Neo4j 4.0 MR2 Documentation

Last Update: Monday 19 August 2019

Safe Harbour Disclaimer

The features available in Neo4j 4.0 MR2 are experimental and they are likely to change in future versions of Neo4j: please consider the nature of the current implementation of these features when you use them.

Neo4j reserves the right to change features and product plans at any time, without obligation to notify any person of such changes.

1. Introduction

Neo4j 4.0 MR2 is the second Milestone Release of the new major version of the world's leading graph database. This release provides exciting new features that can be tested by installing and trying this early release.

IMPORTANT Neo4j 4.0 MR2 should not be used in production, nor should users expect production-quality features. In testing this release you may encounter technical issues that can corrupt your data.

1.1. What is a Milestone Release?

A Milestone Release - MR in short - is a very early stage release that provides only a limited subset of features that will appear in a future Generally Available (GA) release. From a quality standpoint, an MR can be considered a pre-alpha release, therefore performance and stability issues may appear and should be reported to Neo4j.

1.2. Availability and Features

Neo4j 4.0 MR2 is an Enterprise Edition release and it is offered as-is, binaries only. Community Edition code and binaries will not be available: we will publish Community Edition in line with the normal release cycle (Alpha > Beta > Release Candidate > Generally Available release).

The table below shows the features introduced in Neo4j 4.0 MR2:

Feature	Description	Community Edition	Enterprise Edition
Multiple Databases	Ability to create and use more than one database active at the same time.	One user database and one system (metadata) database	Multiple user databases and one system (metadata) database
Reactive Drivers	Client drivers providing reactive streams capabilities. JavaScript and Java driver only in MR2.	Available	Available

Back Pressure and Flow Control	Client drivers benefitting from server-side flow control, result set may be temporarily kept in the server and retrieved in chunks, synchronously or asynchronously. JavaScript and Java driver only in MR2.	Available	Available
Spring Boot	New starter, used to configure the Neo4j Java Driver within a Spring Boot.	Available	Available
SDN/RX	Spring Data Neo4j based on reactive streams.	Available	Available
Role-Based Access Control for Graphs	New RBAC security, applicable to graphs and graph elements.	Not available	Available
User Management	Administration commands used to manage users.	Available, without user suspend/activate	Available
Role Management for Graphs	Administration commands used to manage roles and association of privileges to roles and roles to users.	Not available	Available
Role Management for Databases	Administrative commands used to manage the database access rights associated with roles.	Not available	Available
Index Population Algorithm	New improved algorithm for the Native Index Provider (GB+Tree indexes).	Available	Available
Index Key Size	Increased Max Index Key Size (~8KB) for the Native Index Provider (GB+Tree).	Available	Available
Improved Space Reclaim	The ID store has been re-engineered to remove corner cases where space reclaim after the deletion of graph elements was suboptimal. The new approach benefits both standalone and clustered systems.	Available	Available

1.3. System Requirements

Neo4j 4.0 MR2 requires this minimal physical or virtual hardware requirements:

- 64-bit Intel-based CPU architecture (x86_64)
- 2 GByte RAM
- 10 GByte disk space

MR2 has been tested on the following operating systems:

- Ubuntu 18.04
- RedHat and Centos 7
- Windows Server 2016
- Mac OS 10.14

Neo4j 4.0 MR2 requires Java 11; tests have been performed using OpenJDK 11 and Oracle Java 11. This requirement is also applicable to Cypher Shell, which is available with the server.

The client Java driver can be executed using Java 8 or Java 11.

1.4. Licensing

Neo4j 4.0 MR2 is released under the Neo4j Pre-Release Agreement for Neo4j Software. Under the agreement, users are entitled to use the Milestone Release for a period of thirty days.

1.5. How to Provide Feedback

The main objective of the Milestone Release is to collect feedback from users and improve the experience of the future GA release.
You can provide feedback by using the Neo4j Community Forum - https://community.neo4j.com/c/neo4j-graph-platform.

1.6. How to Submit Issues

We warmly welcome any report of bugs and technical issues of the Milestone Release.
Although MR2 is not officially supported, issues may be reported on the Neo4j public repository - https://github.com/neo4j/neo4j/issues.
if you are a Neo4j customer, you may add a new ticket via the customer support portal.

1.7. Further Information and Documentation

Further information that is not covered in this document may be found in the Neo4j 3.5 manuals:

- Neo4j Operations Manual
- Neo4j Cypher Manual
- Neo4j Drivers Manual

2. Installation

2.1. Neo4j 4.0 MR2 Binaries

The binaries of Neo4j 4.0 MR2 are available on the Neo4j website, in the Downloads section: https://neo4j.com/download

In addition, these are the links to other binaries:

- **Cypher Shell for CentOS 7 and RedHat 7**: http://dist.neo4j.org/cypher-shell-1.2.0-0.alpha03.1.noarch.rpm
- **Cypher Shell for Debian 9 and Ubuntu 18.04**: http://dist.neo4j.org/cypher-shell_1.2.0.alpha03_all.deb
- **DEB package for Debian 9 and Ubuntu 18.04**: http://dist.neo4j.org/neo4j-enterprise_4.0.0_alpha09mr02_all.deb
- **RPM package for CentOS 7 and RedHat 7**: http://yum.neo4j.org/testing/neo4j-enterprise-4.0.0-0.alpha09mr02.1.noarch.rpm
- **Docker image**: http://dist.neo4j.org/neo4j-enterprise-4.0.0-alpha09mr02-docker-complete.tar

2.2. DEB Installation

DEB packages can be used for Debian 9 and Ubuntu 18.04 environments.

2.2.1. Prerequisites

Neo4j 4.0 MR2 requires Java 11 Runtime and the `daemon` package. You can update your testing environment with the following commands:

```
sudo apt install openjdk-11-jre
sudo apt install daemon
```

2.2.2. DEB Packages

Neo4j 4.0 MR2 comes in two Debian packages, and must be installed in this order: **Cypher Shell** and **Neo4j Server**.

First, you need to add the testing repository to the list of the available source for packages:

```
echo 'deb https://debian.neo4j.org/repo testing/' | sudo tee -a /etc/apt/sources.list.d/neo4j.list
sudo apt-get update
```

Then you can install the packages:

```
sudo apt-get install cypher-shell=1.2.0~alpha04
sudo apt-get install neo4j=1:4.0.0~alpha09mr02
```

2.2.3. Further Checks

Check the number of open file descriptors available for your process using the command `ulimit -n`. The number should be greater than 60000.
For further information regarding this limit and how to increase it, you can follow this link.

2.3. RPM Installation

RPM archives can be used for CentOS 7 and RedHat 7 environments.

2.3.1. Prerequisites

Neo4j 4.0 MR2 requires Java 11 Runtime. You can update your testing environment with the following command:

```
sudo yum install java-11-openjdk-devel
```

2.3.2. RPM Archives

Neo4j 4.0 MR2 comes in two RedHat archives, and must be installed in this order: **Cypher Shell** and **Neo4j Server**.

First, you need to add the testing repository to the list of the available source for packages:

```
cd /tmp
wget http://debian.neo4j.org/neotechnology.gpg.key
rpm --import neotechnology.gpg.key
cat <<EOF>  /etc/yum.repos.d/neo4j.repo
[neo4j]
name=Neo4j Yum Repo
baseurl=http://yum.neo4j.org/testing
enabled=1
gpgcheck=1
EOF
```

Then you can install the packages:

```
yum install cypher-shell-1.2.0-0.alpha04.1
yum install neo4j-4.0.0-0.alpha09mr02.1
```

2.3.3. Further Checks

Check the number of open file descriptors available for your process using the command `ulimit -n`. The number should be greater than 60000.
For further information regarding this limit and how to increase it, you can follow this link.

2.4. Tarball Installation

2.4.1. Prerequisites

Neo4j 4.0 MR2 requires Java 11 Runtime.

In Debian and Ubuntu environments, you can update your testing environment with the command:

```
sudo apt install openjdk-11-jre
```

In CentOS and RedHat environments, you can use this command:

```
sudo yum install java-11-openjdk-devel
```

In macOS you have different ways to install Java. One option is to download Java from the official Java Website as a tarball, extract the content of the archive and finally set the `PATH` and `JAVA_HOME` environment variables make the environment available for Neo4j.
Alternatively, you can accept the license agreement, download and install a Java Mac OS package from the official Oracle Java website.

2.4.2. Tarball Installation

Once you have located the tarball in your test environment, extract the content of the archives with the `tar -xzvf` command.

For example:

```
tar xzvf neo4j-enterprise-4.0.0-alpha09mr02-unix.tar.gz
```

2.4.3. Further Checks

Check the number of open file descriptors available for your process using the command `ulimit -n`. The number should be greater than 60000. For further information regarding this limit and how to increase it, you can follow this link.

The last check is Java; you may want to verify if you have access to the Java 11 environment.
One of the ways to verify the current version of Java is by running the `java -version` command:

```
$ java -version
openjdk version "11.0.3" 2019-04-16
OpenJDK Runtime Environment (build 11.0.3+7-Ubuntu-1ubuntu218.04.1)
OpenJDK 64-Bit Server VM (build 11.0.3+7-Ubuntu-1ubuntu218.04.1, mixed mode, sharing)
```

If the `java` command returns a different version (for example Java 8) it is likely that your testing environment has another version of Java pre-installed. You should check the position of the new Java 11 and set `PATH` and `JAVA_HOME` environment variables accordingly.

2.5. Windows Installation

The Windows ZIP file can be tested on all the 64 bit server and desktop versions of Windows.

As a prerequisite, install the Java Runtime JRE 11 on Windows and make sure that `java.exe` is available through the global `PATH`.

The next step is to unarchive the ZIP file in your chosen location, then follow the instructions provided here.

2.6. Docker Installation

The TAR file contains the docker image to install in your docker environment.

Once you have located your archive, load the image with the following command (you may need to be superuser):

```
docker load < neo4j-enterprise-4.0.0-alpha09mr02-docker-complete.tar
```

Next, check the image id of Neo4j 4.0 MR2 with the `docker image ls` command. For example:

```
$ sudo docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
neo4j-enterprise    4.0.0-alpha09mr02   1e666dc35122        8 days ago          646MB
```

Create a data and a log directory that will be used by the container and finally launch the image:

```
$ mkdir neo4j
$ cd neo4j
$ mkdir logs
$ mkdir data
$ docker run --publish=7474:7474 \
  --publish=7687:7687 \
  --volume=$HOME/neo4j/data:/data \
  --volume=$HOME/neo4j/logs:/logs \
  --env=NEO4J_ACCEPT_LICENSE_AGREEMENT=yes \
  1e666dc35122
Warning: Folder mounted to "/logs" is not writable from inside container. Changing folder owner to neo4j.
Warning: Folder mounted to "/data" is not writable from inside container. Changing folder owner to neo4j.
Directories in use:
  home:         /var/lib/neo4j
  config:       /var/lib/neo4j/conf
  logs:         /logs
  plugins:      /var/lib/neo4j/plugins
  import:       /var/lib/neo4j/import
  data:         /var/lib/neo4j/data
  certificates: /var/lib/neo4j/certificates
  run:          /var/lib/neo4j/run
Starting Neo4j.
2019-07-11 21:21:46.140+0000 INFO  ======== Neo4j 4.0.0-alpha09mr02 ========
2019-07-11 21:21:46.155+0000 INFO  Starting…
```

```
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.eclipse.collections.impl.utility.ArrayListIterate
(file:/var/lib/neo4j/lib/eclipse-collections-9.2.0.jar) to field java.util.ArrayList.elementData
WARNING: Please consider reporting this to the maintainers of
org.eclipse.collections.impl.utility.ArrayListIterate
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
2019-07-11 21:21:55.655+0000 INFO  Sending metrics to CSV file at /var/lib/neo4j/metrics
2019-07-11 21:21:56.926+0000 INFO  Bolt enabled on 0.0.0.0:7687.
2019-07-11 21:21:56.927+0000 INFO  Started.
2019-07-11 21:21:57.139+0000 INFO  Mounted REST API at: /db/manage
2019-07-11 21:21:57.315+0000 INFO  Server thread metrics have been registered successfully
2019-07-11 21:21:58.873+0000 INFO  Remote interface available at http://localhost:7474/
```

2.7. Post-Installation

2.7.1. Setting the Initial Password

As for the previous versions of Neo4j, Neo4j 4.0 MR2 requires to set an initial password for the `neo4j` user. This can be achieved by executing the `neo4j-admin` script in the `bin` directory:

```
bin/neo4j-admin set-initial-password mysecretpassword
```

2.7.2. Starting Neo4j

If you have installed Neo4j from a tarball, locate the installed directory. From the Neo4j home directory, you can start the server using the `neo4j` script in the `bin` directory:

```
bin/neo4j start
```

If you have installed a Neo4j package (Debian or RedHat), Neo4j 4.0 MR2 has been set as a service and it can be managed using the `systemctl` command. Follow this link to manage Neo4j as a service.

3. Managing Databases

Neo4j 4.0 offers new database functionalities. Up to Neo4j 3.5, database administrators (DBAs) could create multiple databases, but only one database was active at startup. In this new version, we can have multiple databases active and accessible at the same time.

3.1. Concepts

Before we present the new features of Neo4j 4.0 with regards to databases, it is important to define some terms and basic concepts.

3.1.1. DBMS, Servers, Instances

Neo4j is a *DataBase Management System* - DBMS in short. A DBMS can be a single standalone server or a group of servers (for example in a Causal Cluster).

A *Neo4j instance* is a Java process that is running the Neo4j server code.

3.1.2. Transaction Domain and Execution Context

A *transaction domain* is a collection of graphs that can be updated within the context of a single transaction.

An *execution context* is a runtime environment for the execution of a request. In practical terms, a request may be a query, a transaction or an internal function or procedure.

3.1.3. Database

A *database* is an administrative partition of a DBMS. In practical terms, it is a *physical structure of files* organised within a directory or folder that has the same name of the database. In logical terms, a database is a container for one or more graphs. In MR2, a database is a container for one graph only.

A database defines a *transaction domain* and an *execution context*. This means that a transaction cannot span across multiple databases. Similarly, a procedure is called *within* a database, although its logic may access data that is stored in other databases.

3.1.4. Databases and DBMS

A DBMS has at least two databases:

- A database containing Neo4j metadata - this has a non-configurable name of `system`.
- A database for user data - the default name for this database is `neo4j`.

3.1.5. Separation of Structure

Databases are structurally separated, in that the database files for each database are stored in a separate folder. In MR2, relationships cannot be defined between nodes in different databases and graphs cannot span across multiple databases.

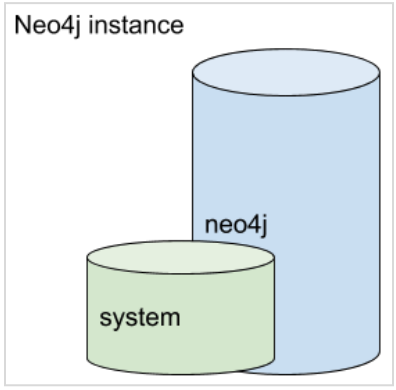
3.1.6. Rules for Database Names

The naming rules for databases are designed to be file system friendly, and cloud friendly:

- Length must be between 3 and 63 characters.
- The first character of a name must be an ASCII alphabetic character.
- Subsequent characters must be ASCII alphabetic or numeric characters, dots or dashes: [a..z][0..9].-
- Names are case-insensitive, and normalized to lowercase.
- Names that begin with an underscore and with the prefix system are reserved for internal use.

3.1.7. Community Edition

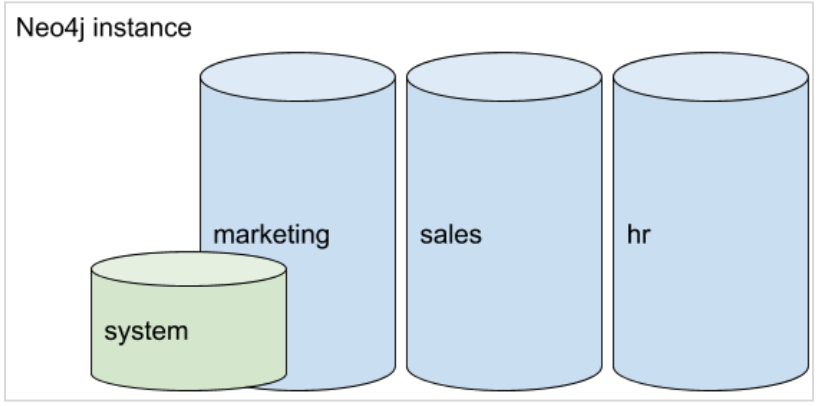
The DBMS can manage one database for user data. The system database is present in each instance.



A default Neo4j installation with Community Edition

3.1.8. Enterprise Edition

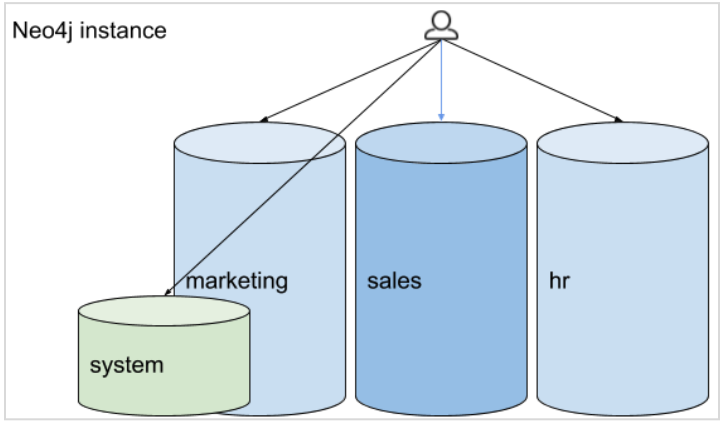
The DBMS can manage multiple databases. The system database is present in each instance.



A Neo4j installation with Enterprise Edition, managing three different databases with the names *marketing*, *sales* and *hr.*

3.1.9. Default database

Each Neo4j instance has a default database. If a user connects to Neo4j without specifying a database, it will connect to the default database.



A Neo4j instance with three databases. The sales database is the default database.

3.2. Configuration Parameters

Configuration parameters are defined in the *neo4j.conf* file.

The following configuration parameters are applicable for managing databases:

Parameter Name	Description	Default Value
`dbms.default_database`	Name of the default database for the Neo4j instance. The database is created if it does not exist when the instance starts.	`neo4j`
`dbms.max_databases`	Maximum number of databases that can be used in a Neo4j instance or Neo4j Cluster. The number includes all the online and offline databases. The value is an integer with a minimum value of 2	`100`

3.2.1. Examples

The following example illustrates how to specify a name for the default database:

```
dbms.default_database=sales
```

The following example illustrates how to specify a maximum number of databases for this instance:

```
dbms.max_databases=100
```

3.3. Commands for Managing Databases

Neo4j 4.0 MR2 offers a new set of Cypher administrative commands, which include commands to manage databases.

All administrative commands must be executed against the `system` database.

3.3.1 Switching Databases

The `:use` command is a client command used in Cypher Shell and Neo4j Browser to initiate a session to a database. All subsequent commands are executed against that database. If the `:use` command is executed without a database name, it initiates a session to the default database.

The `:use` command used with a database name:

```
neo4j@neo4j>
neo4j@neo4j> :use system
neo4j@system>
```

The `:use` command used without a database name:

```
neo4j@system>
neo4j@system> :use
neo4j@neo4j>
```

The `:use` command can be unsuccessful, for instance because it cannot connect to the given database.

NOTE If using Cypher Shell in interactive mode, the connection to the previous database will be intact. In non-interactive mode, i.e. when executing a Cypher script, the execution will fail. If the `--fail-at-end` flag is provided in non-interactive mode, however, Cypher Shell will be in a disconnected mode after a failed `:use` command. This behavior protects against inadvertently applying changes to the previous database.

Refer to the driver documentation in section 5.2 for instructions on how to select a database using client code.

3.3.2. Cypher Administrative Commands

The following Cypher administrative commands are used to manage databases:

Command	Description
`CREATE DATABASE` *name*	Create and start a new database
`DROP DATABASE` *name*	Drop (remove) an existing database **NOTE:** This command is not available in MR2.
`START DATABASE` *name*	Start a database that has been stopped
`STOP DATABASE` *name*	Shut down a database
`SHOW DATABASE` *name*	Show the status of a specific database

`SHOW DATABASES`	Show the name and status of all the databases
`SHOW DEFAULT DATABASE`	Show the name and status of the default database

The following series of commands demonstrate the usage of the Cypher commands for managing databases, using Cypher Shell. The examples assume that we have a standard installation containing the default database `neo4j` and that we are logged in as user `neo4j` in the `system` database. For instructions on changing databases, see section 3.3.0.

NOTE Remember that all administrative commands must be executed against the `system` database.

3.3.2.1. `SHOW DATABASE`

Show the status of database `neo4j`:

```
neo4j@system> SHOW DATABASE neo4j;
+------------------------------+
| name    | status   | default |
+------------------------------+
| "neo4j" | "online" | TRUE    |
+------------------------------+
```

3.3.2.2. `SHOW DATABASES`

Show the status of all databases:

```
neo4j@system> SHOW DATABASES;
+------------------------------+
| name     | status   | default |
+------------------------------+
| "neo4j"  | "online" | TRUE    |
| "system" | "online" | FALSE   |
+------------------------------+
```

The possible values for the status are:

- Online - if the database is running.
- Offline - if the database is not running.

Switching between these states is achieved using the `START DATABASE` and `STOP DATABASE` commands described below.

3.3.2.3. `SHOW DEFAULT DATABASE`

The config setting `dbms.default_database` defines which database is created and started by default when Neo4j starts. The default value of this setting is `neo4j`. In Community Edition, this is the only available database (other than the system database, which is always there). In Enterprise Edition there can be more databases, but this is the one created automatically if it does not exist. The above commands listed the default database as a column with only one TRUE value. However it is also possible to show only the default database using the command:

```
neo4j@system> SHOW DEFAULT DATABASE;
+--------------------+
| name    | status   |
+--------------------+
| "neo4j" | "online" |
+--------------------+
```

Changing the default database requires editing the config setting `dbms.default_database` and restarting the server.

