effect on performance.

Compaction reads the data that is written on the hard disk, then re-organizes the data structure and the indexes to make the data easier to read. The read performance can increase by times after compaction. Thus, to get high read performance, trigger compaction manually when writing a large amount of data into Nebula Graph. Note that compaction leads to long time hard disk IO, we suggest that you do compaction during off-peak hours (for example, early morning).

Nebula Graph has two types of compaction: automatic compaction and full compaction.

9.1.2 Automatic compaction

Automatic compaction is done when the system reads data, writes data, or the system restarts. The automatic compaction is enabled by default. But once triggered during peak hours, it can cause unexpected IO occupancy that has an unwanted effect on the performance. To disable automatic compaction, use this statement:

```
nebula> UPDATE CONFIGS storage:rocksdb_column_family_options = {disable_auto_compactions = true};
```

Caution

The command overwrites all `rocksdb_column_family_options` items. Other items besides `disable_auto_compactions` is overwritten to the default value. You may have to read all the items before the updates.

9.1.3 Full compaction

Full compaction enables large scale background operations for a graph space such as merging files, deleting the data expired by TTL. Use these statements to enable full compaction:

```
nebula> USE <your_graph_space>;
nebula> SUBMIT JOB COMPACT;
```

The preceding statement returns a job_id. To show the compaction progress, use this statement:

```
nebula> SHOW JOB <job_id>;
```

Note

Do the full compaction during off-peak hours because full compaction has a lot of IO operations.

9.1.4 Operation suggestions

These are some operation suggestions to keep Nebula Graph performing well.

- To avoid unwanted IO waste during data writing, set `disable_auto_compactions` to `true` before large amounts of data writing.
- After data import is done, run `SUBMIT JOB COMPACT`.
- Run `SUBMIT JOB COMPACT` periodically during off-peak hours, for example, early morning.
- Set `disable_auto_compactions` to `false` during day time.
- To control the read and write traffic limitation for compactions, set these two parameters in the `nebula-storaged.conf` configuration file.

```
# read from the local configuration file and start
--local-config=true
--rate_limit=20 (in MB/s)
```

9.1.5 FAQ

Q: Can I do full compactions for multiple graph spaces at the same time? A: Yes, you can. But the IO is much larger at this time.

Q: How much time does it take for full compactions? A: When `rate_limit` is set to `20`, you can estimate the full compaction time by dividing the hard disk usage by the `rate_limit`. If you do not set the `rate_limit` value, the empirical value is around 50 MB/s.

Q: Can I modify `--rate_limit` dynamically? A: No, you cannot.

Q: Can I stop a full compaction after it starts? A: No you cannot. When you start a full compaction, you have to wait till it is done. This is the limitation of RocksDB.

Last update: April 22, 2021

9.2 Storage load balance

You can use the `BALANCE` statements to balance the distribution of partitions and Raft leaders, or remove redundant Storage servers.

9.2.1 Prerequisites

The graph spaces stored in Nebula Graph must have more than one replicas for the system to balance the distribution of partitions and Raft leaders.

9.2.2 Balance partition distribution

`BALANCE DATA` starts a task to equally distribute the storage partitions in a Nebula Graph cluster. A group of subtasks will be created and implemented to migrate data and balance the partition distribution.

> **Danger**
>
> DON'T stop any machine in the cluster or change its IP address until all the subtasks finish. Otherwise, the follow-up subtasks fail.

Take scaling out Nebula Graph for an example.

After you add new storage hosts into the cluster, no partition is deployed on the new hosts. You can run `SHOW HOSTS` to check the partition distribution.

```
nebual> SHOW HOSTS;
+-------------+------+----------+--------------+----------------------------------+------------------------+
| Host        | Port | Status   | Leader count | Leader distribution              | Partition distribution |
+-------------+------+----------+--------------+----------------------------------+------------------------+
| "storaged0" | 9779 | "ONLINE" | 4            | "basketballplayer:4"             | "basketballplayer:15"  |
+-------------+------+----------+--------------+----------------------------------+------------------------+
| "storaged1" | 9779 | "ONLINE" | 8            | "basketballplayer:8"             | "basketballplayer:15"  |
+-------------+------+----------+--------------+----------------------------------+------------------------+
| "storaged2" | 9779 | "ONLINE" | 3            | "basketballplayer:3"             | "basketballplayer:15"  |
+-------------+------+----------+--------------+----------------------------------+------------------------+
| "storaged3" | 9779 | "ONLINE" | 0            | "No valid partition"             | "No valid partition"   |
+-------------+------+----------+--------------+----------------------------------+------------------------+
| "storaged4" | 9779 | "ONLINE" | 0            | "No valid partition"             | "No valid partition"   |
+-------------+------+----------+--------------+----------------------------------+------------------------+
| "Total"     |      |          | 15           | "basketballplayer:15"            | "basketballplayer:45"  |
+-------------+------+----------+--------------+----------------------------------+------------------------+
Got 6 rows (time spent 1002/1780 us)
```

