Table of Contents ⊞

8.6. Subgraph access control Enterprise Edition

This section describes how to configure subgraph access control in Neo4j.

This section describes the following:

- Introduction
- Configuration steps
  - Create a custom role
  - Manage procedure permissions

8.6.1. Introduction

It is possible to restrict a user's access to, and subsequent actions on, specified portions of the graph. For example, a user can be allowed to read, but not write, nodes with specific labels and relationships with certain types.

To implement subgraph access control, you must complete the following steps:

1. Put a procedure or function in place that performs the reads from, and/or writes to, the portion of the graph that you wish to control. This can either be developed in-house, or be made available as a third-party library. Please refer to Java Reference → Extending Neo4j (../../../../java-reference/3.5/extending-neo4j/) for a description on creating and using user-defined procedures and functions.
2. Create one, or several, custom roles, with which to run the procedure described above. These roles can subsequently be assigned the relevant privileges.
3. Configure the procedure so that it can be executed by users with the custom roles.

The steps below assume that the procedure or function is already developed and installed.

8.6.2. Configuration steps

8.6.2.1. Create a custom role

Create a custom role and manage it either through native user management or through federated user management with LDAP.

Native users scenario

In the native users scenario, a custom role is created and assigned to the relevant user(s).

Example 8.19. Native users scenario

In this example, we will use Cypher to create a custom `accounting` role and assign it to a pre-existing user, `billsmith`.

```
CALL dbms.security.createRole('accounting')
CALL dbms.security.addRoleToUser('accounting', 'billsmith')
```

Federated users scenario (LDAP)

In the LDAP scenario, the LDAP user group is mapped to a custom role in Neo4j.

Example 8.20. Federated users scenario (LDAP)

In this example, we will use Cypher to create a custom `accounting` role.

```
CALL dbms.security.createRole('accounting')
```

We will then map the `accounting` role to the LDAP group with groupID `101`.

```
dbms.security.realms.ldap.authorization.group_to_role_mapping=101=accounting
```

8.6.2.2. Manage procedure permissions

In standard use, procedures and functions are executed according to the same security rules as regular Cypher statements, as described in Section 8.4.1, “Native roles” (../native-user-role-management/native-roles/). For example, users assigned any one of the native roles `publisher`, `architect` and `admin` will be able to execute a procedure with `mode=WRITE`, whereas a user assigned only the `reader` role will not be allowed to execute the procedure.

For the purpose of subgraph access control, we allow specific roles to execute procedures that they would otherwise be prevented from accessing through their assigned native roles. The user is given the privilege that comes with the mode of the procedure, during the execution of the procedure only. The following two parameters are used to configure the desired behavior:

`dbms.security.procedures.default_allowed (../../reference/configuration-settings/#config_dbms.security.procedures.default_allowed)`

The setting `dbms.security.procedures.default_allowed` defines a single role that is allowed to execute any procedure or function that is not matched by the `dbms.security.procedures.roles` configuration.

Example 8.21. Configure a default role that can execute procedures and functions

Assume that we have the following configuration:

```
dbms.security.procedures.default_allowed=superAdmin
```

This will have the following effects:

- If the setting `dbms.security.procedures.roles` is left unconfigured, the role `superAdmin` will be able to execute all custom procedures and functions.
- If the setting `dbms.security.procedures.roles` has some roles and functions defined, the role `superAdmin` will be able to execute all custom procedures and functions that are *not* configured by `dbms.security.procedures.roles`.

`dbms.security.procedures.roles (../../reference/configuration-settings/#config_dbms.security.procedures.roles)`

The `dbms.security.procedures.roles` setting provides fine-grained control over procedures.

Example 8.22. Configure roles for the execution of specific procedures

Assume that we have the following configuration:

```
dbms.security.procedures.roles=apoc.convert.*:Converter;apoc.load.json.*:Converter,DataSource;apoc.trigger.add:TriggerHappy
```

This will have the following effects:

- All users with the role `Converter` will be able to execute all procedures in the `apoc.convert` namespace.
- All users with the roles `Converter` and `DataSource` will be able to execute procedures in the `apoc.load.json` namespace.
- All users with the role `TriggerHappy` will be able to execute the specific procedure `apoc.trigger.add`.

A procedure will fail if it attempts to execute database operations that violates its mode. For example, a procedure assigned the mode of `READ` will fail if it is programmed to do write actions. This will happen regardless of user or role configuration.