3.3.2.4. `CREATE DATABASE`

Create a database with the name `sales`:

```
neo4j@system> CREATE DATABASE sales;
+--------------------+
| name    | status   |
+--------------------+
| "sales" | "online" |
+--------------------+

1 row available after 58 ms, consumed after another 0 ms
Added 1 nodes, Set 4 properties, Added 1 labels
neo4j@system> SHOW DATABASES;
+-------------------------------+
| name     | status   | default |
+-------------------------------+
| "neo4j"  | "online" | TRUE    |
| "system" | "online" | FALSE   |
| "sales"  | "online" | FALSE   |
+-------------------------------+

3 rows available after 6 ms, consumed after another 0 ms
neo4j@system>
```

3.3.2.5. STOP DATABASE

Stop the database `sales` and try using it after stopping it:

```
neo4j@system> STOP DATABASE sales;
0 rows available after 18 ms, consumed after another 6 ms
neo4j@system> SHOW DATABASES;
+--------------------------------+
| name     | status    | default |
+--------------------------------+
| "neo4j"  | "online"  | TRUE    |
| "system" | "online"  | FALSE   |
| "sales"  | "offline" | FALSE   |
+--------------------------------+

3 rows available after 5 ms, consumed after another 1 ms
neo4j@system> :use sales
The database is not currently available to serve your request, refer to the database logs for more details.
Retrying your request at a later time may succeed.
neo4j@sales[UNAVAILABLE]>
```

3.3.2.6. START DATABASE

Start the database `sales` and try using it after starting it:

```
neo4j@sales[UNAVAILABLE]> :use system
neo4j@system>
neo4j@system> START DATABASE sales;
0 rows available after 21 ms, consumed after another 1 ms
neo4j@system> SHOW DATABASES;
+--------------------------------+
| name     | status   | default |
+--------------------------------+
| "neo4j"  | "online" | TRUE    |
```

```
| "system" | "online" | FALSE   |
| "sales"  | "online" | FALSE   |
+-------------------------------+

3 rows available after 5 ms, consumed after another 0 ms
neo4j@system> :use sales
neo4j@sales>
```

3.3.2.7. DROP DATABASE

Drop the database `sales`:

```
neo4j@sales> :use system
neo4j@system> DROP DATABASE sales;
0 rows available after 82 ms, consumed after another 1 ms
neo4j@system> SHOW DATABASES;
+-------------------------------+
| name     | status   | default |
+-------------------------------+
| "neo4j"  | "online" | TRUE    |
| "system" | "online" | FALSE   |
+-------------------------------+

2 rows available after 5 ms, consumed after another 1 ms
neo4j@system>
```

NOTE This command is not available in MR2.

3.4 Databases in Causal Cluster Installations

Causal Cluster provides multiple databases as if it is a single, standalone DBMS. Administrators can use the same Cypher commands described above to manage databases. This is based on two main principles:

1. All databases are available on all members of a cluster - this applies to core servers and read replicas.
2. Administrative commands must be executed from the `system` database on the `LEADER` member of the cluster - as for a standalone server, where Cypher commands to administer databases must be executed from the `system` database and they often change the content of the repository, similarly in a clustered environment an administrator must connect to the `LEADER` member to update the repository.

3.4.1 Executing Cypher Administrative Commands from Cypher Shell

Let's assume we have a Causal Cluster environment formed by 5 members, 3 core servers and 2 read replicas:

```
neo4j@neo4j> CALL dbms.cluster.overview();
+--------------------------------------------------------------------------------------------------------------------------------------------------+
| id        | addresses                                                                  | databases                                            | groups |
+--------------------------------------------------------------------------------------------------------------------------------------------------+
| "8c...3d" | ["bolt://localhost:7683", "http://localhost:7473", "https://localhost:7483"] | {neo4j: "FOLLOWER", system: "FOLLOWER"}           | []     |
| "8f...28" | ["bolt://localhost:7681", "http://localhost:7471", "https://localhost:7481"] | {neo4j: "LEADER", system: "LEADER"}               | []     |
| "e0...4d" | ["bolt://localhost:7684", "http://localhost:7474", "https://localhost:7484"] | {neo4j: "READ_REPLICA", system: "READ_REPLICA"} | []     |
| "1a...64" | ["bolt://localhost:7682", "http://localhost:7472", "https://localhost:7482"] | {neo4j: "FOLLOWER", system: "FOLLOWER"}           | []     |
| "59...87" | ["bolt://localhost:7685", "http://localhost:7475", "https://localhost:7485"] | {neo4j: "READ_REPLICA", system: "READ_REPLICA"} | []     |
+--------------------------------------------------------------------------------------------------------------------------------------------------+

5 rows available after 5 ms, consumed after another 0 ms
```

The leader is currently the instance exposing port *7681* for the `bolt` protocol and *7471/7481* for the `http/https` protocol.

Administrators can connect and execute Cypher commands in two ways:

1. Using the `bolt://` scheme to connect to the `LEADER`. For example, considering the cluster described above:

```
$ bin/cypher-shell -a bolt://localhost:7681 -d system -u neo4j -p neo4j1
Connected to Neo4j 4.0.0 at bolt://localhost:7681 as user neo4j.
Type :help for a list of available commands or :exit to exit the shell.
Note that Cypher queries must end with a semicolon.
neo4j@system> SHOW DATABASES;
+------------------------------+
| name     | status   | default |
+------------------------------+
| "neo4j"  | "online" | TRUE    |
| "system" | "online" | FALSE   |
+------------------------------+

2 rows available after 34 ms, consumed after another 0 ms
neo4j@system> CREATE DATABASE data001;
0 rows available after 378 ms, consumed after another 12 ms
Added 1 nodes, Set 4 properties, Added 1 labels
neo4j@system> SHOW DATABASES;
+------------------------------+
| name      | status   | default |
+------------------------------+
| "neo4j"   | "online" | TRUE    |
| "system"  | "online" | FALSE   |
| "data001" | "online" | FALSE   |
+------------------------------+

3 rows available after 2 ms, consumed after another 1 ms
neo4j@system>
```

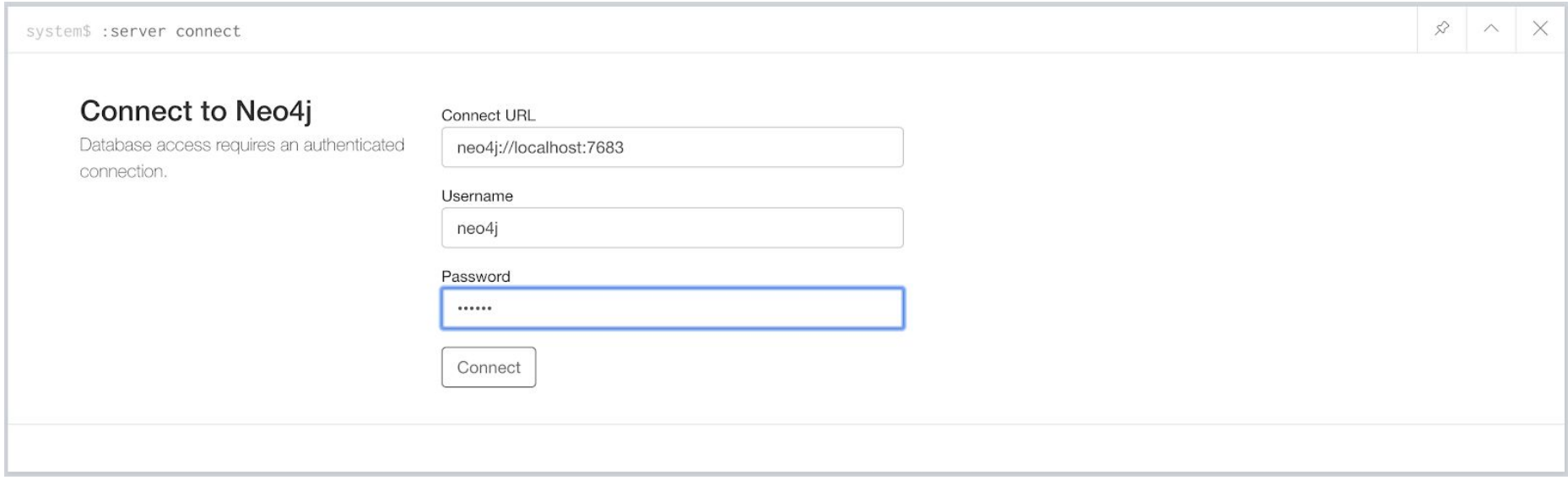
2. Using the `neo4j://` scheme to connect to any core member. The scheme, new in Neo4j 4.0, is equivalent to the `bolt+routing:` scheme available in Neo4j 3.5, but it can be used seamlessly with standalone and clustered DBMS.
   Again, considering the cluster described above:

```
$ bin/cypher-shell -a neo4j://localhost:7683 -d system -u neo4j -p neo4j1
Connected to Neo4j 4.0.0 at neo4j://localhost:7683 as user neo4j.
Type :help for a list of available commands or :exit to exit the shell.
Note that Cypher queries must end with a semicolon.
neo4j@system> SHOW DATABASES;
+--------------------------------+
| name      | status   | default |
+--------------------------------+
| "neo4j"   | "online" | TRUE    |
| "system"  | "online" | FALSE   |
| "data001" | "online" | FALSE   |
+--------------------------------+

3 rows available after 0 ms, consumed after another 0 ms
neo4j@system> CREATE DATABASE data002;
0 rows available after 8 ms, consumed after another 1 ms
Added 1 nodes, Set 4 properties, Added 1 labels
neo4j@system> SHOW DATABASES;
+--------------------------------+
| name      | status   | default |
+--------------------------------+
| "neo4j"   | "online" | TRUE    |
| "system"  | "online" | FALSE   |
| "data001" | "online" | FALSE   |
| "data002" | "online" | FALSE   |
+--------------------------------+

4 rows available after 33 ms, consumed after another 0 ms
neo4j@system>
```

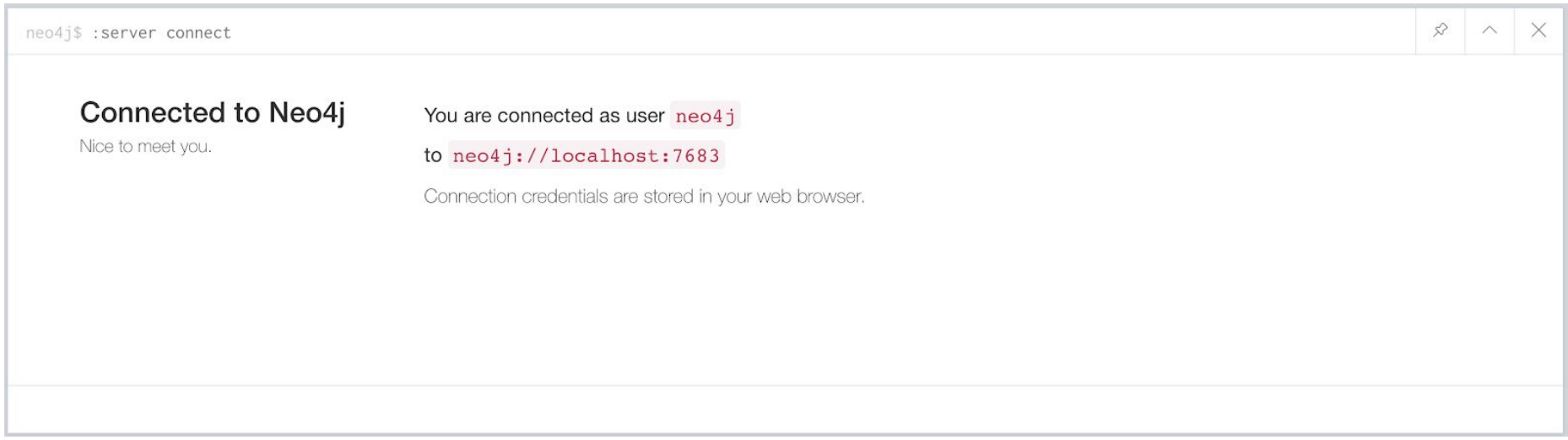
3.4.2 Executing Cypher Administrative Commands from Neo4j Browser

Assuming the same clustering structure presented in the previous paragraph, administators can use a similar approach to connect and execute Cypher commands:

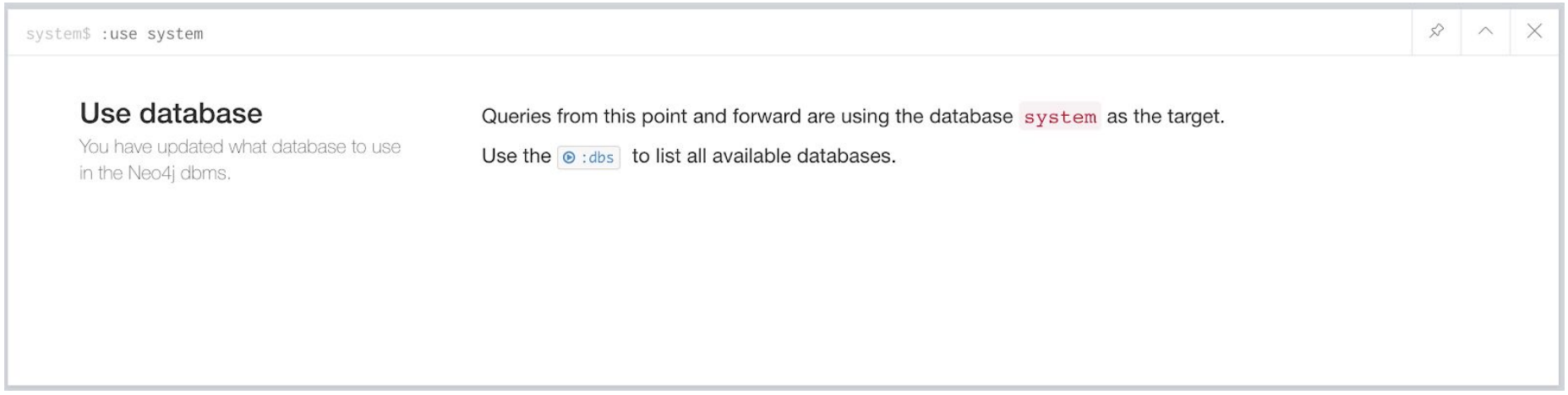
1. Opening Neo4j Browser, with the `bolt://` scheme, it is first important to identify the `LEADER` server, then connect and select the `system` database.
2. Using the `neo4j://` scheme, administrators can connect to any core member. Here is a series of screenshots:
    a. Connect to a Browser instance using the `neo4j://` scheme:



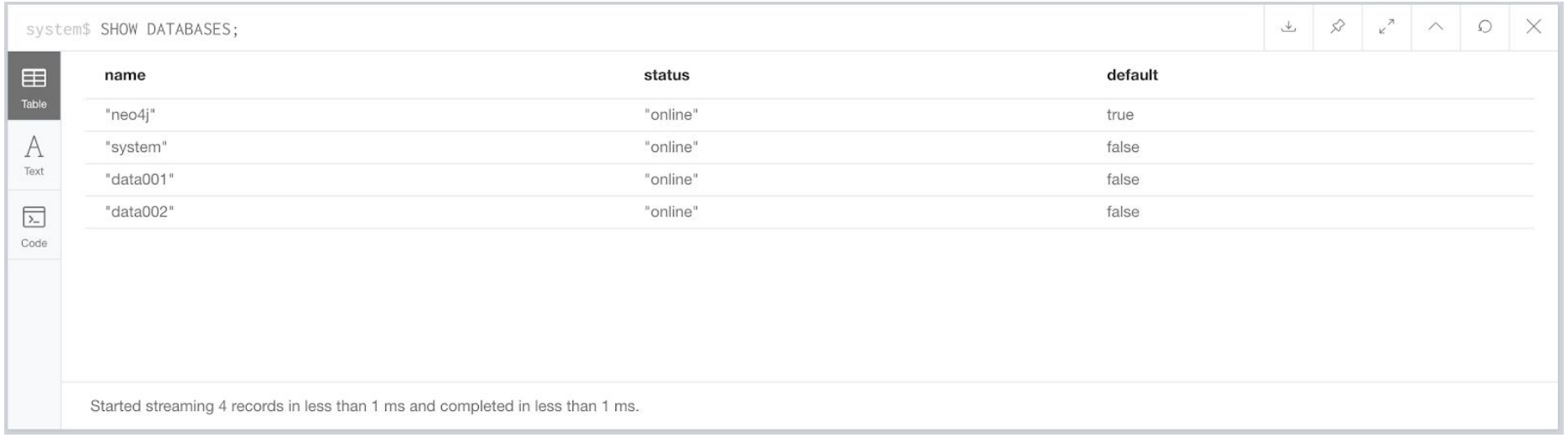
b. Neo4j Browser shows the current connection:

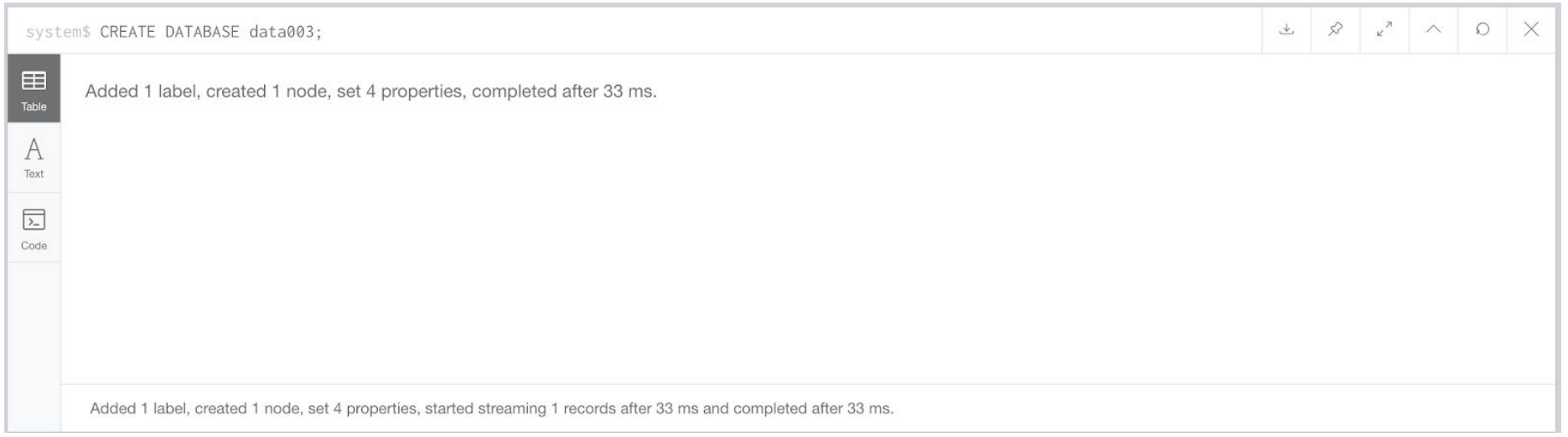


c. Select the `system` database:



d. Execute administrative commands:





4. Security Features

Neo4j 4.0 MR2 provides a sophisticated set of security features that are used to control user access to databases and graphs. The new features fall under one of the three categories below:

- **Schema-based security** - the ability to grant or deny specific access to an object (a graph or a graph element) with a given label, type or property.
- **User Management** - the ability to manage users in terms of authentication and authorization.
- **Role Management** - the ability to assign privileges to a role and associate a role to one or more users.

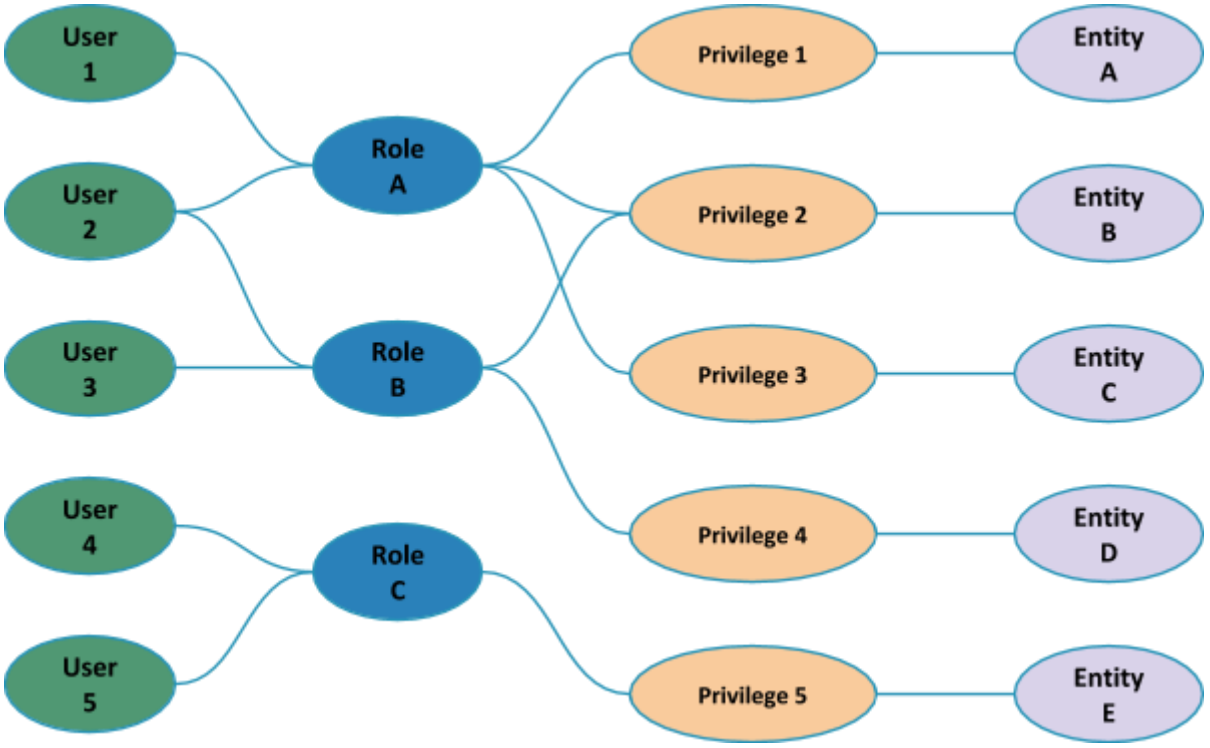
4.1. Role-Based Access Control and Fine-Grained Security

Security in Neo4j 4.0 is based on lists of **privileges** that grant or deny **access** on an **object** to a **grantee**. The grantee is a **role**, i.e. an abstract element that groups a list of grants and denies; a user can be associated to one or more roles, acquiring the lists associated to the roles.