Run `BALANCE DATA` to start balancing the storage partitions. If the partitions are already balanced, `BALANCE DATA` fails.

```
nebula> BALANCE DATA;
+------------+
| ID         |
+------------+
| 1614237867 |
+------------+
Got 1 rows (time spent 3783/4533 us)
```

A BALANCE task ID is returned after running `BALANCE DATA`. Run `BALANCE DATA <balance_id>` to check the status of the `BALANCE` task.

```
nebula> BALANCE DATA 1614237867;
+--------------------------------------------------------------+-------------------+
| balanceId, spaceId:partId, src->dst                          | status            |
+--------------------------------------------------------------+-------------------+
| "[1614237867, 11:1, storaged1:9779->storaged3:9779]"         | "SUCCEEDED"       |
+--------------------------------------------------------------+-------------------+
| "[1614237867, 11:1, storaged2:9779->storaged4:9779]"         | "SUCCEEDED"       |
+--------------------------------------------------------------+-------------------+
| "[1614237867, 11:2, storaged1:9779->storaged3:9779]"         | "SUCCEEDED"       |
+--------------------------------------------------------------+-------------------+
...
+--------------------------------------------------------------+-------------------+
| "Total:22, Succeeded:22, Failed:0, In Progress:0, Invalid:0" | 100               |
+--------------------------------------------------------------+-------------------+
Got 23 rows (time spent 916/1528 us)
```

When all the subtasks succeed, the load balancing process finishes. Run `SHOW HOSTS` again to make sure the partition distribution is balanced.

> **Note**
>
> `BALANCE DATA` does not balance the leader distribution.

```
nebula> SHOW HOSTS;
+-------------+------+----------+--------------+---------------------------------+------------------------+
| Host        | Port | Status   | Leader count | Leader distribution             | Partition distribution |
+-------------+------+----------+--------------+---------------------------------+------------------------+
| "storaged0" | 9779 | "ONLINE" | 4            | "basketballplayer:4"            | "basketballplayer:9"   |
+-------------+------+----------+--------------+---------------------------------+------------------------+
| "storaged1" | 9779 | "ONLINE" | 8            | "basketballplayer:8"            | "basketballplayer:9"   |
+-------------+------+----------+--------------+---------------------------------+------------------------+
| "storaged2" | 9779 | "ONLINE" | 3            | "basketballplayer:3"            | "basketballplayer:9"   |
+-------------+------+----------+--------------+---------------------------------+------------------------+
| "storaged3" | 9779 | "ONLINE" | 0            | "No valid partition"            | "basketballplayer:9"   |
+-------------+------+----------+--------------+---------------------------------+------------------------+
| "storaged4" | 9779 | "ONLINE" | 0            | "No valid partition"            | "basketballplayer:9"   |
+-------------+------+----------+--------------+---------------------------------+------------------------+
| "Total"     |      |          | 15           | "basketballplayer:15"           | "basketballplayer:45"  |
+-------------+------+----------+--------------+---------------------------------+------------------------+
Got 6 rows (time spent 849/1420 us)
```

If any subtask fails, run `BALANCE DATA` again to restart the balancing. If redoing load balancing does not solve the problem, ask for help in the Nebula Graph community.

9.2.3 Stop data balancing

To stop a balance task, run `BALANCE DATA STOP`.

- If no balance task is running, an error is returned.
- If a balance task is running, the task ID is returned.

`BALANCE DATA STOP` does not stop the running subtasks but cancels all follow-up subtasks. The running subtasks continue.

To check the status of the stopped balance task, run `BALANCE DATA <balance_id>`.

Once all the subtasks are finished or stopped, you can run `BALANCE DATA` again to balance the partitions again.

- If any subtask of the preceding balance task failed, Nebula Graph restarts the preceding balance task.
- If no subtask of the preceding balance task failed, Nebula Graph starts a new balance task.

9.2.4 Remove storage servers

To remove specific storage servers and scale in the Storage Service, use the `BALANCE DATA REMOVE <host_list>` syntax.