This security approach goes under the name of **Role-Based Access Control**, and it is commonly used in DBMSs. In Neo4j 4.0, the security process is:

1. A DBA identifies an **entity** (a graph or a graph element).
2. The access to the entity is associated with a **granted privilege.**
3. The privilege is associated with one or more **roles.**
4. The role is associated with one or more **users.**

The figure below shows the relationship between users and roles, roles and privileges, privileges and entities.



NOTE As with all other Cypher administrative commands, granting and denying privileges to a role as well as assigning roles to users, must be done in the `system` database. Granted/denied privileges and the corresponding roles can then be made available to ALL databases, unless otherwise specified.

Neo4j 4.0 provides a fine level of grants over the graphs that be accessed by a user. Administrators can grant or deny access in reading and writing (creating, modifying and removing objects), for graphs, graph elements (i.e. nodes and relationships) and for properties within elements. This approach goes under the name of **Fine-Grained Security**, since administrator may grant or deny access to a user on an individual property in an individual graph element.

4.1.1. Entities

'Entity' is a generic term to identify an object or a component in a DBMS. In Neo4j 4.0 MR2, we control access to the following entities:

- **Graphs** - A persistently stored graph structure. In Neo4j 4.0 MR2 there is only one graph per database and commands can refer to that graph using the database name. This graph can be understood to be the set of all graph elements in the database.
- **Graph Elements** - the components of a graph:
  - **Nodes** - An object in a graph, used to store different kinds of data, such as information about a business entity for example.
  - **Relationships** - An object in the graph that connects two nodes, used to store information about how different kinds of data are related.

NOTE	In Neo4j 4.0 there is support for multiple databases, but not multiple graphs per database. This means that security commands that specify a graph by name are in fact referring to the single graph in the database by that name. In future versions with multiple graphs per database, it will be possible to be more specific in these security commands.

4.1.2. Privileges

The privileges granted to a role and the role granted to a user, determine which operations a user can perform. The privileges available in Neo4j 4.0 MR2 are:

- `TRAVERSE` - This privilege provides the ability to find a graph element during the execution of a query. Graph elements are identified primarily by their label or relationship type.
  For a user with only this privilege, the query would not be able to read or write properties in the traversed object, but it would be able to find and use the object in order to traverse to other objects.
- `READ` - This privilege provides the ability to read the properties of graph elements. Please note that the `READ` privilege does not provide the ability to find the element.
- `MATCH` - This privilege provides the ability to both find and read the contents of elements. It is a combination of `TRAVERSE` and `READ`, and is designed as a convenient method for achieving the same combination necessary to execute some specific Cypher `MATCH` commands. When listing privileges with the `SHOW PRIVILEGES` command you will only be able to see the underlying find (`TRAVERSE`) and read (`READ`) privileges.
- `WRITE` - This privilege provides the ability to perform updates to the graph. In MR2 it is only possible to grant global write permissions, so users with this privilege will be able to perform all data write operations (creating nodes and relationships, setting properties, assigning labels and deleting and unsetting elements, properties and labels).

4.1.3. Grantees

A *grantee* is a role which has a privilege granted. If you wish to grant privileges to users, grant the appropriate roles to the users in question.

4.1.4. Granting and Denying Privileges

Privileges control the access rights to graph elements using a combined whitelist/blacklist mechanism. It is possible to grant access, deny access, or both. The user will be able to access a resource if they have a grant (whitelist) and do not have a deny (blacklist) relevant to that resource. If there are no read privileges provided at all, then the user will be denied access to the entire graph, and this will generate an error. All other combinations of `GRANT` and `DENY` will result in the matching subgraph being visible, which will appear to the user as if they have a smaller database (smaller graph).

4.1.4.1. Fine-Grained Security

The `GRANT` and `DENY` commands can provide a fine-grained level of security, allowing the DBA to control access to:

- Nodes identified by their label(s).
- Relationships identified by their relationship type.
- A subset of properties on nodes and relationships.
- A selected list of graphs.

When the DBA wants to limit the access to a selected list, they can specify the list in the `GRANT` command. Alternatively, an `*` in the command means that the grant is applied to all the properties, graphs or nodes. If they wish to restrict access to all but a selected list, they can use the `DENY` command.

NOTE

In a `GRANT` or `DENY` command:

- The property list appears as part of the *`graph-privilege`*.
- The graph and node lists appear as part of the *`entity`*.

4.1.4.2. The GRANT/DENY Commands

The GRANT command allows a DBA to grant a privilege to a role in order to access an entity. The DENY command allows a DBA to deny a privilege to a role in order to prevent access to an entity. The syntax is:

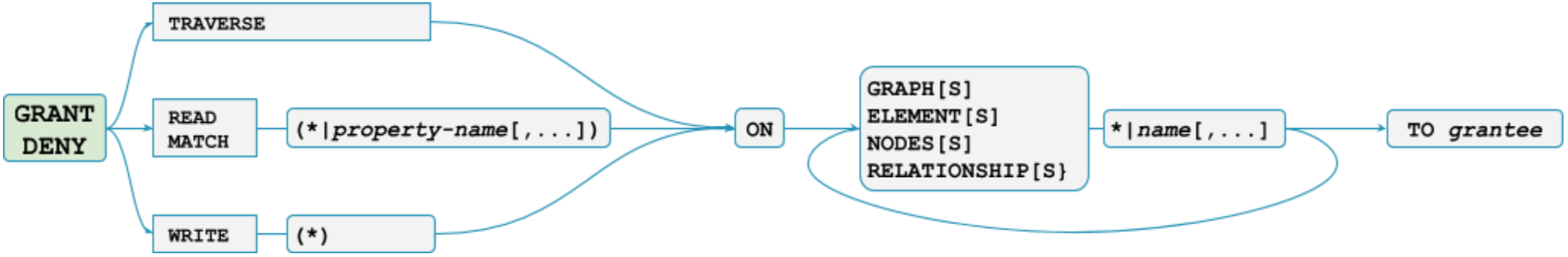
```
GRANT graph-privilege ON GRAPH graphname entity TO role
DENY  graph-privilege ON GRAPH graphname entity TO role
```

Where the components are:

- graph-privilege
  - The privilege that is being assigned. graph-privilege can have the following values:
    - TRAVERSE - allows the specified entities to be found.
    - READ (property) - allows the specified properties to be read on the found entities. Note that if the query used to find the entity requires reading the property, it will not be found. Multiple properties can be specified, comma-separated. property can be set to * which means *all properties*.
    - MATCH (property) - this combines both TRAVERSE and READ allowing an entity to be found and the specified properties read. Multiple properties can be specified, comma-separated.
    - WRITE (*) - in MR2 this privilege can only be assigned to all nodes, relationships and properties in the entire graph (this means that the entity part of the command must also be ELEMENTS * and cannot be more specific (yet)).
- graphname
  - The name of the graph or graphs to associate the privilege with. In Neo4j 4.0, each database has exactly one graph, and the graph name is the same as the database name. Note that if you delete a database and create a new one with the same name, the new database will not have any of the privileges specifically assigned to the original (deleted) graph.
  - Multiple graph names can be specified, comma-separated.
  - graphname can be * which means *all graphs*. Graphs associated to databases created after this command has been issued will be given these privileges.
- entity
  - The graph elements that this privilege applies to. entity can have the following values:
    - NODES label, where label specifies which node label(s) that are included in this privilege.
    - RELATIONSHIPS type , where type specifies which relationship type(s) that are included in this privilege.

        - `ELEMENTS {label | type}`, where `label` and `type` are the same as defined above.
    - Multiple labels and relationship types can be specified, comma-separated.
    - The `label` or `type` can be set to `*` which means *all labels* or *all relationship types*.
- `role`
    - The role or roles to associate the privilege with.

NOTE It is important to note that using `DENY` does NOT erase a `GRANT` command; they both exist. The only way to erase a privilege is with `REVOKE`.



`GRANT/DENY` syntax and options in MR2

4.1.4.3. `GRANT` Examples

The following examples assume that a DBA want to grant access on entities to the role `my_role`. (See section below on how to create a role.)

> **NOTE** As with all the other Cypher administrative commands, both `GRANT` and `DENY` must be executed in the `system` database.

`TRAVERSE` all nodes in all graphs:

```
neo4j@system> GRANT TRAVERSE ON GRAPHS * NODES * TO my_role;
```

`MATCH` all relationships in all graphs:

```
neo4j@system> GRANT MATCH (*) ON GRAPHS * RELATIONSHIPS * TO my_role;
```

`MATCH` all nodes in graph `neo4j`:

```
neo4j@system> GRANT MATCH (*) ON GRAPH neo4j NODES * TO my_role;
```

`MATCH` nodes with labels `labA` and `labB` in graph `neo4j`:

```
neo4j@system> GRANT MATCH (*) ON GRAPH neo4j NODES labA, labB TO my_role;
```

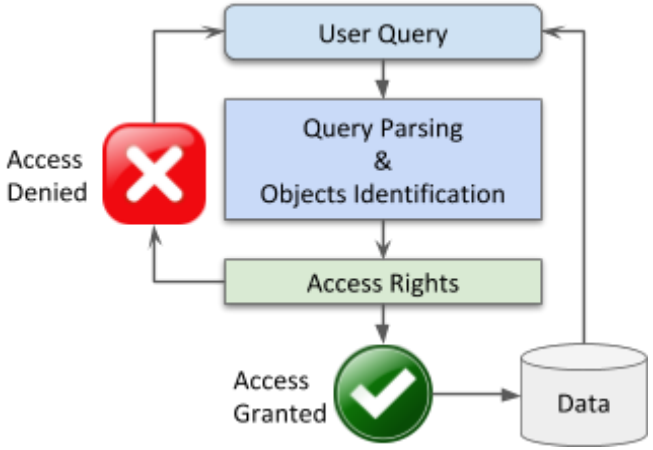
`MATCH` properties `p0` and `p1` in nodes with label *labA* and properties `p0`, `p1`, `p2` and `p3` in relationships with `labB`:

```
neo4j@system> GRANT MATCH (p0,p1) ON GRAPHS * NODES labA TO my_role;
neo4j@system> GRANT MATCH (p0,p1,p2,p3) ON GRAPHS * RELATIONSHIPS labB TO my_role;
```

The `DENY` command can be used in a similar fashion.

4.2 Security Model in Neo4j 4.0 vs. Other DBMSs

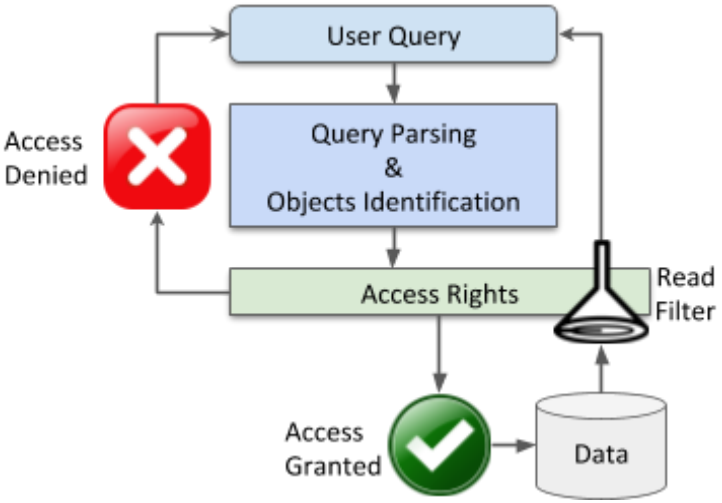
In many DBMSs, and more specifically with Relational DBMSs, a security layer is used to grant or deny access to a known entity. The layer provides a straightforward control over the access, where a granted access ends with a successful result returned to the user or to the application, and a denied access ends with an error. Furthermore, the layer is applied to the query by analyzing which objects are involved and which access the user is allowed against these objects.



Evaluating fine-grained security rules in relational DBMSs

A graph model and the nature of connected data requires a different approach to security. In simple terms, in Neo4j 4.0 the security layer is applied during the execution of the query and the behavior differs between *writes* (change or deletion of an existing object, or creation of a new object) and *reads*. With writes, a user has its access granted or denied, depending on the privilege, and the consequences are query success or error: this is similar to the grant or denied access of the security model of relational DBMSs.

With reads however, the access is evaluated for all the objects involved in the execution of the query and the results differ depending on which object can be accessed - or in other terms, the result is filtered depending on the privileges of the user. Instead of an error, a restricted user will see a subset of the graph. The only situation in which an error occurs during read-only queries is when the user has no read privileges at all.



Evaluating fine-grained security rules in Neo4j

4.3. User Management

Neo4j 4.0 MR2 provides new administration commands to manage users. In the Enterprise Edition versions of Neo4j 3.1 to 3.5, users were managed using procedures. Although these procedures still exist, they may be deprecated in the future and it is strongly advised that they must not be used in this release.

4.3.1. Cypher Commands for User Management

The following Cypher commands are available for managing users:

Command	Description	Type of user
`CREATE USER name SET PASSWORD password [[SET PASSWORD] CHANGE [NOT] REQUIRED] [SET STATUS {ACTIVE|SUSPENDED}]`	Create a new user	Admin
`DROP USER name`	Drop (remove) an existing user	Admin

`ALTER USER name SET {`
`  PASSWORD password [[SET PASSWORD] CHANGE [NOT] REQUIRED]`
`  [SET STATUS {ACTIVE | SUSPENDED} ]`
`  | PASSWORD CHANGE [NOT] REQUIRED [SET STATUS {ACTIVE |`
`SUSPENDED}]`
`  | STATUS {ACTIVE | SUSPENDED}}`
	Modify the settings for an existing user	Admin

`ALTER CURRENT USER SET PASSWORD FROM original TO password`	Change the logged in users password	Normal user
`SHOW USERS`	List all users	Admin
`SHOW USER name PRIVILEGES`	List the privileges granted to a user	Admin

NOTE Like all the other new administration commands, user management commands must be executed in the `system` database.

4.3.2. Creating a New User

`CREATE USER` is used to create a new user. The full syntax is:

```
CREATE USER name SET PASSWORD password [[SET PASSWORD] CHANGE [NOT] REQUIRED]
              [SET STATUS {ACTIVE|SUSPENDED}]
```

Arguments and options for the command are:

- *`name`* - The name of the user to create.
- `SET PASSWORD` *`password`* - An initial password needs to be set. It can either be a string enclosed in quotes or a parameter with a string value.
- `CHANGE REQUIRED` or `CHANGE NOT REQUIRED` - By default, the initial password must be modified on first login (`CHANGE REQUIRED`). With `CHANGE NOT REQUIRED`, the user can keep the initial password.

- `SET STATUS ACTIVE` or `SET STATUS SUSPENDED` - By default, the user is active, i.e. the user can immediately login. The DBA may suspend the user, preventing the login.

4.3.2.1. `CREATE USER` Example

This command creates the user `joe` with the password `soap`, and does not require a password change once they login:

```
neo4j@system> CREATE USER joe SET PASSWORD 'soap' CHANGE NOT REQUIRED;
```

4.3.3. Removing an Existing User

`DROP USER` is used to remove an existing user. The syntax is:

```
DROP USER name
```

4.3.4. Modifying the Settings of an Existing User

`ALTER USER` is used to modify an existing user. The full syntax is:

```
ALTER USER name SET {
                    PASSWORD password [[SET PASSWORD] CHANGE [NOT] REQUIRED] [SET STATUS {ACTIVE | SUSPENDED} ]
                    | PASSWORD CHANGE [NOT] REQUIRED [SET STATUS {ACTIVE | SUSPENDED}]
                    | STATUS {ACTIVE | SUSPENDED}}
```

At least one of the options needs to be set for this command and if an option is not set the users current value will be kept.

Arguments and options for the command are:

- *`name`* - The name of the user to modify
- `SET PASSWORD` *`password`* - A new password for the user. It can either be a string enclosed in quotes or a parameter with a string value.
- `CHANGE REQUIRED` or `CHANGE NOT REQUIRED` - This updates the flag for if the users password needs to be modified on next login (`CHANGE REQUIRED`) or if it can be kept (`CHANGE NOT REQUIRED`).

- `SET STATUS ACTIVE` or `SET STATUS SUSPENDED` - This updates the user's status, i.e. active, meaning that the user can login, or suspended, preventing the login.

4.3.4.1. `ALTER USER` Examples

This command changes the password for the user `joe` to `supersecret`, but if `joe` was suspended or needed to change his password on the next login that is still true:

```
neo4j@system> ALTER USER joe SET PASSWORD 'supersecret';
```

This command changes the password for the user `joe` and does not require him to change his password on the next login, however if he was suspended that is still true:

```
neo4j@system> ALTER USER joe SET PASSWORD 'supersecret' CHANGE NOT REQUIRED;
```

This command just updates that the user `joe` doesn't need to change his password on the next login, without changing his current password and if he was suspended that is still true:

```
neo4j@system> ALTER USER joe SET PASSWORD CHANGE NOT REQUIRED;
```

This command suspends the user `joe`, but he keeps his old password, and if he was previously required to change his password on the next login, that is still true:

```
neo4j@system> ALTER USER joe SET STATUS SUSPENDED;
```

This command activates the user `joe`, gives him a new password and makes sure he needs to change password on next login:

```
neo4j@system> ALTER USER joe SET PASSWORD 'changeme' CHANGE REQUIRED SET STATUS ACTIVE;
```

4.3.5. Listing All Users

`SHOW USERS` is used to provide a list of all users. The syntax is:

```
SHOW USERS
```

For example:

```
neo4j@system> SHOW USERS;
+-----------------------------------------------------------------------+
| user      | roles          | passwordChangeRequired | suspended |
+-----------------------------------------------------------------------+
| "neo4j"   | ["admin"]      | FALSE                  | FALSE     |
| "joe"     | []             | FALSE                  | FALSE     |
| "normal"  | ["user"]       | FALSE                  | FALSE     |
| "limited" | ["restricted"] | FALSE                  | FALSE     |
+-----------------------------------------------------------------------+
```

The command returns the following information:

- `user` - The username.
- `roles` - The roles assigned to the user.
- `passwordChangeRequired` - True if the user requires a password change on next login, otherwise false.
- `suspended` - True if the user is suspended, otherwise false.

4.3.6. Listing Privileges for a User

`SHOW USER PRIVILEGES` is used to provide a list of privileges for a given user. The syntax is explained in detail in *4.3.9. Listing privileges*:

For example:

```
neo4j@system> SHOW USER normal PRIVILEGES;
+-----------------------------------------------------------------------------------------+
| grant     | action | resource         | graph | segment           | role   | user     |
+-----------------------------------------------------------------------------------------+
| "DENIED"  | "read" | "property(ssn)"  | "*"   | "NODE(*)"         | "user" | "normal" |
| "GRANTED" | "read" | "all_properties" | "*"   | "NODE(*)"         | "user" | "normal" |
| "GRANTED" | "find" | "graph"          | "*"   | "NODE(*)"         | "user" | "normal" |
| "DENIED"  | "find" | "graph"          | "*"   | "NODE(Category)"  | "user" | "normal" |
| "DENIED"  | "read" | "property(ssn)"  | "*"   | "RELATIONSHIP(*)" | "user" | "normal" |
| "GRANTED" | "read" | "all_properties" | "*"   | "RELATIONSHIP(*)" | "user" | "normal" |
| "GRANTED" | "find" | "graph"          | "*"   | "RELATIONSHIP(*)" | "user" | "normal" |
+-----------------------------------------------------------------------------------------+
```

4.3.7. Changing your own password

`ALTER CURRENT USER` is used to change the current user's own password. This command can be run by any user. The syntax is:

```
ALTER CURRENT USER SET PASSWORD FROM oldpw TO newpw
```

Arguments and options for the command are:

- `oldpw` - The current password for the user.
- `newpw` - The new password.

For example:

```
userx@system> ALTER CURRENT USER SET PASSWORD FROM 'myoldpw' TO 'mynewpw';
```

The command above is the equivalent to an admin executing the following command:

```
neo4j@system> ALTER USER userx SET PASSWORD 'mynewpw' CHANGE NOT REQUIRED;
```

NOTE

In MR2 it is still possible to change your own password when connected to any database using the procedure **`CALL dbms.security.changePassword('secretpassword')`**, but this will be removed before the final release of Neo4j 4.0. Two key differences between the 3.x and the 4.x way of setting a new password:

- In 4.0 this action must be performed on the system database.
- In 4.0 it is necessary to know and use the original password when setting the new one.

4.4. Role Management

Roles are a named collection of privileges. In Neo4j 4.0 MR2, roles are used to connect privileges to users, because:

- privileges can only be granted or denied to roles.
- roles are granted to users.

4.4.1. Built-in Roles

Neo4j 3.X provided built-in roles. Such roles are also available in Neo4j 4.0. In Neo4j 4.0 MR2, built-in roles are created the first time the database is started as part of the initialization of the system database. But from that point on they are the same as custom roles and can be modified and even dropped.

The built-in roles are:

- **`reader`** - A typical read-only role:
  ```
  GRANT MATCH (*) ON GRAPHS * ELEMENTS * TO reader
  ```
- **`editor`** - A regular user with read/write access to the database:
  ```
  GRANT MATCH (*) ON GRAPHS * ELEMENTS * TO editor
  GRANT WRITE (*) ON GRAPHS * ELEMENTS * TO editor
  ```
- **`publisher`** - A user who can also create or remove indexes and other tokens:
  ```
  GRANT MATCH (*) ON GRAPHS * ELEMENTS * TO publisher
  GRANT WRITE (*) ON GRAPHS * ELEMENTS * TO publisher
  GRANT INDEX      MANAGEMENT ON DATABASES * TO publisher // Command not available in Neo4j 4.0 MR2
  ```
- **`architect`** - A user who can manage indexes and constraints:
  ```
  GRANT MATCH (*) ON GRAPHS * ELEMENTS * TO architect
  GRANT WRITE (*) ON GRAPHS * ELEMENTS * TO architect
  GRANT INDEX      MANAGEMENT ON DATABASES * TO architect // Command not available in Neo4j 4.0 MR2
  GRANT CONSTRAINT MANAGEMENT ON DATABASES * TO architect // Command not available in Neo4j 4.0 MR2
  GRANT NAME       MANAGEMENT ON DATABASES * TO architect // Command not available in Neo4j 4.0 MR2
  ```
- **`admin`** - A superuser, database administrator:
  ```
  GRANT MATCH (*) ON GRAPHS * ELEMENTS * TO admin
  GRANT WRITE (*) ON GRAPHS * ELEMENTS * TO admin
  GRANT INDEX MANAGEMENT        ON DATABASES * TO admin // Command not available in Neo4j 4.0 MR2
  GRANT CONSTRAINT MANAGEMENT   ON DATABASES * TO admin // Command not available in Neo4j 4.0 MR2
  GRANT NAME MANAGEMENT         ON DATABASES * TO admin // Command not available in Neo4j 4.0 MR2
  GRANT ALL DATABASE PRIVILEGES ON DATABASES * TO admin // Command not available in Neo4j 4.0 MR2
  GRANT ALL DBMS PRIVILEGES     ON DBMS        TO admin // Command not available in Neo4j 4.0 MR2
  ```

You can review the privileges for the built-in roles by running the `SHOW PRIVILEGES` command. One of the examples under *4.3.9. Listing privileges* lists the output.