For example, to remove the following storage servers:

Server name	IP	Port
storage3	192.168.0.8	19779
storage4	192.168.0.9	19779

Run the following statement:

```
BALANCE DATA REMOVE 192.168.0.8:19779,192.168.0.9:19779;
```

Nebula Graph will start a balance task, migrate the storage partitions in storage3 and storage4, and then remove them from the cluster.

> **Note**
>
> The removed server's state will change to `OFFLINE`.

9.2.5 Balance leader distribution

`BALANCE DATA` only balances the partition distribution. If the raft leader distribution is not balanced, some of the leaders may overload. To load balance the raft leaders, run `BALANCE LEADER`.

```
nebula> BALANCE LEADER;
Execution succeeded (time spent 7576/8657 us)
```

Run `SHOW HOSTS` to check the balance result.

```
nebula> SHOW HOSTS;
+-------------+------+----------+--------------+---------------------------------+-----------------------+
| Host        | Port | Status   | Leader count | Leader distribution             | Partition distribution |
+-------------+------+----------+--------------+---------------------------------+-----------------------+
| "storaged0" | 9779 | "ONLINE" | 3            | "basketballplayer:3"            | "basketballplayer:9"   |
+-------------+------+----------+--------------+---------------------------------+-----------------------+
| "storaged1" | 9779 | "ONLINE" | 3            | "basketballplayer:3"            | "basketballplayer:9"   |
+-------------+------+----------+--------------+---------------------------------+-----------------------+
| "storaged2" | 9779 | "ONLINE" | 3            | "basketballplayer:3"            | "basketballplayer:9"   |
+-------------+------+----------+--------------+---------------------------------+-----------------------+
| "storaged3" | 9779 | "ONLINE" | 3            | "basketballplayer:3"            | "basketballplayer:9"   |
+-------------+------+----------+--------------+---------------------------------+-----------------------+
| "storaged4" | 9779 | "ONLINE" | 3            | "basketballplayer:3"            | "basketballplayer:9"   |
+-------------+------+----------+--------------+---------------------------------+-----------------------+
| "Total"     |      |          | 15           | "basketballplayer:15"           | "basketballplayer:45"  |
+-------------+------+----------+--------------+---------------------------------+-----------------------+
```

Last update: May 10, 2021

10. Ecosystem

10.1 Nebula Exchange

Nebula Exchange (hereinafter referred to as Exchange) is an Apache Spark™ application for migrating data into Nebula Graph from distributed systems. Exchange supports the migration of migrating batch data and stream data of different formats.

10.1.1 Use cases

Exchange applies to transforming the following data into vertices and edges in Nebula Graph:

- Stream data stored in Kafka or Pulsar, including Logs, online shopping records, online game player activities, social network information, financial trading data, and geospatial service data.
- Telemeasuring data recorded by equipment connected to IDCs.
- Batch data stored in relational databases such as MySQL or distributed file systems such as HDFS.

10.1.2 Benefits

- Adaptable. Exchange supports importing data with many different formats and sources into the Nebula Graph for easy data migration.
- Supports SST import. Exchange can transform data from different sources into SST files for importing.

> **Note**
>
> SST import is only supported on Linux.

- Supports breakpoint continuous transmission. To save time and improve efficiency, Exchange can continue the data transmission after the transmission is stopped.

> **Note**
>
> For now, breakpoint continuous transmission is only supported when importing Neo4j data.

- Asynchronous operations. Exchange generates a writing statement and then sends it to the Graph Service for data insertion.
- Flexible. Exchange supports importing data with multiple tags and edge types that originated from different data formats or sources.
- Supports statistics. Exchange uses Apache Spark™ Accumulators to make statistics for successful and failed insertion operations.
- Easy to use. Exchange applies the Human-Optimized Config Object Notation (HOCON) format for configuration files. HOCON is object-oriented and easy to understand and use.

10.1.3 Data formats and origins

Exchange 2.0 can migrate data with the following formats or origins.

- Data stored in HDFS, including
  - Apache Parquet
  - Apache ORC
  - JSON
  - CSV
- Apache HBase™
- Data warehouse: Hive
- Graph database: Neo4j
- Relational database: MySQL
- Event streaming platform: Apache Kafka®
- Message publishing/subscribing platform: Apache Pulsar 2.4.5

Last update: April 28, 2021

11. Contribution

11.1 How to Contribute

11.1.1 Before you get started

File an issue on the github or forum

You are welcome to contribute any code or files to the project. But first we suggest you raise an issue on the github or on the forum to start a discussion with the community. Check through the topic for Github.