4.4.2. Cypher Commands for Role Management

The following Cypher commands are available for managing roles:

Command	Description
`CREATE ROLE` *`name`* `[AS COPY OF` *`name`*`]`	Create a new role
`DROP ROLE` *`name`*	Drop (remove) an existing role
`SHOW [ALL|POPULATED] ROLES [WITH USERS]`	List roles
`SHOW ROLE` *`name`* `PRIVILEGES`	List the privileges granted to a role
`GRANT ROLE` *`name`* `TO` *`user`*	Assign a role to a user
`REVOKE ROLE` *`name`* `FROM` *`user`*	Remove a role from a user

NOTE Like all the other new administration commands, role management commands must be executed in the `system` database.

4.4.3. Creating a New Role

`CREATE ROLE` is used to create a new role. The full syntax is:

```
CREATE ROLE name [AS COPY OF name]
```

Arguments and options for the command are:

- *`name`* - The name of the role to create.

- `AS COPY OF name` - The new role is created as a copy of an existing role, including all associated privileges.

NOTE In the previous Milestone Release the only way to create custom roles with write privileges was to copy a built-in role and edit it. However, in Neo4j 4.0 MR2 it is now possible to directly `GRANT` global write permissions to custom roles.

4.4.4. Removing an Existing Role

`DROP ROLE` is used to remove an existing role. The syntax is:

```
DROP ROLE name
```

4.4.5. Modifying Existing Roles

Roles are used to define a particular security behaviour. This is done by associating a collection of privileges to the role using the `GRANT`, `DENY` and `REVOKE` commands.

In Neo4j 4.0 MR2, privileges granted or denied to a role can be revoked using a command similar to the original `GRANT` or `DENY` command, but prefixed with the term `REVOKE`. However, it is important to remember that `GRANT` and `DENY` commands often map to several underlying privileges, and if you perform multiple overlapping `GRANT` or `DENY` commands when revoking one of them, you will also remove the intersection with the others. In particular, if you `REVOKE` without specifying `GRANT` or `DENY` you will remove all privileges (both granted and denied) that match the `REVOKE` specification.

For example, consider making the single role `special` and a single `GRANT MATCH`.

```
neo4j@system> CREATE ROLE detective;
0 rows available after 48 ms, consumed after another 37 ms

neo4j@system> GRANT MATCH (foo,bar) ON GRAPH neo4j ELEMENTS * TO detective;
0 rows available after 326 ms, consumed after another 2 ms
```

```
neo4j@system> SHOW ROLE detective PRIVILEGES;
+-------------------------------------------------------------------------------------+
| grant     | action | resource        | graph   | segment           | role        |
+-------------------------------------------------------------------------------------+
| "GRANTED" | "read" | "property(bar)" | "neo4j" | "NODE(*)"         | "detective" |
| "GRANTED" | "read" | "property(foo)" | "neo4j" | "NODE(*)"         | "detective" |
| "GRANTED" | "find" | "graph"         | "neo4j" | "NODE(*)"         | "detective" |
| "GRANTED" | "read" | "property(bar)" | "neo4j" | "RELATIONSHIP(*)" | "detective" |
| "GRANTED" | "find" | "graph"         | "neo4j" | "RELATIONSHIP(*)" | "detective" |
| "GRANTED" | "read" | "property(foo)" | "neo4j" | "RELATIONSHIP(*)" | "detective" |
+-------------------------------------------------------------------------------------+
```

Note how the single `GRANT` command resulted in six underlying privileges. This is because `MATCH` is syntactic sugar for a combination of `TRAVERSE` (find) and `READ` (read). In addition we specified '`ELEMENTS *`' which means both '`NODES *`' and '`RELATIONSHIPS *`'. Further, the `READ` command referred to two properties, each of which gets a separate privilege.

This also means it is possible to revoke any subset of the above set of privileges with an appropriate command:

```
neo4j@system> REVOKE TRAVERSE ON GRAPH neo4j ELEMENTS * FROM detective;
0 rows available after 160 ms, consumed after another 1 ms

neo4j@system> SHOW ROLE detective PRIVILEGES;
+-------------------------------------------------------------------------------------+
| grant     | action | resource        | graph   | segment           | role        |
+-------------------------------------------------------------------------------------+
| "GRANTED" | "read" | "property(bar)" | "neo4j" | "NODE(*)"         | "detective" |
| "GRANTED" | "read" | "property(foo)" | "neo4j" | "NODE(*)"         | "detective" |
| "GRANTED" | "read" | "property(bar)" | "neo4j" | "RELATIONSHIP(*)" | "detective" |
```

```
| "GRANTED" | "read" | "property(foo)" | "neo4j" | "RELATIONSHIP(*)" | "detective" |
+-------------------------------------------------------------------------------------+
```

In this example we revoked only the `TRAVERSE` (find) privilege, but still specified '`ELEMENTS *`' so two underlying privileges were removed (for the nodes and relationships).

4.4.6. Listing Roles

`SHOW ROLES` is used to provide a list of roles. The syntax is:

```
SHOW [ALL|POPULATED] ROLES [WITH USERS]
```

Arguments and options for the command are:

- `ALL` - Similar to `SHOW ROLES`, the option provides a list of all roles.
- `POPULATED` - List of roles that have been granted to at least one user.
- `WITH USERS` - Adds an additional column for the users that the role is granted to, containing null if the role has no users. Will provide one row for each user the role is granted to.

For example:

```
neo4j@system> SHOW ROLES;
+--------------------------+
| role         | isBuiltIn |
+--------------------------+
| "admin"      | TRUE      |
| "publisher"  | TRUE      |
| "editor"     | TRUE      |
| "reader"     | TRUE      |
| "architect"  | TRUE      |
| "my_role"    | FALSE     |
| "detective"  | FALSE     |
```

```
| "user"       | FALSE     |
| "restricted" | FALSE     |
+--------------------------+

9 rows available after 1 ms, consumed after another 1 ms
neo4j@system> SHOW POPULATED ROLES;
+--------------------------+
| role         | isBuiltIn |
+--------------------------+
| "admin"      | TRUE      |
| "user"       | FALSE     |
| "restricted" | FALSE     |
+--------------------------+

3 rows available after 18 ms, consumed after another 1 ms
neo4j@system> SHOW POPULATED ROLES WITH USERS;
+-------------------------------------+
| role         | isBuiltIn | member   |
+-------------------------------------+
| "admin"      | TRUE      | "neo4j"  |
| "user"       | FALSE     | "normal" |
| "restricted" | FALSE     | "limited"|
+-------------------------------------+

3 rows available after 22 ms, consumed after another 1 ms
neo4j@system> SHOW ALL ROLES WITH USERS;
+-------------------------------------+
| role         | isBuiltIn | member   |
+-------------------------------------+
| "admin"      | TRUE      | "neo4j"  |
| "publisher"  | TRUE      | NULL     |
| "editor"     | TRUE      | NULL     |
| "reader"     | TRUE      | NULL     |
| "architect"  | TRUE      | NULL     |
| "my_role"    | FALSE     | NULL     |
```

```
| "detective"  | FALSE     | NULL      |
| "user"       | FALSE     | "normal"  |
| "restricted" | FALSE     | "limited" |
+-------------------------------------+

9 rows available after 26 ms, consumed after another 2 ms
neo4j@system
```

The command returns the following information:

- `role` - The name of the role.
- `is_built_in` - True for the original built-in roles and false for any custom roles.
- `member` - Shown only with the `WITH USER` option and is the name of the user to whom the role has been granted. If a role is granted to multiple users, then the output will contain one row for each user the role is granted to.

4.4.7. Granting Roles to Users

The `GRANT` command is used to grant a role to a user. The syntax is:

```
GRANT ROLE name[,...] TO user[,...]
```

The command can be used to assign multiple roles to multiple users.

4.4.8. Revoking Roles from Users

The `REVOKE` command is used to revoke one or more roles from one or more users. The syntax is:

```
REVOKE ROLE name[,...] FROM user[,...]
```

4.4.9. Listing privileges

The `SHOW PRIVILEGES` command provides a list of privileges. The syntax is:

```
SHOW [ ALL
       | USER [ username ]
       | ROLE [ rolename ] ] PRIVILEGES ]
```

Arguments and options for the command are:

- `username` - The name of the user selected to list the privileges for.
- `rolename` - The name of the role selected to list the privileges for.

Specifically, there are three ways of running the command:

Command	Description
`SHOW PRIVILEGES`	Show all privileges currently defined for all roles
`SHOW ROLE` *`rolename`* `PRIVILEGES`	Show all privileges for a specific role
`SHOW USER` *`username`* `PRIVILEGES`	Show all privileges for a specific user and from which roles they got those privileges

Running the `SHOW PRIVILEGES` command before making any edits to the security model shows the privileges available for the built-in roles:

```
neo4j@system> SHOW PRIVILEGES;
+-------------------------------------------------------------------------------------+
| grant     | action  | resource         | graph | segment           | role         |
+-------------------------------------------------------------------------------------+
| "GRANTED" | "read"  | "all_properties" | "*"   | "NODE(*)"         | "admin"      |
| "GRANTED" | "find"  | "graph"          | "*"   | "NODE(*)"         | "admin"      |
| "GRANTED" | "write" | "all_properties" | "*"   | "NODE(*)"         | "admin"      |
| "GRANTED" | "write" | "token"          | "*"   | "NODE(*)"         | "admin"      |
| "GRANTED" | "write" | "schema"         | "*"   | "NODE(*)"         | "admin"      |
| "GRANTED" | "write" | "system"         | "*"   | "NODE(*)"         | "admin"      |
| "GRANTED" | "read"  | "all_properties" | "*"   | "RELATIONSHIP(*)" | "admin"      |
```

```
| "GRANTED" | "find"  | "graph"          | "*"   | "RELATIONSHIP(*)" | "admin"     |
| "GRANTED" | "write" | "all_properties" | "*"   | "RELATIONSHIP(*)" | "admin"     |
| "GRANTED" | "read"  | "all_properties" | "*"   | "NODE(*)"         | "architect" |
| "GRANTED" | "find"  | "graph"          | "*"   | "NODE(*)"         | "architect" |
| "GRANTED" | "write" | "all_properties" | "*"   | "NODE(*)"         | "architect" |
| "GRANTED" | "write" | "token"          | "*"   | "NODE(*)"         | "architect" |
| "GRANTED" | "write" | "schema"         | "*"   | "NODE(*)"         | "architect" |
| "GRANTED" | "read"  | "all_properties" | "*"   | "RELATIONSHIP(*)" | "architect" |
| "GRANTED" | "find"  | "graph"          | "*"   | "RELATIONSHIP(*)" | "architect" |
| "GRANTED" | "write" | "all_properties" | "*"   | "RELATIONSHIP(*)" | "architect" |
| "GRANTED" | "read"  | "all_properties" | "*"   | "NODE(*)"         | "editor"    |
| "GRANTED" | "find"  | "graph"          | "*"   | "NODE(*)"         | "editor"    |
| "GRANTED" | "write" | "all_properties" | "*"   | "NODE(*)"         | "editor"    |
| "GRANTED" | "read"  | "all_properties" | "*"   | "RELATIONSHIP(*)" | "editor"    |
| "GRANTED" | "find"  | "graph"          | "*"   | "RELATIONSHIP(*)" | "editor"    |
| "GRANTED" | "write" | "all_properties" | "*"   | "RELATIONSHIP(*)" | "editor"    |
| "GRANTED" | "read"  | "all_properties" | "*"   | "NODE(*)"         | "publisher" |
| "GRANTED" | "find"  | "graph"          | "*"   | "NODE(*)"         | "publisher" |
| "GRANTED" | "write" | "all_properties" | "*"   | "NODE(*)"         | "publisher" |
| "GRANTED" | "write" | "token"          | "*"   | "NODE(*)"         | "publisher" |
| "GRANTED" | "read"  | "all_properties" | "*"   | "RELATIONSHIP(*)" | "publisher" |
| "GRANTED" | "find"  | "graph"          | "*"   | "RELATIONSHIP(*)" | "publisher" |
| "GRANTED" | "write" | "all_properties" | "*"   | "RELATIONSHIP(*)" | "publisher" |
| "GRANTED" | "read"  | "all_properties" | "*"   | "NODE(*)"         | "reader"    |
| "GRANTED" | "find"  | "graph"          | "*"   | "NODE(*)"         | "reader"    |
| "GRANTED" | "read"  | "all_properties" | "*"   | "RELATIONSHIP(*)" | "reader"    |
| "GRANTED" | "find"  | "graph"          | "*"   | "RELATIONSHIP(*)" | "reader"    |
+-------------------------------------------------------------------------------------+
```

The command returns the following information:

- `grant` - The privilege access, i.e. if the privilege has been `GRANTED` or `DENIED`.
- `action` - The underlying action that is enabled by the grant. Possible values are:
  - `find` - the ability to find nodes by label or relationships by type (see the 'segment' column).
  - `read` - the ability to read the properties of graph elements.

    - `write` - coarse-grained write access to the entire graph.
- `resource` - The resource that is granted access to. Possible values are:
    - `graph` - coarse-grained access to the graph. There are two specializations of this resource applied to sub-graphs:
        - `all_properties` - when permission is granted to read all properties of a sub-graph.
        - `property` - when permission is granted to read a specific property of a sub-graph.
    - `token` - coarse-grained write access to tokens (ability to create new labels, relationship types or property names).
    - `schema` - coarse-grained write access to creating indexes and constraints.
    - `system` - coarse-grained write access to administrative functions (both multiple database management commands and user, role, and privilege management commands).
- `graph` - The graph, or list of graphs, that the privilege applies to.
- `segment` - When describing the scope to which the privilege is granted, it is possible to limit the scope to nodes of a particular label, relationships of a particular type, or * if there is no limitation.
- `role` - The role to which this privilege was granted, or the role via which the user is granted the privilege.
- `user` - When using the 'SHOW USER' version of the command, this column lists the user.

To see the privileges available to the built-in role `editor` which can perform all read and write operations on the graph:

```
neo4j@system> SHOW ROLE editor PRIVILEGES;
+-------------------------------------------------------------------------------------+
| grant     | action  | resource         | graph | segment           | role        |
+-------------------------------------------------------------------------------------+
| "GRANTED" | "read"  | "all_properties" | "*"   | "NODE(*)"         | "editor"    |
| "GRANTED" | "find"  | "graph"          | "*"   | "NODE(*)"         | "editor"    |
| "GRANTED" | "write" | "all_properties" | "*"   | "NODE(*)"         | "editor"    |
| "GRANTED" | "read"  | "all_properties" | "*"   | "RELATIONSHIP(*)" | "editor"    |
| "GRANTED" | "find"  | "graph"          | "*"   | "RELATIONSHIP(*)" | "editor"    |
| "GRANTED" | "write" | "all_properties" | "*"   | "RELATIONSHIP(*)" | "editor"    |
+-------------------------------------------------------------------------------------+
```

To see the privileges available to the built-in user `neo4j`:

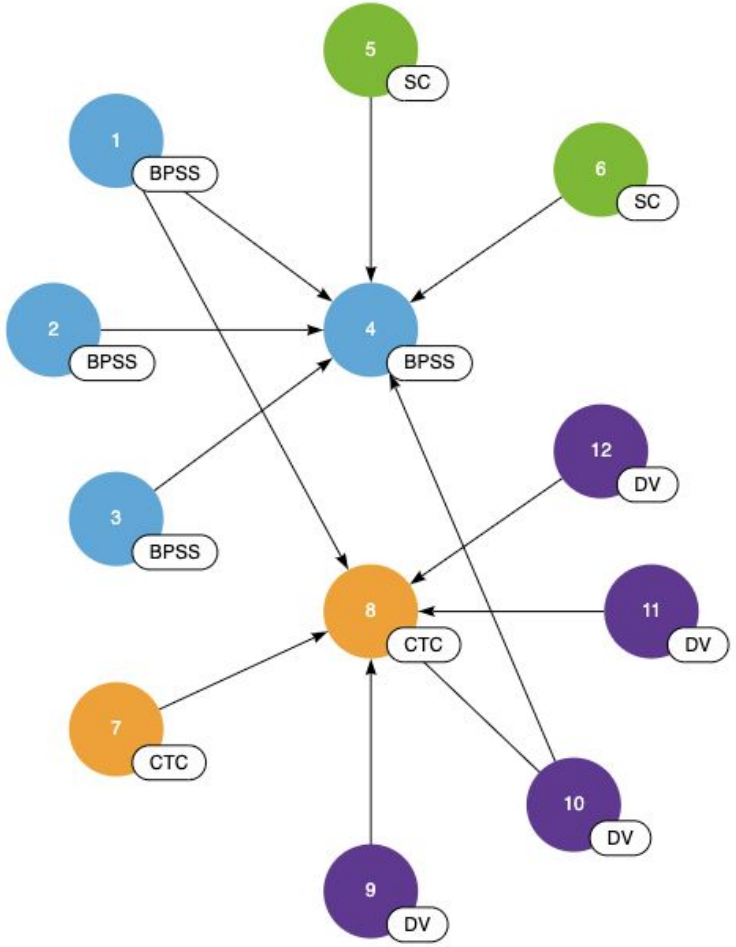
```
neo4j@system> SHOW USER neo4j PRIVILEGES;
+-------------------------------------------------------------------------------------------+
| grant     | action  | resource         | graph | segment           | role    | user    |
+-------------------------------------------------------------------------------------------+
| "GRANTED" | "read"  | "all_properties" | "*"   | "NODE(*)"         | "admin" | "neo4j" |
| "GRANTED" | "find"  | "graph"          | "*"   | "NODE(*)"         | "admin" | "neo4j" |
| "GRANTED" | "write" | "all_properties" | "*"   | "NODE(*)"         | "admin" | "neo4j" |
| "GRANTED" | "write" | "token"          | "*"   | "NODE(*)"         | "admin" | "neo4j" |
| "GRANTED" | "write" | "schema"         | "*"   | "NODE(*)"         | "admin" | "neo4j" |
| "GRANTED" | "write" | "system"         | "*"   | "NODE(*)"         | "admin" | "neo4j" |
| "GRANTED" | "read"  | "all_properties" | "*"   | "RELATIONSHIP(*)" | "admin" | "neo4j" |
| "GRANTED" | "find"  | "graph"          | "*"   | "RELATIONSHIP(*)" | "admin" | "neo4j" |
| "GRANTED" | "write" | "all_properties" | "*"   | "RELATIONSHIP(*)" | "admin" | "neo4j" |
+-------------------------------------------------------------------------------------------+
```

4.5. Security Walk-through: Label-Based Security

This example can help in understanding the capabilities of the security features in Neo4j 4.0 MR2.

4.5.1. The Graph

Suppose we have a simple graph with these nodes, associated with four labels; `BPSS`, `SC`, `CTC` and `DV`:




```
CREATE (:BPSS {p0:1, p1: 11, p2: 21});
CREATE (:BPSS {p0:2, p1: 12, p2: 22});
CREATE (:BPSS {p0:3, p1: 13, p2: 23});
CREATE (:BPSS {p0:4, p1: 14, p2: 24});

MATCH (n1:BPSS {p0:1}), (n4:BPSS {p0:4}) MERGE (n1)-[:R]->(n4);
MATCH (n2:BPSS {p0:2}), (n4:BPSS {p0:4}) MERGE (n2)-[:R]->(n4);
MATCH (n3:BPSS {p0:3}), (n4:BPSS {p0:4}) MERGE (n3)-[:R]->(n4);

CREATE (:SC {p0:5, p1: 15, p2: 25});
CREATE (:SC {p0:6, p1: 16, p2: 26});

MATCH (n5:SC {p0:5}), (n4:BPSS {p0:4}) MERGE (n5)-[:R]->(n4);
MATCH (n6:SC {p0:6}), (n4:BPSS {p0:4}) MERGE (n6)-[:R]->(n4);

CREATE (:CTC {p0:7, p1: 17, p2: 27})-[:R]->(:CTC {p0:8, p1: 18, p2:28});

MATCH (n1:BPSS {p0:1}), (n8:CTC {p0:8}) MERGE (n1)-[:R]->(n8);

CREATE (:DV {p0:9,  p1:19, p2:29});
CREATE (:DV {p0:10, p1:20, p2:30});
CREATE (:DV {p0:11, p1:21, p2:31});
CREATE (:DV {p0:12, p1:22, p2:32});

MATCH (n9:DV {p0:9}), (n8:CTC {p0:8}) MERGE (n9)-[:R]->(n8);
MATCH (n10:DV {p0:10}), (n8:CTC {p0:8}) MERGE (n10)-[:R]->(n8);
MATCH (n11:DV {p0:11}), (n8:CTC {p0:8}) MERGE (n11)-[:R]->(n8);
MATCH (n12:DV {p0:12}), (n8:CTC {p0:8}) MERGE (n12)-[:R]->(n8);
MATCH (n10:DV {p0:10}), (n4:BPSS {p0:4}) MERGE (n10)-[:R]->(n4);
```

4.5.2. Users and Roles

In our example, we want to create four security levels, and each level is associated with a role.

This is the list of levels, in order of importance:

- Baseline Personnel Security Standard - associated with node label `BPSS`.
- Security Check - associated with node label `SC`.
- Counter-Terrorism Check - associated with node label `CTC`.
- Developed Vetting - associated with node label `DV`.

These are the commands we can use to create the roles and to `GRANT` access to the nodes with the given labels:

```
CREATE ROLE Baseline_Personnel_Security_Standard;
CREATE ROLE Security_Check;
CREATE ROLE Counter_Terrorism_Check;
CREATE ROLE Developed_Vetting;

GRANT TRAVERSE  ON GRAPH * ELEMENTS * TO Baseline_Personnel_Security_Standard;
GRANT TRAVERSE  ON GRAPH * ELEMENTS * TO Security_Check;
GRANT TRAVERSE  ON GRAPH * ELEMENTS * TO Counter_Terrorism_Check;
GRANT TRAVERSE  ON GRAPH * ELEMENTS * TO Developed_Vetting;
GRANT READ (*)  ON GRAPH * NODES BPSS TO Baseline_Personnel_Security_Standard;
GRANT READ (*)  ON GRAPH * NODES SC   TO Security_Check;
GRANT READ (*)  ON GRAPH * NODES CTC  TO Counter_Terrorism_Check;
GRANT READ (*)  ON GRAPH * NODES DV   TO Developed_Vetting;
```

Note that the `TRAVERSE` and `READ` privileges are referring to different elements. In particular we grant `TRAVERSE` to all `ELEMENTS` (both `NODES` and `RELATIONSHIPS`) so that the entire graph can be traversed. However, we only `GRANT READ` to the `NODES` since we do not need the users to see properties of the `RELATIONSHIPS`.

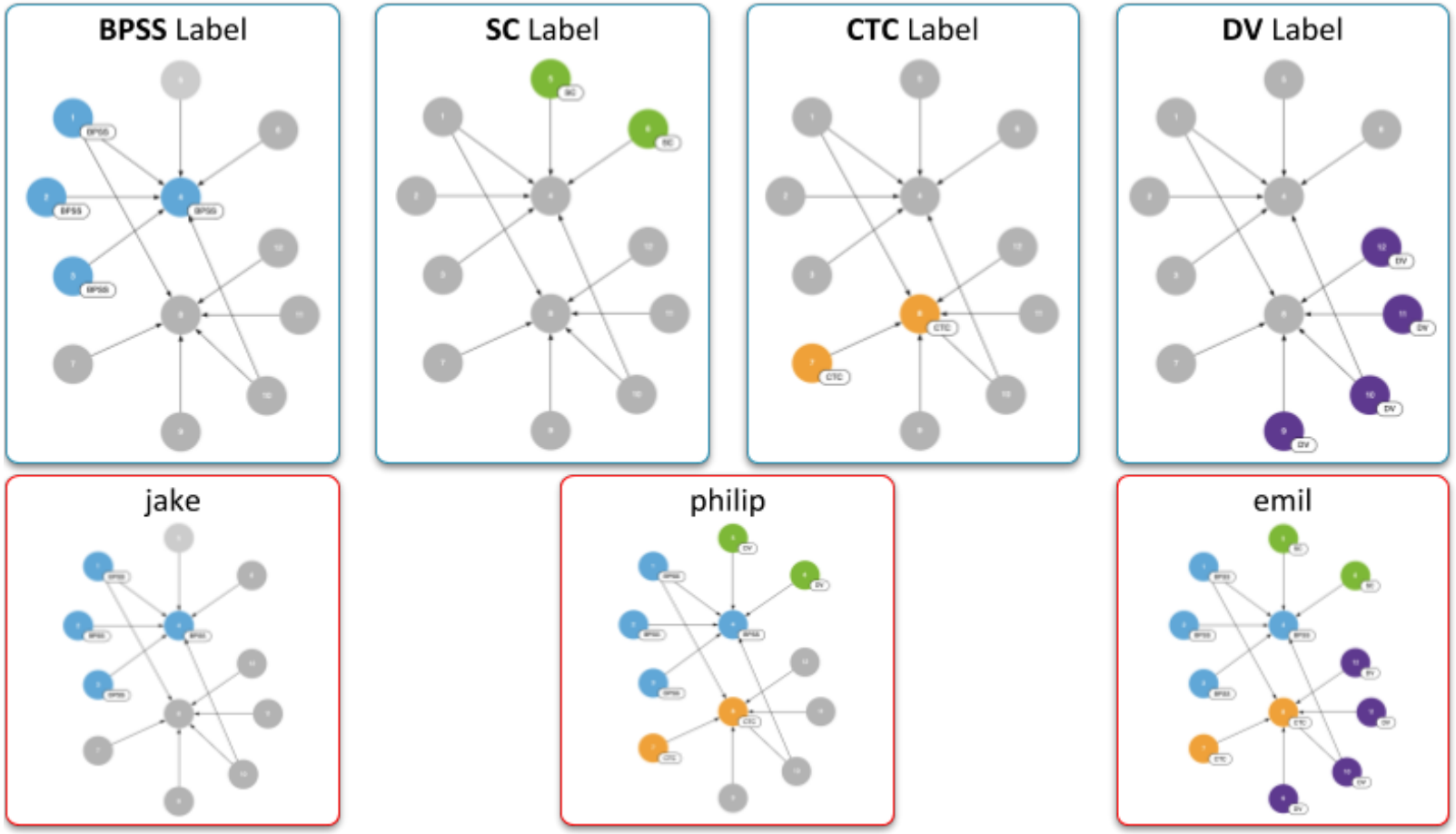
We also create users with different levels of security clearance:

- User `jake` has security clearance associated to the role `Baseline_Personnel_Security_Standard`.
- User `philip` has security clearance associated to the roles `Baseline_Personnel_Security_Standard`, `Security_Check` and `Counter_Terrorism_Check`.
- User `emil` has maximum security clearance associated to all the roles, i.e. `Baseline_Personnel_Security_Standard`, `Security_Check`, `Counter_Terrorism_Check` and `Developed_Vetting`.

These are the commands we can use to create the users and grant the roles:

```
CREATE USER jake   SET PASSWORD 'jake'   CHANGE NOT REQUIRED SET STATUS ACTIVE;
CREATE USER philip SET PASSWORD 'philip' CHANGE NOT REQUIRED SET STATUS ACTIVE;
CREATE USER emil   SET PASSWORD 'emil'   CHANGE NOT REQUIRED SET STATUS ACTIVE;

GRANT ROLE Baseline_Personnel_Security_Standard TO jake, philip, emil;
GRANT ROLE Security_Check TO philip, emil;
GRANT ROLE Counter_Terrorism_Check TO philip, emil;
GRANT ROLE Developed_Vetting TO emil;
```

Putting all together, the three users will have access to sub-graphs, that can be visualized in this picture:



4.5.3. Data Return from `MATCH` queries

We can now experiment a simple query:

```
MATCH (i)-[j]->(k) RETURN i,j,k ORDER BY i.p0, j.p0, j.p1, k.p0;
```

This query will return different results depending on the user (and their related security clearance and role) who execute it.

User `emil`, who has full access to the graph, will see these results:

```
emil@neo4j> MATCH (i)-[j]->(k) RETURN i,j,k ORDER BY i.p0, j.p0, j.p1, k.p0;
+-------------------------------------------------------------------------------+
| i                                  | j     | k                                |
+-------------------------------------------------------------------------------+
| (:BPSS {p0: 1, p1: 11, p2: 21})  | [:R]  | (:BPSS {p0: 4, p1: 14, p2: 24}) |
| (:BPSS {p0: 1, p1: 11, p2: 21})  | [:R]  | (:CTC {p0: 8, p1: 18, p2: 28})  |
| (:BPSS {p0: 2, p1: 12, p2: 22})  | [:R]  | (:BPSS {p0: 4, p1: 14, p2: 24}) |
| (:BPSS {p0: 3, p1: 13, p2: 23})  | [:R]  | (:BPSS {p0: 4, p1: 14, p2: 24}) |
| (:SC {p0: 5, p1: 15, p2: 25})    | [:R]  | (:BPSS {p0: 4, p1: 14, p2: 24}) |
| (:SC {p0: 6, p1: 16, p2: 26})    | [:R]  | (:BPSS {p0: 4, p1: 14, p2: 24}) |
| (:CTC {p0: 7, p1: 17, p2: 27})   | [:R]  | (:CTC {p0: 8, p1: 18, p2: 28})  |
| (:DV {p0: 9, p1: 19, p2: 29})    | [:R]  | (:CTC {p0: 8, p1: 18, p2: 28})  |
| (:DV {p0: 10, p1: 20, p2: 30})   | [:R]  | (:BPSS {p0: 4, p1: 14, p2: 24}) |
| (:DV {p0: 10, p1: 20, p2: 30})   | [:R]  | (:CTC {p0: 8, p1: 18, p2: 28})  |
| (:DV {p0: 11, p1: 21, p2: 31})   | [:R]  | (:CTC {p0: 8, p1: 18, p2: 28})  |
| (:DV {p0: 12, p1: 22, p2: 32})   | [:R]  | (:CTC {p0: 8, p1: 18, p2: 28})  |
+-------------------------------------------------------------------------------+
```

User `philip` does not have access to nodes with label `DV`, therefore the result will be:

```
philip@neo4j> MATCH (i)-[j]->(k) RETURN i,j,k ORDER BY i.p0, j.p0, j.p1, k.p0;
+-------------------------------------------------------------------------------+
| i                                  | j     | k                                |
+-------------------------------------------------------------------------------+
| (:BPSS {p0: 1, p1: 11, p2: 21})  | [:R]  | (:BPSS {p0: 4, p1: 14, p2: 24}) |
| (:BPSS {p0: 1, p1: 11, p2: 21})  | [:R]  | (:CTC {p0: 8, p1: 18, p2: 28})  |
| (:BPSS {p0: 2, p1: 12, p2: 22})  | [:R]  | (:BPSS {p0: 4, p1: 14, p2: 24}) |
| (:BPSS {p0: 3, p1: 13, p2: 23})  | [:R]  | (:BPSS {p0: 4, p1: 14, p2: 24}) |
| (:SC {p0: 5, p1: 15, p2: 25})    | [:R]  | (:BPSS {p0: 4, p1: 14, p2: 24}) |
| (:SC {p0: 6, p1: 16, p2: 26})    | [:R]  | (:BPSS {p0: 4, p1: 14, p2: 24}) |
| (:CTC {p0: 7, p1: 17, p2: 27})   | [:R]  | (:CTC {p0: 8, p1: 18, p2: 28})  |
| (:DV)                            | [:R]  | (:BPSS {p0: 4, p1: 14, p2: 24}) |
```

```
| (:DV)                          | [:R] | (:CTC {p0: 8, p1: 18, p2: 28})  |
| (:DV)                          | [:R] | (:CTC {p0: 8, p1: 18, p2: 28})  |
| (:DV)                          | [:R] | (:CTC {p0: 8, p1: 18, p2: 28})  |
| (:DV)                          | [:R] | (:CTC {p0: 8, p1: 18, p2: 28})  |
+--------------------------------------------------------------------------+
```

Finally, user `jake` has a more restrictive access (only to nodes with label `BPSS`), therefore the result will be:

```
jake@neo4j> MATCH (i)-[j]->(k) RETURN i,j,k ORDER BY i.p0, j.p0, j.p1, k.p0;
+--------------------------------------------------------------------------+
| i                              | j    | k                               |
+--------------------------------------------------------------------------+
| (:BPSS {p0: 1, p1: 11, p2: 21}) | [:R] | (:BPSS {p0: 4, p1: 14, p2: 24}) |
| (:BPSS {p0: 1, p1: 11, p2: 21}) | [:R] | (:CTC)                          |
| (:BPSS {p0: 2, p1: 12, p2: 22}) | [:R] | (:BPSS {p0: 4, p1: 14, p2: 24}) |
| (:BPSS {p0: 3, p1: 13, p2: 23}) | [:R] | (:BPSS {p0: 4, p1: 14, p2: 24}) |
| (:SC)                          | [:R] | (:BPSS {p0: 4, p1: 14, p2: 24}) |
| (:SC)                          | [:R] | (:BPSS {p0: 4, p1: 14, p2: 24}) |
| (:DV)                          | [:R] | (:BPSS {p0: 4, p1: 14, p2: 24}) |
| (:CTC)                         | [:R] | (:CTC)                          |
| (:DV)                          | [:R] | (:CTC)                          |
| (:DV)                          | [:R] | (:CTC)                          |
| (:DV)                          | [:R] | (:CTC)                          |
| (:DV)                          | [:R] | (:CTC)                          |
+--------------------------------------------------------------------------+
```

4.5.4. Removing the `TRAVERSE` Privilege

Continuing with the example, we now suppose that we intend to remove the `TRAVERSE` privilege on all the nodes, i.e. `TRAVERSE` is only valid for the same nodes that are accessible with the `MATCH` privilege. We can `REVOKE` all the individual privileges, but it is easier to `DROP` the roles and recreate the privileges, so we'll do that here.

```
DROP ROLE Baseline_Personnel_Security_Standard;
```

```
DROP ROLE Security_Check;
DROP ROLE Counter_Terrorism_Check;
DROP ROLE Developed_Vetting;
```

The commands to set the privileges are:

```
CREATE ROLE Baseline_Personnel_Security_Standard;
CREATE ROLE Security_Check;
CREATE ROLE Counter_Terrorism_Check;
CREATE ROLE Developed_Vetting;

// Since the GRANT MATCH will only cover NODES, we need to separately grant RELATIONSHIPS
GRANT TRAVERSE ON GRAPH * RELATIONSHIPS * TO Baseline_Personnel_Security_Standard;
GRANT TRAVERSE ON GRAPH * RELATIONSHIPS * TO Security_Check;
GRANT TRAVERSE ON GRAPH * RELATIONSHIPS * TO Counter_Terrorism_Check;
GRANT TRAVERSE ON GRAPH * RELATIONSHIPS * TO Developed_Vetting;

// The GRANT MATCH … NODES will provide TRAVERSE for those NODES as well
GRANT MATCH (*) ON GRAPH * NODES BPSS TO Baseline_Personnel_Security_Standard;
GRANT MATCH (*) ON GRAPH * NODES SC TO Security_Check;
GRANT MATCH (*) ON GRAPH * NODES CTC TO Counter_Terrorism_Check;
GRANT MATCH (*) ON GRAPH * NODES DV TO Developed_Vetting;

// Now assign these roles to the users
GRANT ROLE Baseline_Personnel_Security_Standard TO jake, philip, emil;
GRANT ROLE Security_Check TO philip, emil;
GRANT ROLE Counter_Terrorism_Check TO philip, emil;
GRANT ROLE Developed_Vetting TO emil;
```

The same `MATCH` query executed with the new security model provides different results:

```
emil@neo4j> MATCH (i)-[j]->(k) RETURN i,j,k ORDER BY i.p0, j.p0, j.p1, k.p0;
+-------------------------------------------------------------------------------+
| i                                | j    | k                                |
+-------------------------------------------------------------------------------+
| (:BPSS {p0: 1, p1: 11, p2: 21}) | [:R] | (:BPSS {p0: 4, p1: 14, p2: 24}) |
| (:BPSS {p0: 1, p1: 11, p2: 21}) | [:R] | (:CTC {p0: 8, p1: 18, p2: 28})  |
| (:BPSS {p0: 2, p1: 12, p2: 22}) | [:R] | (:BPSS {p0: 4, p1: 14, p2: 24}) |
| (:BPSS {p0: 3, p1: 13, p2: 23}) | [:R] | (:BPSS {p0: 4, p1: 14, p2: 24}) |
| (:SC {p0: 5, p1: 15, p2: 25})   | [:R] | (:BPSS {p0: 4, p1: 14, p2: 24}) |
| (:SC {p0: 6, p1: 16, p2: 26})   | [:R] | (:BPSS {p0: 4, p1: 14, p2: 24}) |
| (:CTC {p0: 7, p1: 17, p2: 27})  | [:R] | (:CTC {p0: 8, p1: 18, p2: 28})  |
| (:DV {p0: 9, p1: 19, p2: 29})   | [:R] | (:CTC {p0: 8, p1: 18, p2: 28})  |
| (:DV {p0: 10, p1: 20, p2: 30})  | [:R] | (:BPSS {p0: 4, p1: 14, p2: 24}) |
| (:DV {p0: 10, p1: 20, p2: 30})  | [:R] | (:CTC {p0: 8, p1: 18, p2: 28})  |
| (:DV {p0: 11, p1: 21, p2: 31})  | [:R] | (:CTC {p0: 8, p1: 18, p2: 28})  |
| (:DV {p0: 12, p1: 22, p2: 32})  | [:R] | (:CTC {p0: 8, p1: 18, p2: 28})  |
+-------------------------------------------------------------------------------+
```

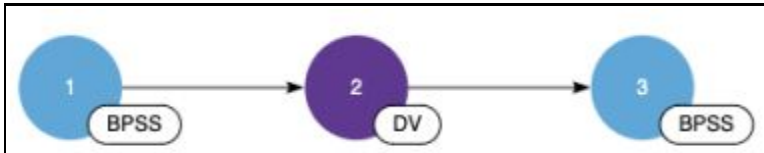
```
philip@neo4j> MATCH (i)-[j]->(k) RETURN i,j,k ORDER BY i.p0, j.p0, j.p1, k.p0;
+-------------------------------------------------------------------------------+
| i                                | j    | k                                |
+-------------------------------------------------------------------------------+
| (:BPSS {p0: 1, p1: 11, p2: 21}) | [:R] | (:BPSS {p0: 4, p1: 14, p2: 24}) |
| (:BPSS {p0: 1, p1: 11, p2: 21}) | [:R] | (:CTC {p0: 8, p1: 18, p2: 28})  |
| (:BPSS {p0: 2, p1: 12, p2: 22}) | [:R] | (:BPSS {p0: 4, p1: 14, p2: 24}) |
| (:BPSS {p0: 3, p1: 13, p2: 23}) | [:R] | (:BPSS {p0: 4, p1: 14, p2: 24}) |
| (:SC {p0: 5, p1: 15, p2: 25})   | [:R] | (:BPSS {p0: 4, p1: 14, p2: 24}) |
| (:SC {p0: 6, p1: 16, p2: 26})   | [:R] | (:BPSS {p0: 4, p1: 14, p2: 24}) |
| (:CTC {p0: 7, p1: 17, p2: 27})  | [:R] | (:CTC {p0: 8, p1: 18, p2: 28})  |
+-------------------------------------------------------------------------------+
```

```
jake@neo4j> MATCH (i)-[j]->(k) RETURN i,j,k ORDER BY i.p0, j.p0, j.p1, k.p0;
+-------------------------------------------------------------------------------+
| i                                | j    | k                                |
+-------------------------------------------------------------------------------+
| (:BPSS {p0: 1, p1: 11, p2: 21}) | [:R] | (:BPSS {p0: 4, p1: 14, p2: 24}) |
| (:BPSS {p0: 2, p1: 12, p2: 22}) | [:R] | (:BPSS {p0: 4, p1: 14, p2: 24}) |
| (:BPSS {p0: 3, p1: 13, p2: 23}) | [:R] | (:BPSS {p0: 4, p1: 14, p2: 24}) |
+-------------------------------------------------------------------------------+
```

In the previous example, all users retrieved the same number rows, while in this new example, users `jake` and `philip` retrieve fewer rows.

4.5.5. Another Example with `TRAVERSE`

In this example, we illustrate the impact of the TRAVERSE privilege in queries with a simpler data model. Users, roles and privileges are intact from the previous example, but the data is recreated as follows:




```
MATCH (n) DETACH DELETE n;
CREATE (:BPSS {p0:1, p1: 11, p2: 21})-[:R]->(:DV {p0: 2, p1: 12, p2:
22})-[:R]->(:BPSS {p0: 3, p1: 13, p2: 23});
```

Without `TRAVERSE` access to all nodes, user `jake` would get no rows in his results:

```
GRANT MATCH (*) ON GRAPH * NODES BPSS TO Baseline_Personnel_Security_Standard;

jake@neo4j> MATCH (a)-[b]->(c)-[d]->(e) RETURN a,b,c,d,e;
+-------------------+
| a | b | c | d | e |
+-------------------+
+-------------------+
```

A query that does not include the traversal of a node with higher security clearance would return values:

```
jake@neo4j> MATCH (n) RETURN n ORDER BY n.p0;
+---------------------------------+
| n                               |
+---------------------------------+
| (:BPSS {p0: 1, p1: 11, p2: 21}) |
| (:BPSS {p0: 3, p1: 13, p2: 23}) |
+---------------------------------+
```

With the `TRAVERSE` access to all nodes, user `jake` would see these results:

```
GRANT TRAVERSE  ON GRAPH * NODES * TO Baseline_Personnel_Security_Standard;
GRANT MATCH (*) ON GRAPH * NODES BPSS TO Baseline_Personnel_Security_Standard;

jake@neo4j> MATCH (a)-[b]->(c)-[d]->(e) RETURN a,b,c,d,e;
+-----------------------------------------------------------------------------------------+
| a                                | b    | c     | d    | e                               |
+-----------------------------------------------------------------------------------------+
| (:BPSS {p0: 1, p1: 11, p2: 21}) | [:R] | (:DV) | [:R] | (:BPSS {p0: 3, p1: 13, p2: 23}) |
+-----------------------------------------------------------------------------------------+
```

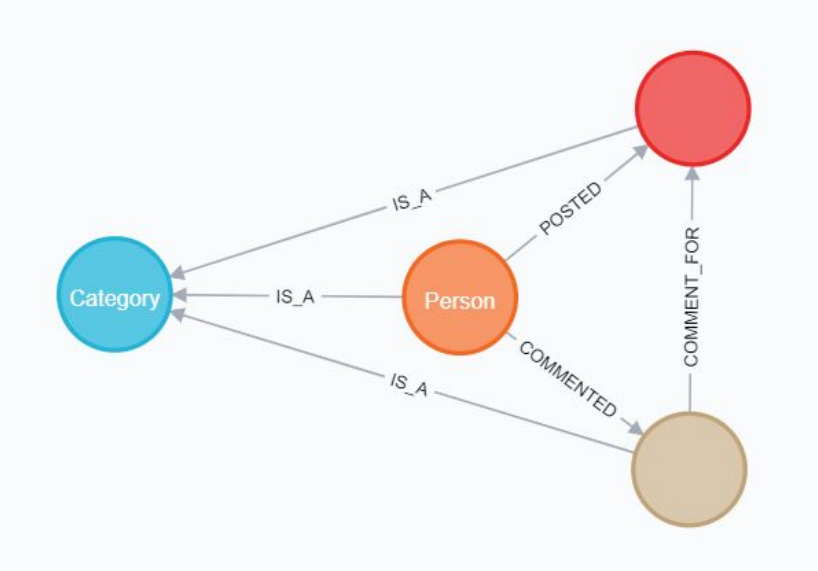
CAUTION! Denying a `TRAVERSE` access may be necessary for some use cases, but it may significantly restrict the number of rows returned by `MATCH` queries, therefore it should be used with caution.