Sign the Contributor License Agreement (CLA)

What is CLA?

Here is the vesoft inc. Contributor License Agreement.

Click the **Sign in with GitHub to agree** button to sign the CLA.

If you have any question, send an email to `info@vesoft.com`.

11.1.2 Step 1: Fork in the github.com

The Nebula Graph project has many repositories. Take the graph engine repository for example:

1. Visit https://github.com/vesoft-inc/nebula-graph
2. Click the `Fork` button (top right) to establish an online fork.

11.1.3 Step 2: Clone Fork to Local Storage

Define a local working directory:

```
# Define your working directory
working_dir=$HOME/Workspace
```

Set `user` to match your Github profile name:

```
user={your Github profile name}
```

Create your clone:

```
mkdir -p $working_dir
cd $working_dir
git clone https://github.com/$user/nebula-graph.git
# the following is recommended
# or: git clone git@github.com:$user/nebula-graph.git

cd $working_dir/nebula
git remote add upstream https://github.com/vesoft-inc/nebula-graph.git
# or: git remote add upstream git@github.com:vesoft-inc/nebula-graph.git

# Never push to upstream master since you do not have write access.
git remote set-url --push upstream no_push

# Confirm that your remotes make sense:
# It should look like:
# origin    git@github.com:$(user)/nebula-graph.git (fetch)
# origin    git@github.com:$(user)/nebula-graph.git (push)
# upstream  https://github.com/vesoft-inc/nebula-graph (fetch)
# upstream  no_push (push)
git remote -v
```

Define a Pre-Commit Hook

Please link the **Nebula Graph** pre-commit hook into your `.git` directory.

This hook checks your commits for formatting, building, doc generation, etc.

```
cd $working_dir/nebula-graph/.git/hooks
ln -s $working_dir/nebula-graph/.linters/cpp/hooks/pre-commit.sh .
```

Sometimes, pre-commit hook can not be executable. You have to make it executable manually.

```
cd $working_dir/nebula-graph/.git/hooks
chmod +x pre-commit
```

11.1.4 Step 3: Branch

Get your local master up to date:

```
cd $working_dir/nebula-graph
git fetch upstream
git checkout master
git rebase upstream/master
```

Checkout a new branch from master:

```
git checkout -b myfeature
```

> **Note**
>
> Because your PR often consists of several commits, which might be squashed while being merged into upstream, we strongly suggest you open a separate topic branch to make your changes on. After merged, this topic branch could be just abandoned, thus you could synchronize your master branch with upstream easily with a rebase like above. Otherwise, if you commit your changes directly into master, maybe you must use a hard reset on the master branch, like:
>
> ```
> git fetch upstream
> git checkout master
> git reset --hard upstream/master
> git push --force origin master
> ```

11.1.5 Step 4: Develop

Code Style

We adopt `cpplint` to make sure that the project conforms to Google's coding style guides. The checker will be implemented before the code is committed.

Unit Tests Required

Please add unit tests for your new features or bug fixes.

Build Your Code with Unit Tests Enable

Please refer to the build source code documentation to compile.

> Make sure you have enabled the build of unit tests by setting `-DENABLE_TESTING=ON`.

Run Tests

In the root folder of `nebula-graph`, run the following command:

```
ctest -j$(nproc)
```

11.1.6 Step 5: Bring Your Branch Update to Date

```
# While on your myfeature branch.
git fetch upstream
git rebase upstream/master
```

You need to bring the head branch up to date after other collaborators merge pull requests to the base branch.

11.1.7 Step 6: Commit

Commit your changes.

```
git commit -a
```

Likely you'll go back and edit/build/test some more than `--amend` in a few cycles.

11.1.8 Step 7: Push

When ready to review (or just to establish an offsite backup or your work), push your branch to your fork on `github.com`:

```
git push origin myfeature
```

11.1.9 Step 8: Create a Pull Request

1. Visit your fork at https://github.com/$user/nebula-graph (replace `$user` obviously).
2. Click the `Compare & pull request` button next to your `myfeature` branch.

11.1.10 Step 9: Get a Code Review

Once your pull request has been opened, it will be assigned to at least two reviewers. Those reviewers will do a thorough code review to make sure that the changes meet the repository's contributing guidelines and other quality standards.

Last update: April 22, 2021

https://docs.nebula-graph.io/2.0.1