4.6. Security Walk-through: Combining `GRANT` and `DENY`

This example can help in understanding the new `DENY` capability introduced to the security features in Neo4j 4.0 MR2.

4.6.1. The Graph

Suppose we have a simple graph of a social network, with people that make posts, and comments on posts, and all three types can be categorized for auditing purposes. We would like normal users to be able to see the people, posts and comments, but not the categories. And restricted users to also be unable to see the comments.




```
// Delete current model
MATCH (n) DETACH DELETE n;

// Create Person nodes
UNWIND range( 0, 10 ) AS index
WITH index, toInteger( rand() * 10 ) AS cat, toInteger( rand() * 1000000 ) + 987654 AS ssn
CREATE( user:Person {name:'user_'+index, ssn:ssn} )
MERGE ( category:Category {name:'People category '+cat} )
MERGE ( user )-[:IS_A]->( category )
RETURN user.name, user.ssn, category.name;

// Create Post nodes
MATCH ( user:Person )
WITH user, ['A','B','C'] AS titles
UNWIND range( 0, size( titles ) - 1 ) AS pi
WITH user, titles, pi, titles[pi] AS title, toInteger( rand() * 10 ) AS rnd, toInteger( rand() * 10 ) AS
cat
CREATE ( post:Post {title:'My post ' + title, content: 'Some content about ' + title} )
```

```
CREATE ( user )-[posted:POSTED {created_at:date({year:2019, month:7, day:1 + rnd + pi} )}]->( post )
MERGE ( category:Category {name:'Post category ' + cat} )
MERGE ( post )-[:IS_A]->( category )
RETURN user.name, posted.created_at, post.title;

// Create comments to posts
MATCH ( user:Person ),
      ( post:Post )<-[:POSTED]-( poster:Person )
WHERE NOT ( ( user )-[:POSTED]->( post ) )
WITH poster, user, post, toInteger( rand() * 10 ) AS rnd, toInteger( rand() * 10 ) AS cat
CREATE ( user )-[c:COMMENTED {created_at:date( {year:2019, month:7, day: rnd + post.created_at.day } )
}]->
       ( comment:Comment {text: 'I think that "' + poster.name + '" made some very valid points in "' +
post.title + '"'} )-[:COMMENT_FOR]->( post )
MERGE ( category:Category {name:'Comment category ' + cat } )
MERGE ( comment )-[:IS_A]->( category )
RETURN user.name, c.created_at, comment.text, post.title;
```

4.6.2. Users and Roles

In this example, the neo4j user will be the administrator and able to see all, while two new roles will be created for normal users and restricted users:

- Normal users can see all nodes except the 'Category' nodes, and are also restricted from seeing the users social security property.
- Restricted users can only see the posts that users create, but not the comments.

Both of these roles can be created using combinations of `GRANT` and `DENY`. We will use both `GRANT` and `DENY` to setup the normal users, but will use only whitelisting (`GRANT`) for the restricted user:

```
// Remove users and roles
DROP USER normal;
DROP USER limited;
DROP ROLE user;
DROP ROLE restricted;
```

```
// show current security rules
SHOW PRIVILEGES;

// Create and configure the normal user - see everything except Category and ssn
:param secret => 'secret'
CREATE USER normal SET PASSWORD $secret CHANGE NOT REQUIRED;
CREATE ROLE user;
GRANT ROLE user TO normal;
GRANT TRAVERSE ON GRAPH * ELEMENTS * TO user;
GRANT READ (*) ON GRAPH * ELEMENTS * TO user;
DENY TRAVERSE ON GRAPH * NODES Category TO user;
DENY READ (ssn) ON GRAPH * ELEMENTS * TO user;
SHOW USER normal PRIVILEGES;

// Create and configure a more restricted user - see only Person(name, email) and Post(*)
CREATE USER limited SET PASSWORD $secret CHANGE NOT REQUIRED;
CREATE ROLE restricted;
GRANT ROLE restricted TO limited;
GRANT TRAVERSE ON GRAPH * RELATIONSHIPS * TO restricted;
GRANT TRAVERSE ON GRAPH * NODES Person, Post TO restricted;
GRANT READ (*) ON GRAPH * NODES Post (*) TO restricted;
GRANT READ (name, email) ON GRAPH * NODES Person (*) TO restricted;
SHOW USER limited PRIVILEGES;
```

4.6.3. Example queries

Let's investigate the privileges of these two new roles. We can see that the normal user is controlled with a combination of `GRANTED` and `DENIED` privileges:

```

```

```
neo4j@system> SHOW USER normal PRIVILEGES;
+-----------------------------------------------------------------------------------------+
| grant     | action | resource         | graph | segment           | role   | user     |
+-----------------------------------------------------------------------------------------+
| "DENIED"  | "read" | "property(ssn)"  | "*"   | "NODE(*)"         | "user" | "normal" |
| "GRANTED" | "read" | "all_properties" | "*"   | "NODE(*)"         | "user" | "normal" |
| "GRANTED" | "find" | "graph"          | "*"   | "NODE(*)"         | "user" | "normal" |
| "DENIED"  | "find" | "graph"          | "*"   | "NODE(Category)"  | "user" | "normal" |
| "DENIED"  | "read" | "property(ssn)"  | "*"   | "RELATIONSHIP(*)" | "user" | "normal" |
| "GRANTED" | "read" | "all_properties" | "*"   | "RELATIONSHIP(*)" | "user" | "normal" |
| "GRANTED" | "find" | "graph"          | "*"   | "RELATIONSHIP(*)" | "user" | "normal" |
+-----------------------------------------------------------------------------------------+
```

While the limited user is controlled through whitelisting alone:

```
neo4j@system> SHOW USER limited PRIVILEGES;
+-------------------------------------------------------------------------------------------------+
| grant     | action | resource          | graph | segment           | role         | user      |
+-------------------------------------------------------------------------------------------------+
| "GRANTED" | "read" | "property(email)" | "*"   | "NODE(Person)"    | "restricted" | "limited" |
| "GRANTED" | "read" | "property(name)"  | "*"   | "NODE(Person)"    | "restricted" | "limited" |
| "GRANTED" | "find" | "graph"           | "*"   | "NODE(Person)"    | "restricted" | "limited" |
| "GRANTED" | "read" | "all_properties"  | "*"   | "NODE(Post)"      | "restricted" | "limited" |
| "GRANTED" | "find" | "graph"           | "*"   | "NODE(Post)"      | "restricted" | "limited" |
| "GRANTED" | "find" | "graph"           | "*"   | "RELATIONSHIP(*)" | "restricted" | "limited" |
+-------------------------------------------------------------------------------------------------+
```

Let’s first investigate the differences in terms of whether the users can see the social security numbers of users:

```

```

```
neo4j@neo4j> MATCH (user:Person)
             RETURN count(user), substring(toString(user.ssn),0,2);
+-----------------------------------------------------+
| count(user) | substring(toString(user.ssn),0,2) |
+-----------------------------------------------------+
| 1           | "15"                                |
| 2           | "19"                                |
| 4           | "11"                                |
| 1           | "16"                                |
| 1           | "14"                                |
| 1           | "17"                                |
| 1           | "18"                                |
+-----------------------------------------------------+
```

```
normal@neo4j> MATCH (user:Person)
              RETURN count(user), substring(toString(user.ssn),0,2);
+-----------------------------------------------------+
| count(user) | substring(toString(user.ssn),0,2) |
+-----------------------------------------------------+
| 11          | null                                |
+-----------------------------------------------------+
```

```
limited@neo4j> MATCH (user:Person)
               RETURN count(user), substring(toString(user.ssn),0,2);
+-----------------------------------------------------+
| count(user) | substring(toString(user.ssn),0,2) |
+-----------------------------------------------------+
| 11          | null                                |
+-----------------------------------------------------+
```

Now let's see what happens when we try to look at Categories:

```
neo4j@neo4j> MATCH (user:Person)-[:IS_A]->(category:Category)
             RETURN category.name, count(user);
+-----------------------------------+
| category.name       | count(user) |
+-----------------------------------+
| "People category 1" | 3           |
| "People category 7" | 2           |
| "People category 3" | 2           |
| "People category 9" | 1           |
| "People category 2" | 1           |
| "People category 5" | 1           |
| "People category 8" | 1           |
+-----------------------------------+
```

```
normal@neo4j> MATCH (user:Person)-[:IS_A]->(category:Category)
              RETURN category.name, count(user);
(no changes, no records)
```

```
limited@neo4j> MATCH (user:Person)-[:IS_A]->(category:Category)
               RETURN category.name, count(user);
(no changes, no records)
```

We know that `normal` and `limited` users differ in terms of the visibility of comments, so let's take a look at those:

```
neo4j@neo4j> MATCH (user:Person)
             RETURN user.name AS poster, size((user)-[:POSTED]->(:Post)<-[:COMMENT_FOR]-(:Comment));
+-------------------------------------------------------------------------+
| poster    | size((user)-[:POSTED]->(:Post)<-[:COMMENT_FOR]-(:Comment)) |
+-------------------------------------------------------------------------+
| "user_0"  | 30                                                         |
| "user_1"  | 30                                                         |
| "user_2"  | 30                                                         |
| "user_3"  | 30                                                         |
| "user_4"  | 30                                                         |
| "user_5"  | 30                                                         |
| "user_6"  | 30                                                         |
| "user_7"  | 30                                                         |
| "user_8"  | 30                                                         |
| "user_9"  | 30                                                         |
| "user_10" | 30                                                         |
+-------------------------------------------------------------------------+
```

```
normal@neo4j> MATCH (user:Person)
              RETURN user.name AS poster, size((user)-[:POSTED]->(:Post)<-[:COMMENT_FOR]-(:Comment));
+-------------------------------------------------------------------------+
| poster    | size((user)-[:POSTED]->(:Post)<-[:COMMENT_FOR]-(:Comment)) |
+-------------------------------------------------------------------------+
| "user_0"  | 30                                                         |
| "user_1"  | 30                                                         |
| "user_2"  | 30                                                         |
| "user_3"  | 30                                                         |
| "user_4"  | 30                                                         |
```

```
| "user_5"  | 30                                                                  |
| "user_6"  | 30                                                                  |
| "user_7"  | 30                                                                  |
| "user_8"  | 30                                                                  |
| "user_9"  | 30                                                                  |
| "user_10" | 30                                                                  |
+---------------------------------------------------------------------------+
```

```
limited@neo4j> MATCH (user:Person)
               RETURN user.name AS poster, size((user)-[:POSTED]->(:Post)<-[:COMMENT_FOR]-(:Comment));
+---------------------------------------------------------------------------+
| poster    | size((user)-[:POSTED]->(:Post)<-[:COMMENT_FOR]-(:Comment)) |
+---------------------------------------------------------------------------+
| "user_0"  | 0                                                                 |
| "user_1"  | 0                                                                 |
| "user_2"  | 0                                                                 |
| "user_3"  | 0                                                                 |
| "user_4"  | 0                                                                 |
| "user_5"  | 0                                                                 |
| "user_6"  | 0                                                                 |
| "user_7"  | 0                                                                 |
| "user_8"  | 0                                                                 |
| "user_9"  | 0                                                                 |
| "user_10" | 0                                                                 |
+---------------------------------------------------------------------------+
```

It is possible to create even more complex scenarios with combinations of `GRANT` and `DENY`.

5. Drivers and Client/Server Connectivity

NOTE

New drivers are under construction and will be needed to explore the new Bolt server features, but the latest alpha 2.0 drivers enable a preview.

It is also important to note that the 4.0 release will see the Drivers rename the 2.0 series to 4.0 to align the versioning between the client and the server before server GA.

5.1. Bolt Server

A new bolt protocol is introduced in this Milestone Release, which enables database selection for running queries on multiple databases, as well as back pressure with Reactive clients.

To fully use the new features provided by bolt server, 2.0 series drivers are required. Currently the drivers are still under heavy construction and the API is not fully stable yet. However we can still peek at the future with the latest alpha 2.0 drivers:

- The latest Java alpha driver has the full support of multi-database and back pressure.
- The latest JavaScript alpha driver also provides database selection for managing multiple databases.
- The full support of 4.0 Bolt server features will arrive shortly in all language drivers.

5.2. Database Selection

As managing multiple databases is introduced in this release, before starting a transaction, we need to first select the database where the transaction would run against.

The following examples show how to run a query against database called `foo`.

5.2.1. Java driver

```
try( Session session = neo4j.driver().session( t -> t.withDatabase( "foo" ) ) )
{
   StatementResult result = session.run( "RETURN 42" );
   assertThat( result.single().get( 0 ).asInt(), equalTo( 1 ) );
   assertThat( result.summary.database().name(), equalTo(“foo”) );
}
```

5.2.2. Javascript driver

```
const neoSession = driver.session({ database: 'foo' })

try {
  const result = await session.run('RETURN 42')

  expect(result.records[0].get(0).toInt()).toBe(1)
  expect(result.summary.database.name).toBe('foo')
} finally {
  neoSession.close()
}
```

The database name is passed at session creation. All the transactions created in the same session will be executed against the same database specified on the session creation. If no database is specified, then the default database set in server configuration will be chosen to execute the query.

5.3. Back Pressure

Neo4j provides a full stack back pressure, from client to Bolt server, all the way down to Cypher Execution Engine.

To make use of back pressure, a reactive client is required. For the Neo4j 4.0 Milestone 1 release, only the Java driver and SDN-RX will fully deliver this feature.

The reactive API in the Java driver is located in package `org.neo4j.driver.reactive`. We've exposed the Publisher-Subscriber API specified by `Reactive Streams`. You can adopt our driver into a reactive application by using the `Reactive Streams` libraries, such as `observable streams` or `project reactor`.

The following are two code examples to run queries with `observable streams` and `project reactor`:

```
public Flowable<String> readProductTitlesRxJava()
{
   String query = "MATCH (p:Product) WHERE p.id = $id RETURN p.title";
   Map<String,Object> parameters = Collections.singletonMap( "id", 0 );

   return Flowable.using( driver::rxSession,
           session -> Flowable.fromPublisher( session.run( query, parameters ).records() ).map( record ->
record.get( 0 ).asString() ),
           RxSession::close );
}
public Flux<String> readProductTitlesReactor()
{
   String query = "MATCH (p:Product) WHERE p.id = $id RETURN p.title";
   Map<String,Object> parameters = Collections.singletonMap( "id", 0 );

   return Flux.using( driver::rxSession,
           session -> Flux.from( session.run( query, parameters ).records() ).map( record -> record.get( 0
).asString() ),
           RxSession::close );
}
```

More examples for using the Java driver Reactive API can be found in the org.neo4j.docs.driver package.

5.4. HTTP Server

In 4.0 we will extend the transactional endpoint to support managing multiple databases.

Meanwhile, we will also be removing a lot of previously deprecated HTTP endpoints such as RESTful endpoints, batch endpoint, JMX endpoint etc.

By adding support to manage multiple databases, the transaction endpoint `/db/data/transaction` is retained for backward-compatibility purposes. By sending queries to this endpoint, the query will be executed on the default database configured in server configuration.

To execute a query against a specified database, the new transactional endpoint: is:

```
db/databasename/transaction
```

For example, if we want to execute a query against the database `foo`, the endpoint would be:

```
db/foo/transaction
```

All old endpoints under transaction, such as `/transaction/commit`, are used in the same way as before against the database specified in path.

NOTE Unmanaged extensions are now supposed to work with `DatabaseManagementService` instead of `GraphDatabaseService` after management for multiple databases are introduced.

5.5. Drivers

New drivers are needed to explore the new Bolt Server features.

> **NOTE** Drivers will rename the 2.0 series to 4.0 to align the versioning between the client and the server before server GA.

The following table shows the feature availability in different 2.0 languages drivers:

Feature\Language	Java	Javascript	.NET	Python	Go
Multiple databases	✔	✔			
Back pressure	✔				

Besides new methods added to support 4.0 server features, we also made some updates to the driver public APIs across all language drivers:

- A new connection scheme `neo4j` is now introduced and preferred over *bolt+routing* and *bolt*. With the new `neo4j` scheme, it can connect the driver with either a cluster or a single instance.
- `Session` is split into three sessions, namely `Session`, `AsyncSession`, and `RxSession`.
- *v1* is removed in the drivers package name.
- Session parameters are introduced to accept database names, bookmarks and other session configurations on session creation.

Using the Java driver as an example, here are some useful suggestions for when migrating from an earlier driver version to the latest 2.0 pre-release Java driver:

- The driver classes should be re-imported as they may in a different package.
- The scheme of the initial URI used at driver creation need to be changed to `neo4j`. If you have used asynchronous sessions in your application, then the async sessions should be created with `driver.asyncSession` with the new driver.

The session parameters are now passed at session creation using lambda expressions.

5.6. Spring Boot

We provide a new Spring Boot starter for the Neo4j Java driver called *neo4j-java-driver-spring-boot-starter*.

This starter provides a convenient way to configure all aspects of the Neo4j-Java-Driver from within a Spring Boot application. It provides a single, managed Spring Bean of type `org.neo4j.driver.Driver`, configured to your needs.

The starter does not add any additional functionality on top of the driver, but only exposes the driver's configuration in a Spring friendly way. However, it configures the driver to use SLF4J logging by default.

As with any other Spring Boot starter, the only thing you have to do is to include the starter module via your dependency management, either with Maven or with Gradle:

Inclusion of the neo4j-java-driver-spring-boot-starter in a Maven project:

```
<dependency>
<groupId>org.neo4j.driver</groupId>
<artifactId>neo4j-java-driver-spring-boot-starter</artifactId>
<version>1.0.0-alpha02</version>
</dependency>
```

Inclusion of the neo4j-java-driver-spring-boot-starter in a Gradle project:

```
dependencies {
  compile 'org.neo4j.driver:neo4j-java-driver-spring-boot-starter:1.0.0-alpha02'
}
```

The starter brings all the dependency you need for your project, including the official Neo4j Java driver.

If you don't configure anything, than the starter assumes `bolt://localhost:7687` as Neo4j URI, and a server that has disabled authentication.

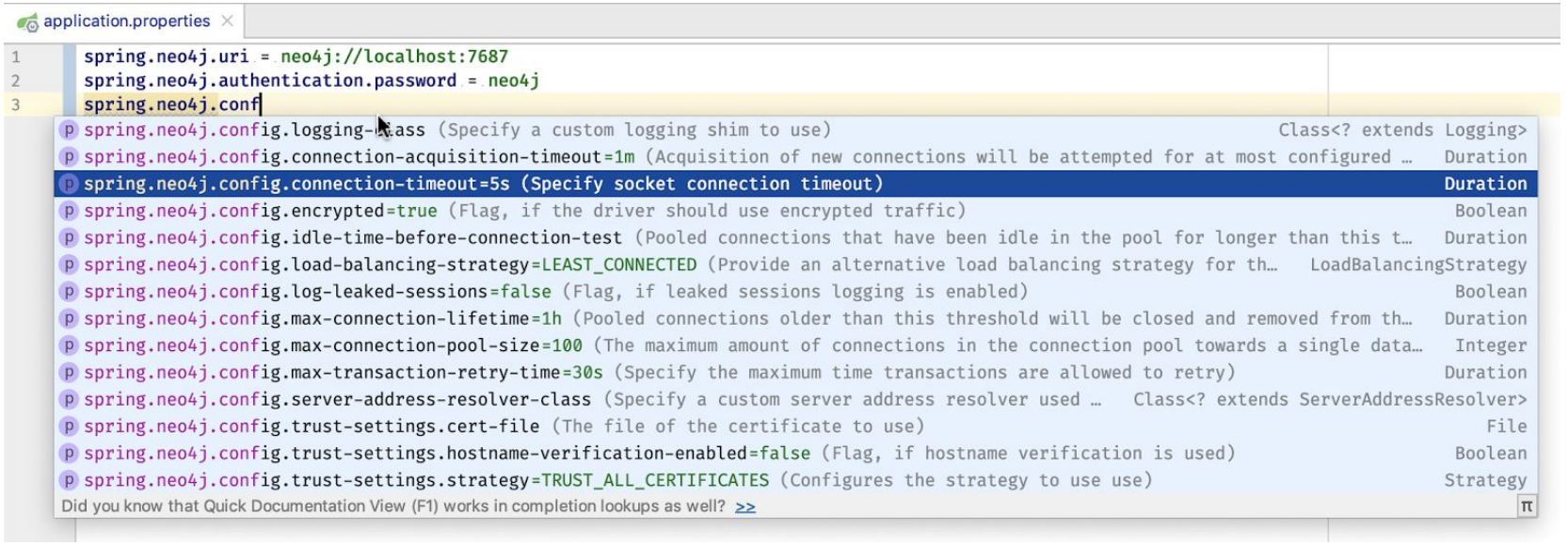
One possible configuration looks like this:

```
org.neo4j.driver.uri=bolt://localhost:7687
org.neo4j.driver.authentication.username=neo4j
org.neo4j.driver.authentication.password=secret
```

If only a single URI is provided, then the configuration tries to use that. Otherwise, it passes all URIs to the Java driver which in turn uses the first one that is a reachable `bolt+routing` instance.

The automatic configuration will fail fast if the driver cannot connect to a single Neo4j database or to a routing server.

The starter comes with meta data for your IDE that allows helpful autocomplete:



Your Spring context will contain a bean of type `org.neo4j.driver.Driver`. This is the native instance of the driver and you may use it directly. We provide examples on GitHub.

5.7. Cypher Shell

Cypher Shell in Neo4j 4.0 MR2 allows you to connect to a specific database by specifying the database name as argument. Optionally, you can select a different database once you are in Cypher Shell, using the `:use` command.

5.7.1. Initial Use of Cypher Shell

Change the password for user `neo4j`:

The following steps illustrate how to start Cypher Shell, and change the default password:

1. First log in with default password neo4j and connect to the system database:

```
$ bin/cypher-shell -u neo4j -p neo4j -d system
Connected to Neo4j 4.0.0 at bolt://localhost:7687 as user neo4j.
Type :help for a list of available commands or :exit to exit the shell.
Note that Cypher queries must end with a semicolon.
neo4j@system>
```

2. Then change from the default password:

```
neo4j@system> ALTER CURRENT USER SET PASSWORD FROM 'neo4j' TO 'secretpassword';
0 rows available after 107 ms, consumed after another 4 ms
Set 2 properties
neo4j@system>
```

3. Log out from Cypher Shell and log in again to the database of interest using the new password:

```
neo4j@system> :exit

Bye!
$ bin/cypher-shell -u neo4j -p secretpassword

Connected to Neo4j 4.0.0 at bolt://localhost:7687 as user neo4j.
Type :help for a list of available commands or :exit to exit the shell.
Note that Cypher queries must end with a semicolon.
neo4j@neo4j>
```

NOTE

In MR2 it is still possible to change your own password when connected to any database using the procedure **`CALL dbms.security.changePassword('secretpassword')`**, but this will be removed before the final release of Neo4j 4.0. Two key differences between the 3.x and the 4.x way of setting a new password:

- In 4.0 this action must be performed on the system database.
- In 4.0 it is necessary to know and use the original password when setting the new one.

5.7.2. New Cypher Shell Arguments

The new `-d` and `--database` arguments have been introduced to connect to a database. If the user does not specify any database, Cypher Shell connects to the default database (`neo4j` in fresh installations).

```
$ bin/cypher-shell --help
usage: cypher-shell [-h] [-a ADDRESS] [-u USERNAME] [-p PASSWORD] [--encryption {true,false}] [-d DATABASE]
[--format {auto,verbose,plain}] [--debug] [--non-interactive] [--sample-rows SAMPLE-ROWS] [--wrap
{true,false}] [-v] [--driver-version] [--fail-fast |
                    --fail-at-end] [cypher]
```

```
...

connection arguments:
  -a ADDRESS, --address ADDRESS
                        address and port to connect to (default: bolt://localhost:7687)
  -u USERNAME, --username USERNAME
                        username to connect as. Can also be specified using environment variable
NEO4J_USERNAME (default: )
  -p PASSWORD, --password PASSWORD
                        password to connect with. Can also be specified using environment variable
NEO4J_PASSWORD (default: )
  --encryption {true,false}
                        whether the connection to Neo4j should be encrypted; must be consistent with
Neo4j's configuration (default: true)
  -d DATABASE, --database DATABASE
                        database to connect to. Can also be specified using environment variable
NEO4J_DATABASE (default: )
$ bin/cypher-shell -a localhost:7687 -u neo4j -p secretpassword -d system
Connected to Neo4j 4.0.0 at bolt://localhost:7687 as user neo4j.
Type :help for a list of available commands or :exit to exit the shell.
Note that Cypher queries must end with a semicolon.
neo4j@system>
```

5.7.3. New Cypher Shell Environment variables

Cypher Shell now recognizes the environment variable `NEO4J_DATABASE`. If no `-d` or `--database` argument is given when starting Cypher Shell, the value of this environment variable is used to determine which database to connect to.

5.7.4. The `neo4j` Scheme

Cypher Shell accepts the new `neo4j` scheme as part of the URI used to connect to the DBMS. See section *5.5. Drivers* for information on the `neo4j` scheme.

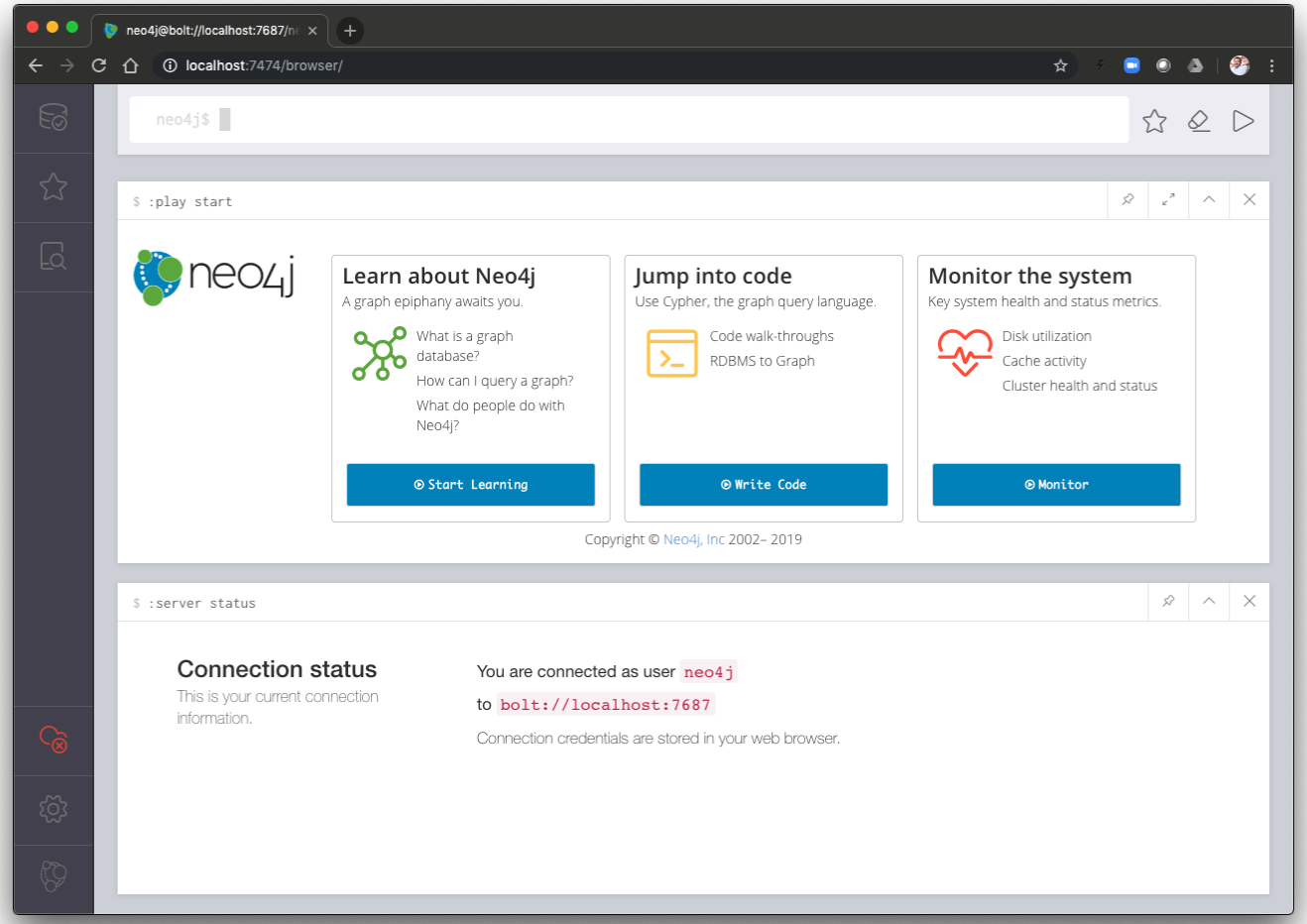
```
$ bin/cypher-shell -a neo4j://localhost:7687 -u neo4j -p secretpassword
Connected to Neo4j 4.0.0 at neo4j://localhost:7687 as user neo4j.
Type :help for a list of available commands or :exit to exit the shell.
Note that Cypher queries must end with a semicolon.
neo4j@neo4j>
```

5.7.5. The `:use` Command

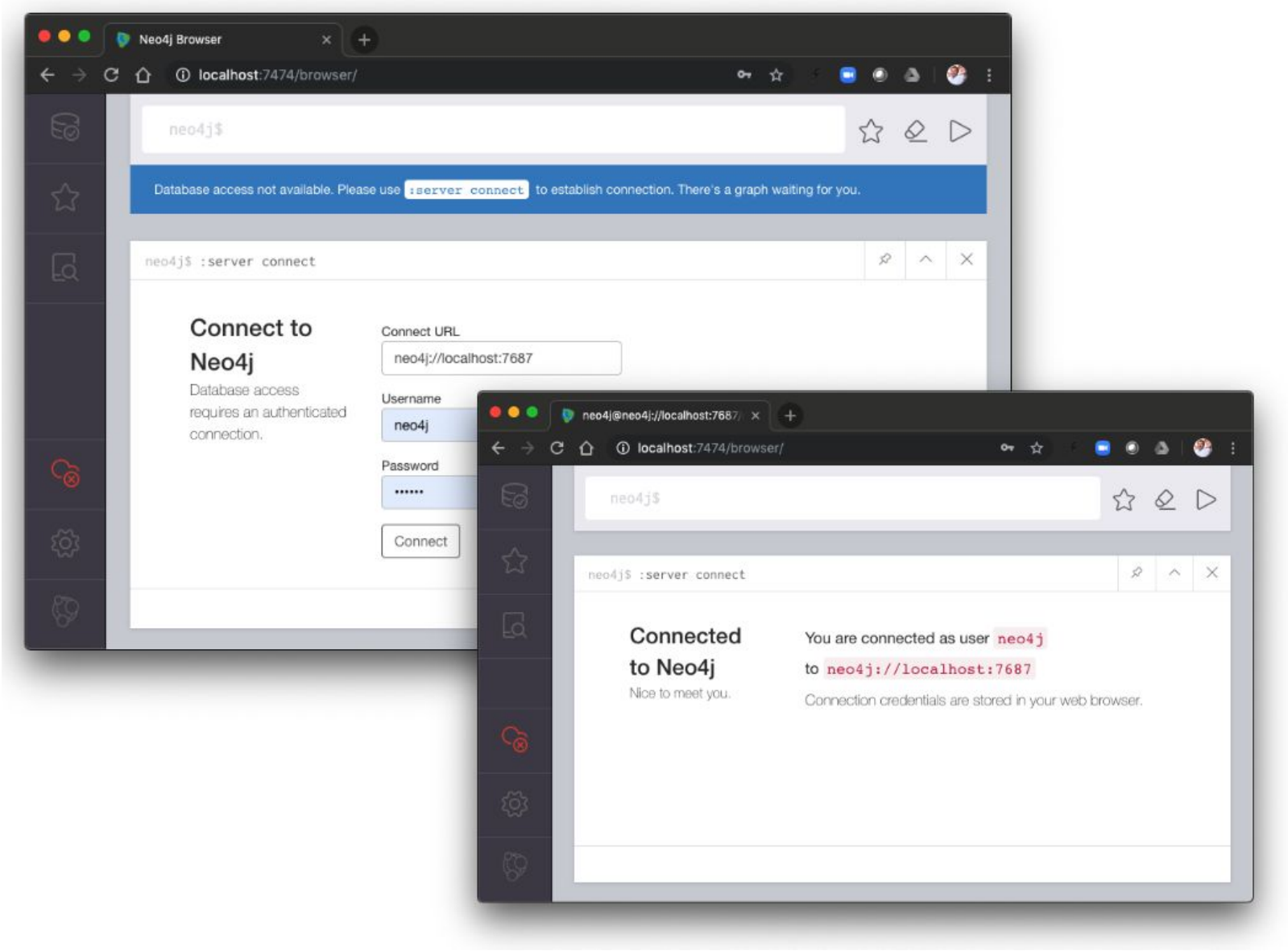
See section *3.3.0 Switching Databases* for details about this client-side command.

5.8. Neo4j Browser

Neo4j Browser in 4.0 MR2 enables the use of multiple databases. The HTTP server responds to the same URL as in previous releases, i.e. http://server:port/browser:

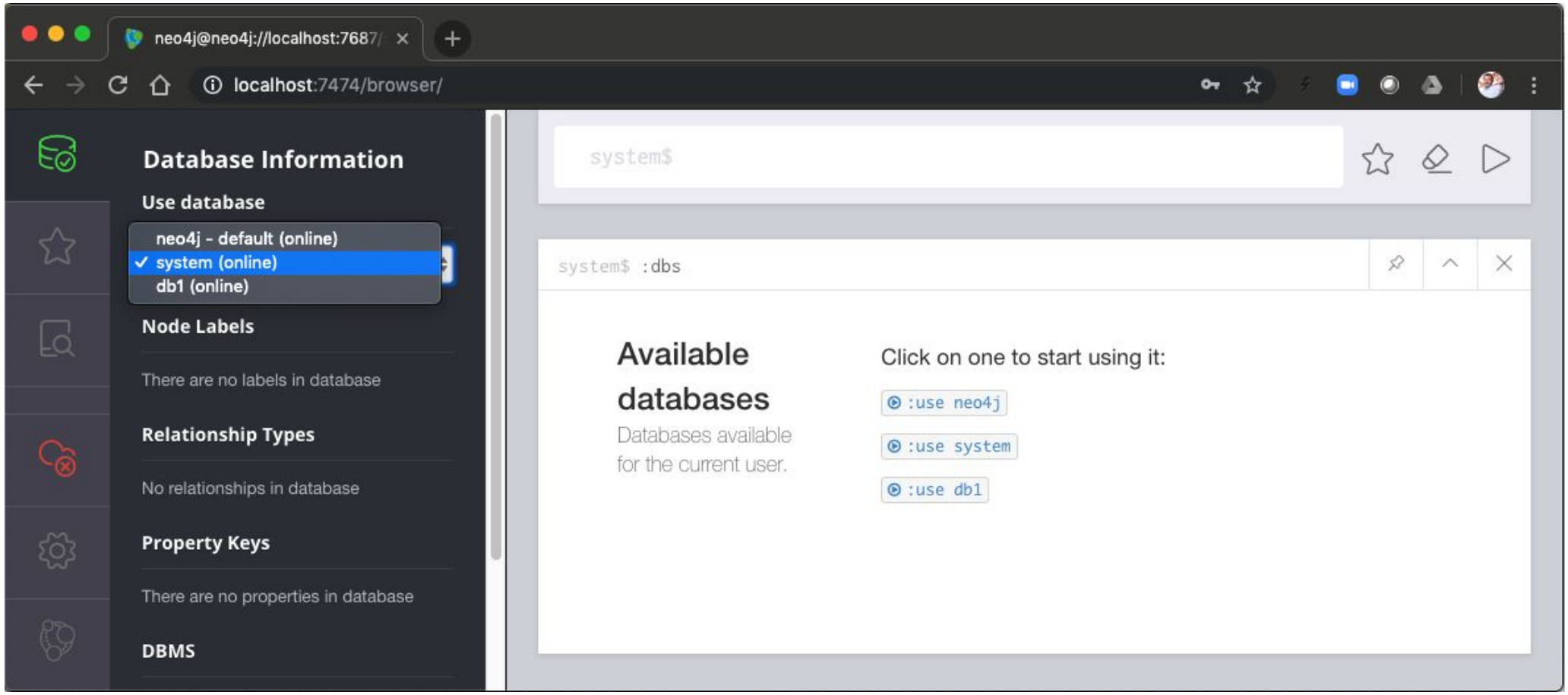


The user can use the `neo4j` scheme in the connect dialog. See section *5.5. Drivers* for information on the `neo4j` scheme. When first logging in to Neo4j Browser you get logged in to the default database:



5.8.1. Database Selection in Neo4j Browser

Users can issue the `:use` command to switch databases. See section *3.3.0 Switching Databases* for details about this client-side command. Alternatively, they can use the `:dbs` command to list the `:use` commands available, or they can open the sidebar and specify the database to use from the first combo box:



6. SDN-RX

6.1 Introduction

Spring Data Neo4j⚡RX, or SDN-RX, is a next-generation Spring Data module, created and maintained by Neo4j, in close collaboration with the Pivotal Spring Data Team.

SDN-RX relies completely on the Neo4j Java Driver, without introducing another "driver" or "transport" layer between the mapping framework and the driver. The Neo4j Java Driver - sometimes dubbed Bolt or the Bolt driver - is used as a protocol much like JDBC is with relational databases.

Noteworthy features that differentiate SDN-RX from Spring Data Neo4j + OGM are:

- Full support for immutable entities and thus full support for Kotlin's data classes right from the start.
- Full support for the reactive programming model in the Spring Framework itself and Spring Data.
- Brand new Neo4j client and reactive client feature, resurrecting the idea of a template over the plain driver, easing database access.

SDN-RX is currently developed with Spring Data Neo4j (https://github.com/spring-projects/spring-data-neo4j) in parallel and will eventually replace it when they are on feature parity in regards of repository support and mapping.

6.1.1. SDN-RX and Neo4j OGM

Neo4j OGM is an Object Graph Mapping library, which is mainly used by Spring Data Neo4j as its backend for the heavy lifting of mapping Nodes and Relationships into domain object. SDN-RX does not need, and does not support Neo4j-OGM. SDN-RX uses Spring Data's mapping context exclusively for scanning classes and building the meta model.

This pins SDN-RX to the Spring eco-systems, and it has several advantages, among them the smaller footprint in regards of CPU and memory usage and especially all the features of Springs mapping context.

SDN-RX has several features not present in SDN+OGM, notably:

- Full support for Springs reactive story, including reactive transaction.
- Full support for Query By Example.
- Full support for fully immutable entities.
- Support for all modifiers and variations of derived finder methods, including spatial queries.

Additionally, you cannot use *both* SDN-RX and Spring Data Neo4j simultaneously in a project since they are mutually exclusive.

NOTE SDN-RX does not support connections over HTTP to Neo4j.

6.1.2. SDN-RX and Embedded Neo4j

Embedded Neo4j has multiple facets to it:

- SDN-RX does not provide an embedded instance for your application.
- SDN-RX does not interact directly with an embedded instance. An embedded database is usually represented by an instance of `org.neo4j.graphdb.GraphDatabaseService`, and has no Bolt connector out of the box.
- SDN-RX can work with Neo4j's test harness; the test harness is specially meant to be a drop-in replacement for the real database. For more information, see Neo4j Client.

6.2 Getting Started

A Spring Boot starter is provided for SDN-RX.

As with any other Spring Boot starter, the only thing you have to do is to include the starter module via your dependency management. If you don't configure anything, then the starter assumes `bolt://localhost:7687` as Neo4j URI, and a server that has disabled authentication.

The SDN-RX starter depends on the starter for the Java Driver, which can be found here: https://github.com/neo4j/neo4j-java-driver-spring-boot-starter/blob/master/docs/manual.adoc.

Everything regarding configuration for the Java Driver, apply for SDN-RX also.

SDN-RX supports:

- The well-known and understood imperative a.k.a. blocking programming model (much like Spring Data JDBC or JPA).
- Reactive programming based on Reactive Streams, including full support for reactive transactions.

Those are all included in the same binary. The reactive programming model requires a Neo4j 4.0 server on the database side, and reactive Spring on the other. For examples, see the examples directory here: https://github.com/neo4j/sdn-rx/tree/master/examples

6.2.1. Preparing the Database

For this example, we use the movie graph demo which is available with every Neo4j instance (see https://neo4j.com/developer/movie-database/ for more information).

If you don't have a running database, but you have Docker installed you can run:

```
docker run --publish=7474:7474 --publish=7687:7687 neo4j:4.0.0-alpha09
```

You can now access the database at http://localhost:7474. Note that you will be prompted to change your password if this is the first time accessing this database.

You can now fill your database with some test data, using the movie graph demo.

6.2.2. Create a New Spring Boot Project

The easiest way to setup a Spring Boot project is https://start.spring.io/. If you do not want to use the website, the Spring Boot project is also integrated in the major IDEs.

Add the **Spring Web Starter** to get all the dependencies needed for creating a Spring based web application. The Spring Initializr will take care of creating a valid project structure for you, with all the files and settings in place for the selected build tool.

NOTE Do not select the **Spring Data Neo4j** dependencies here, as it will get you the previous generation of Spring Data Neo4j including OGM and additional abstraction over the driver.

6.2.2.1. Maven

You can issue a CURL request against the Spring Initializr to create a basic Maven project:

```
curl https://start.spring.io/starter.tgz \
  -d dependencies=webflux,actuator \
  -d bootVersion=2.2.0.M4 \
  -d baseDir=Neo4jSpringBootExample \
  -d name=Neo4j%20SpringBoot%20Example | tar -xzvf -
```

This will create a new folder `Neo4jSpringBootExample`. As this starter is not yet on the initializer, you'll have to add the following dependency manually to your *pom.xml* file:

```
<dependency>
```

```
        <groupId>org.neo4j.springframework.data</groupId>
        <artifactId>spring-data-neo4j-rx-spring-boot-starter</artifactId>
        <version>1.0.0-alpha03</version>
    </dependency>
```

In case of an existing project, you will also need to add the dependency manually.

6.2.2.2. Gradle

Using an approach similar to the Maven instructions above, you can generate a Gradle project:

```
curl https://start.spring.io/starter.tgz \
  -d dependencies=webflux,actuator \
  -d type=gradle-project \
  -d bootVersion=2.2.0.M4 \
  -d baseDir=Neo4jSpringBootExampleGradle \
  -d name=Neo4j%20SpringBoot%20Example | tar -xzvf -
```

The dependency for Gradle looks like this, and must be added to the *build.gradle* file:

```
dependencies {
    compile 'org.neo4j.springframework.data:spring-data-neo4j-rx-spring-boot-starter:1.0.0-alpha03'
}
```

In case of an existing project, you will also need to add the dependency manually.

6.2.2.3. Configuration

Now you can open any of those projects in your chosen IDE.

Find `application.properties` and configure your Neo4j credentials:

```
org.neo4j.driver.uri=bolt://localhost:7687
org.neo4j.driver.authentication.username=neo4j
org.neo4j.driver.authentication.password=secret
```

These credentials are the bare minimum of what is required to connect to a Neo4j instance.

NOTE SDN-RX repositories are automatically enabled by this starter and it is not necessary to add any programmatic configuration of the driver.

6.2.3. Creating a Domain

Your domain layer should accomplish two things:

- Map your Graph to objects.
- Provide access to objects.

6.2.3.1. Example Node-Entity

SDN-RX supports unmodifiable entities, for both Java and data classes in Kotlin. Therefore, this chapter will focus on immutable entities.

NOTE SDN-RX supports all the same data types that the Neo4j Java Driver supports. For more information, see *Example 4.1* here: https://neo4j.com/docs/driver-manual/current/cypher-values/#driver-neo4j-type-system.
Future versions will support additional converters.

```
import org.neo4j.springframework.data.core.schema.GeneratedValue;
import org.neo4j.springframework.data.core.schema.Id;
import org.neo4j.springframework.data.core.schema.Node;
import org.neo4j.springframework.data.core.schema.Property;

import org.springframework.data.annotation.PersistenceConstructor;

@Node("Movie")
public class MovieEntity {

      @Id @GeneratedValue
      private Long id;

      private final String title;

      @Property("tagline")
      private final String description;

      public MovieEntity(String title, String description) {
            this.id = null;
            this.title = title;
            this.description = description;
      }

      public Long getId() {
            return id;
      }
```

```
        public String getTitle() {
            return title;
        }

        public String getDescription() {
            return description;
        }

        public MovieEntity withId(Long id) {
            if (this.id == null) {
                return this;
            } else {
                MovieEntity newObject = new MovieEntity(this.title, this.description);
                newObject.id = this.id;
                return newObject;
            }
        }
    }
```

- `@Node` is used to mark this class as a managed entity. It also is used to configure the Neo4j label. The label defaults to the name of the class, if you're just using plain `@Node`.
- Each entity has to have an ID. The combination of `@Id` and `@GeneratedValue` configures SDN-RX to use Neo4j's internal ID.
- `@Property` is used as a way to use a different name for the field, rather than for the Graphs property.
- `public MovieEntity(String title, String description){}` is the constructor to be used by your application code. It sets the ID to null, as the field containing the internal ID should never be manipulated.
- `public MovieEntity withId(Long id) {}` creates a new entity and sets the field accordingly, without modifying the original entity, thus making it immutable.

NOTE Immutable entities using internally generated IDs are a bit contradictory, as SDN-RX needs a way to set the field with the value generated by the database.

The same entity using Project Lombok annotations (https://projectlombok.org/) for creating value objects is shown below:

MovieEntity.java

```
import lombok.Value;

import org.neo4j.springframework.data.core.schema.GeneratedValue;
import org.neo4j.springframework.data.core.schema.Id;
import org.neo4j.springframework.data.core.schema.Node;
import org.neo4j.springframework.data.core.schema.Property;

@Node("Movie")
@Value(staticConstructor = "of")
public class MovieEntity {

      @Id @GeneratedValue
      private Long id;

      private String title;

      @Property("tagline")
      private String description;
}
```

The corresponding entity as a Kotlin Data Class is shown below:

MovieEntity.kt

```
@Node("Movie")
data class MovieEntity (

    @Id
    @GeneratedValue
    val id: Long? = null,
```

```
        val title: String,

        @Property("tagline")
        val description: String
    )
```

6.2.3.2. Declaring Spring Data Repositories

There are two options:

- You can work store agnostic with SDN-RX and make your domain specific extends one of:
  - `org.springframework.data.repository.Repository`
  - `org.springframework.data.repository.CrudRepository`
  - `org.springframework.data.repository.reactive.ReactiveCrudRepository`
  - `org.springframework.data.repository.reactive.ReactiveSortingRepository`

  You can select imperative or reactive accordingly.

- Settle on a store-specific implementation, and gain all the methods we support out of the box.

The advantage of the 2nd option is also the biggest disadvantage. Once out, all those methods will be part of your API. Most of the time it's harder to take something away, than add. Furthermore, using store specifics leaks your store into your domain. From a performance point of view however, there is no penalty.

The example below demonstrates the first option, which is a store-agnostic method, based upon the Movie Entities examples above:

MovieRepository.java

```
    import reactor.core.publisher.Flux;
    import reactor.core.publisher.Mono;
```

```
import org.springframework.data.domain.Example;
import org.springframework.data.repository.reactive.ReactiveCrudRepository;

public interface MovieRepository extends ReactiveCrudRepository<MovieEntity, Long> {

      Mono<MovieEntity> findOneByTitle(String title);

      Flux<MovieEntity> findAll(Example<MovieEntity> example);
}
```

NOTE The declaration of these two methods is purely optional; if not needed, don't add them. These are reused in later examples.

This repository can be used in any Spring component like this:

MovieController.java

```
import reactor.core.publisher.Flux;
import reactor.core.publisher.Mono;

import org.neo4j.springframework.data.examples.spring_boot.domain.MovieEntity;
import org.neo4j.springframework.data.examples.spring_boot.domain.MovieRepository;
import org.springframework.http.MediaType;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

@RestController
```

```
@RequestMapping("/movies")
public class MovieController {

      private final MovieRepository movieRepository;

      public MovieController(MovieRepository movieRepository) {
            this.movieRepository = movieRepository;
      }

      @PutMapping
      Mono<MovieEntity> createOrUpdateMovie(@RequestBody MovieEntity newMovie) {
            return movieRepository.save(newMovie);
      }

      @GetMapping(value = { "", "/" }, produces = MediaType.TEXT_EVENT_STREAM_VALUE)
      Flux<MovieEntity> getMovies() {
            return movieRepository
                  .findAll();
      }

      @GetMapping("/by-title")
      Mono<MovieEntity> byTitle(@RequestParam String title) {
            return movieRepository.findOneByTitle(title);
      }

      @DeleteMapping("/{id}")
      Mono<Void> delete(@PathVariable Long id) {
            return movieRepository.deleteById(id);
      }
}
```

NOTE

Testing reactive code is done with a `reactor.test.StepVerifier`.
For more information, see the Project Reactor documentation: https://projectreactor.io/docs/core/release/reference/#testing

6.3. Neo4j Client

SDN-RX comes with a Neo4j client, providing a human-readable layer on top of the Neo4j Java driver.

It has the following main goals:

- Integrate into Springs transaction management, for both imperative and reactive scenarios.
- Participate in JTA-Transactions if necessary.
- Provide a consistent API for both imperative and reactive scenarios.
- Not add any mapping overhead.

SDN-RX relies on all those features and uses them to fulfill it's entity mapping features.

The Neo4j Java Driver (https://github.com/neo4j/neo4j-java-driver) is a versatile tool and provides an asynchronous API, in addition to the imperative and reactive versions. SDN-RX uses the Java driver as directly as possible, while also being as user-friendly and idiomatic as possible.

The Neo4j client comes in two flavors:

- `org.neo4j.springframework.data.core.Neo4jClient`
- `org.neo4j.springframework.data.core.ReactiveNeo4jClient`

While both versions provide an API using the same vocabulary and syntax, they are not API compatible. Both versions feature the same, fluent API to specify queries, bind parameters and extract results.

6.3.1. Imperative and Reactive

Interactions with a Neo4j client usually ends with a call to:

- `fetch().one()`

- `fetch().first()`
- `fetch().all()`
- `run()`

The imperative version will interact at this moment with the database and get the requested results or summary, wrapped in a `Optional<>` or a `Collection`.

The reactive version will in contrast return a publisher of the requested type. Interaction with the database and retrieval of the results will not happen until the publisher is subscribed to. The publisher can only be subscribed once.

6.3.2. Getting an Instance of the Client

With SDN-RX, both clients depend on a configured driver instance. The following will create an instance of the *imperative* Neo4j client:

```
import org.neo4j.driver.AuthTokens;
import org.neo4j.driver.Driver;
import org.neo4j.driver.GraphDatabase;

import org.neo4j.springframework.data.core.Neo4jClient;

public class Demo {

    public static void main(String...args) {

        Driver driver = GraphDatabase
            .driver("neo4j://localhost:7687", AuthTokens.basic("neo4j", "secret"));

        Neo4jClient client = Neo4jClient.create(driver);
    }
}
```

NOTE The driver can only open a reactive session against a 4.0 database and will fail with an exception on any lower version.

The following will create an instance of the *reactive* Neo4j client:

```
import org.neo4j.driver.AuthTokens;
import org.neo4j.driver.Driver;
import org.neo4j.driver.GraphDatabase;

import org.neo4j.springframework.data.core.ReactiveNeo4jClient;

public class Demo {

    public static void main(String...args) {

        Driver driver = GraphDatabase
            .driver("neo4j://localhost:7687", AuthTokens.basic("neo4j", "secret"));

        ReactiveNeo4jClient client = ReactiveNeo4jClient.create(driver);
    }
}
```

NOTE Make sure you use the same driver instance for the client as you used for providing a `Neo4jTransactionManager` or `ReactiveNeo4jTransactionManager`, in case you have enabled transactions.
The client won't be able to synchronize transactions if you use another instance of a driver.

The Spring Boot starter provides ready-to-use beans of the Neo4j client that fit the environment (imperative or reactive) and you usually don't have to configure your own instance.

6.3.3. Usage

6.3.3.1. Selecting the Target Database

The Neo4j client is prepared for use with database management features of Neo4j 4.0; the client uses the default database unless you specify otherwise. The fluent API of the client is enabled to specify the target database exactly once, after the declaration of the query to execute.

The example below demonstrates this with a reactive client, and a target database named `neo4j`:

```
Flux<Map<String, Object>> allActors = client
      .query("MATCH (p:Person) RETURN p")
      .in("neo4j")
      .fetch()
      .all();
```

6.3.3.2. Specifying Queries

The interaction with the clients starts with a query. A query can be defined by a plain `String` or a `Supplier<String>`. The supplier will be evaluated as late as possible and can be provided by any query builder. For example:

```
Mono<Map<String, Object>> firstActor = client
      .query(() -> "MATCH (p:Person) RETURN p")
      .fetch()
      .first();
```

6.3.3.3. Retrieving Results

As the previous examples show, the interaction with the client always ends with a call to fetch and specifying how many results shall be received. Both reactive and imperative client offer:

- `one()` - Expect exactly one result from the query.
- `first()` - Expect results and return the first record.
- `all()` - Retrieve all records returned.

The imperative client returns `Optional<T>` and `Collection<T>` respectively, while the reactive client returns `Mono<T>` and `Flux<T>`, the later one being executed only when subscribed to.

If you don't expect any results from your query, then use `run()` after specifying the query.

The following example demonstrates retrieving result summaries in a *reactive* way:

```
Mono<ResultSummary> summary = reactiveClient
    .query("MATCH (m:Movie) where m.title = 'Aeon Flux' DETACH DELETE m")
    .run();

summary
    .map(ResultSummary::counters)
    .subscribe(counters ->
        System.out.println(counters.nodesDeleted() + " nodes have been deleted")
    );
```

The actual query is triggered above by subscribing to the publisher.

The following example demonstrates retrieving result summaries in an *imperative* way:

```
ResultSummary resultSummary = imperativeClient
      .query("MATCH (m:Movie) where m.title = 'Aeon Flux' DETACH DELETE m")
      .run();

SummaryCounters counters = resultSummary.counters();
System.out.println(counters.nodesDeleted() + " nodes have been deleted")
```

In this example, the query is triggered immediately.

6.3.3.4. Mapping Parameters

Queries can contain named parameters (`$someName`), and the Neo4j client allows for comfortable binding.

NOTE The client doesn't check whether all parameters are bound or whether there are too many values - that is left to the driver. However, the client does prevent you from using a parameter name twice.

You can either map simple types that the Java driver understands or complex classes. For more information, see https://neo4j.com/docs/driver-manual/current/cypher-values/#driver-neo4j-type-system, where simple types are described.

```
Map<String, Object> parameters = new HashMap<>();
parameters.put("name", "Li.*");

Flux<Map<String, Object>> directorAndMovies = client
      .query(
            "MATCH (p:Person) - [:DIRECTED] -> (m:Movie {title: $title}), (p) - [:WROTE] -> (om:Movie) " +
                  "WHERE p.name =~ $name " +
                  "  AND p.born < $someDate.year " +
```

```
                "RETURN p, om"
        )
        .bind("The Matrix").to("title")
        .bind(LocalDate.of(1979, 9, 21)).to("someDate")
        .bindAll(parameters)
        .fetch()
        .all();
```

In the example above, there is a fluent API for binding simple types. Alternatively, parameters can be bound via a map of named parameters.

SDN-RX does a lot of complex mapping and it uses the same API that you can use from the client.

You can provide a `Function<T, Map<String, Object>>` for any given domain object in order to map those domain objects to parameters that the driver can understand.

The following example demonstrates a domain type:

```
public class Director {

    private final String name;

    private final List<Movie> movies;

    Director(String name, List<Movie> movies) {
        this.name = name;
        this.movies = new ArrayList<>(movies);
    }

    public String getName() {
        return name;
    }
```

```
        public List<Movie> getMovies() {
            return Collections.unmodifiableList(movies);
        }
    }

    public class Movie {

        private final String title;

        public Movie(String title) {
            this.title = title;
        }

        public String getTitle() {
            return title;
        }
    }
```

The mapping function has to fill in all named parameters that might occur in the query. The following example demonstrates how to use a mapping function for binding domain objects:

```
    Director joseph = new Director("Joseph Kosinski",
            Arrays.asList(new Movie("Tron Legacy"), new Movie("Top Gun: Maverick")));

    Mono<ResultSummary> summary = client
        .query(""
            + "MERGE (p:Person {name: $name}) "
            + "WITH p UNWIND $movies as movie "
            + "MERGE (m:Movie {title: movie}) "
            + "MERGE (p) - [o:DIRECTED] -> (m) "
        )
```

```
        .bind(joseph).with(director -> {
            Map<String, Object> mappedValues = new HashMap<>();
            List<String> movies = director.getMovies().stream()
                .map(Movie::getTitle).collect(Collectors.toList());
            mappedValues.put("name", director.getName());
            mappedValues.put("movies", movies);
            return mappedValues;
        })
        .run();
```

In the example, the `with` method enables you to specify the binder function.

6.3.3.5. Working with Result Objects

Both clients return collections or publishers of maps (`Map<String, Object>`). Those maps correspond exactly with the records that a query might have produced.

In addition, you can plugin your own `BiFunction<TypeSystem, Record, T>` through `fetchAs` to reproduce your domain object.

```
    Mono<Director> lily = client
        .query(""
            + " MATCH (p:Person {name: $name}) - [:DIRECTED] -> (m:Movie)"
            + "RETURN p, collect(m) as movies")
        .bind("Lilly Wachowski").to("name")
        .fetchAs(Director.class).mappedBy((TypeSystem t, Record record) -> {
            List<Movie> movies = record.get("movies")
                .asList(v -> new Movie((v.get("title").asString())));
            return new Director(record.get("name").asString(), movies);
        })
        .one();
```

`TypeSystem` gives access to the types the underlying Java driver used to fill the record.

6.3.3.6. Interacting Directly with the Driver While Using Managed Transactions

In case you don't want the opinionated "client" approach of the `Neo4jClient` or the `ReactiveNeo4jClient`, you can have the client delegate all interactions with the database to your code. The interaction after the delegation is slightly different with the imperative and reactive versions of the client.

The imperative version takes in a `Function<StatementRunner, Optional<T>>` as a callback. Additionally, it is possible to return an empty optional.

The following example demonstrates how to delegate database interaction to an *imperative* `StatementRunner`:

```
Optional<Long> result = client
    .delegateTo((StatementRunner runner) -> {
        // Do as many interactions as you want
        long numberOfNodes = runner.run("MATCH (n) RETURN count(n) as cnt")
            .single().get("cnt").asLong();
        return Optional.of(numberOfNodes);
    })
    // .in("aDatabase")
    .run();
```

The following example demonstrates how to delegate database interaction to a *reactive* `RxStatementRunner`:

```
Mono<Integer> result = client
    .delegateTo((RxStatementRunner runner) ->
        Mono.from(runner.run("MATCH (n:Unused) DELETE n").summary())
            .map(ResultSummary::counters)
            .map(SummaryCounters::nodesDeleted))
```

```
        // .in("aDatabase")
        .run();
```

In both of the examples above, the types of runner has only been stated to provide more clarity.

6.4. Migrating from SDN+OGM to SDN-RX

NOTE As the relationship mapping of SDN-RX is not yet fully complete, those topics are not addressed here.
Additionally, SDN-RX is still in alpha, so things are liable to change in the near future. The content in this chapter should be seen as an ongoing effort, that can help when considering your current state and where you might want to go.

6.4.1. Known Issues with Past SDN+OGM Migrations

The main issues observed when migrating from older versions of Spring Data Neo4j to newer ones are:

- **Skipping more than one major upgrade**:
  While Neo4j-OGM can be used stand-alone, Spring Data Neo4j cannot. It depends, to a large extent, on the Spring Data and therefore, on the Spring Framework itself, which eventually affects large parts of your application. Depending on how the application has been structured, and how much any of the framework part leaked into your business code, the more you have to adapt your application.
  It is even more challenging when you have more than one Spring Data module in your application, and if you accessed a relational database in the same service layer as your graph database.
  Updating two object mapping frameworks can also be challenging.

- **Relying on a embedded database configured through Spring Data itself**R:
  The embedded database in a SDN+OGM project is configured by Neo4j-OGM. For example, if you want to upgrade from Neo4j 3.0 to 3.5, you can't without upgrading your whole application. If you choose to embed a database into your application, you tie yourself into the modules that configure this embedded database. To have another embedded database version, you will have to upgrade the module that configured it,

because the old one does not support the new database. As there is always a Spring Data version corresponding to Neo4j-OGM, you would have to upgrade that as well. Spring Data, however, depends on Spring Framework and then the arguments from the first bullet apply.

- **Being unsure about which building blocks to include:**
  It's not easy to get the terms right. The building blocks of an SDN+OGM setting are described here:
  https://michael-simons.github.io/neo4j-sdn-ogm-tips/what_are_the_building_blocks_of_sdn_and_ogm.html.
  It may be that you are dealing with a lot of conflicting dependencies.

Backed by those observations, we recommend to make sure you're using only the Bolt or HTTP transport in your current application before switching from SDN+OGM to SDN-RX. Thus, your application and the access layer of your application is to a large extent independent from the databases version. From that state, consider moving from SDN+OGM to SDN-RX.

6.4.2. Preparation for Migration from SDN+OGM Lovelace or SDN+OGM Moore

Prior to making the switch, you will need to ensure that your application runs against Neo4j in server mode over the Bolt protocol. There are three possible states for your application:

> **NOTE** The *Lovelace* release train corresponds to SDN 5.1.x and OGM 3.1.x, while the *Moore* is SDN 5.2.x and OGM 3.2.x.

- **Using Neo4j embedded**
  If you have added `org.neo4j:neo4j-ogm-embedded-driver` and `org.neo4j:neo4j` to you project and you are starting the database via OGM facilities, you will have to remove those dependencies since this is no longer supported. You will have to set up a standard Neo4j server instead (both stand-alone and cluster are supported).

- **Using the HTTP Transport**
  If you have added `org.neo4j:neo4j-ogm-http-driver` and configured a URL like `http://user:password@localhost:7474`, you will need to remove this dependency and configure a Bolt url like `bolt://localhost:7687` instead, or use the new `neo4j://` protocol, which also takes care of routing.

- **Using Bolt indirectly**
  If you are using a default SDN+OGM project, you can keep your existing URL since this uses `org.neo4j:neo4j-ogm-bolt-driver` and thus indirectly, the pure Java Driver.

6.4.3 Migrating

Once you have made sure that your SDN+OGM application works over Bolt as expected, you can start migrating to SDN-RX by the following steps:

1. Remove all `org.neo4j:neo4j-ogm-*` dependencies.
2. Remove `org.springframework.data:spring-data-neo4j`.
3. Adapt the properties for the URL and authentication as per the example below:

```
a. # Old
b. spring.data.neo4j.embedded.enabled=false # No longer support
c. spring.data.neo4j.uri=bolt://localhost:7687
d. spring.data.neo4j.username=neo4j
e. spring.data.neo4j.password=secret
f.
g. # New
h. org.neo4j.driver.uri=bolt://localhost:7687
i. org.neo4j.driver.authentication.username=neo4j
j. org.neo4j.driver.authentication.password=secret
```

NOTE The new properties above might change in the future when SDN-RX and the driver will eventually replace the old setup.

4. Add the new dependencies according to the previous sections on Gradle and Maven.

Keep in mind that configuring SDN-RX through a `ord.neo4j.ogm.config.Configuration` bean is not supported. Instead, all configuration of the driver is done through our new starter (see section 6.2.2 for more information). Configuration of other properties than the ones mentioned above can be done through standard Spring Boot means.

You can now also replace the annotations:

Old	New
`org.neo4j.ogm.annotation.NodeEntity`	`org.neo4j.springframework.data.core.schema.Node`
`org.neo4j.ogm.annotation.GeneratedValue`	`org.neo4j.springframework.data.core.schema.GeneratedValue`
`org.neo4j.ogm.annotation.Id`	`org.neo4j.springframework.data.core.schema.GeneratedValue`
`org.neo4j.ogm.annotation.Property`	`org.neo4j.springframework.data.core.schema.Property`
`org.neo4j.ogm.annotation.Relationship`	`org.neo4j.springframework.data.core.schema.Relationship`

NOTE Several Neo4j-OGM annotations do not yet have a corresponding annotation in SDN-RX, and some never will.
We will add to this list when we add more features.

7. Other Features

7.1. Index Population for the Native Index Provider

Index population is the task executed by the `CREATE INDEX` and `CREATE CONSTRAINT` commands and is done automatically by the DBMS when the index is missing. This task is non-blocking when it refers to a `CREATE INDEX` command and blocking when it refers to a `CREATE CONSTRAINT` command.

7.1.1. Improvement in Index Population

A major improvement in index population is the new index population algorithm that provides significant time-saving benefits. You can see the results for yourself by creating an index on a set of nodes with a given label and property/properties and then check the time consumed by the creation. This can be done in two different ways:

1. By checking the `debug.log` file, which contains entries created by the index population functions.
2. By executing the `awaitIndex` procedure. You should consider that this procedure blocks the client connection until the index has been fully built. The following procedure will return the time necessary to build the index:

```
neo4j@neo4j> CREATE INDEX ON :N1(p0);
0 rows available after 83 ms, consumed after another 0 ms
Added 1 indexes
neo4j@neo4j> CALL db.awaitIndex(":N1(p0)");
0 rows available after 29487 ms, consumed after another 0 ms
neo4j@neo4j>
```

7.2. Native Index Provider Max Key Size

In Neo4j 4.0 MR2, the native index provider has extended the maximum key size from approximately 4KB to approximately 8KB. This size should satisfy the vast majority of the use cases.

NOTE In case of list properties, the key size considers all the values in the list.

7.3. Lucene Index Provider

Neo4j 4.0 MR2 includes the Lucene Index Provider for full-text search. The index provider is now based on the latest stable version of Lucene, 8.1.0.