The Neo4j Java Developer Reference v3.4

Table of Contents

This document contains information on advanced Java-centric usage of Neo4j. Among the topics covered are embedding Neo4j in your own software and writing extensions.

You might want to keep the Neo4j JavaDocs (https://neo4j.com/docs/java-reference/3.4/javadocs/) handy while reading!

The main parts of this reference are:

- Extending Neo4j — How to build unmanaged extensions and procedures.
- REST API authorization rules — How to use the REST API for restricting access to Neo4j.
- Setup for remote debugging — How to configure the Neo4j server for remote debugging sessions.
- Using Neo4j embedded in Java applications — Instructions on embedding Neo4j in an application.
- The traversal framework — A walkthrough of the traversal framework.
- Manual indexing — How to use explicit indexes.
- Transaction management — Details on transaction semantics in Neo4j.
- Online Backup from Java — How to perform an online backup in Java.
- JMX metrics — How to monitor Neo4j with JMX and a reference of available metrics.

Chapter 1. Extending Neo4j

Neo4j provides a pluggable infrastructure for extensions. Procedures extend the capabilities of the Cypher query language. Server extensions allow new surfaces to be created in the REST API. Both require the user to be familiar with the Java programming language and to have an environment set up for compiling Java code.

When running your own code and Neo4j in the same JVM, there are a few things you should keep in mind:

- Don't create or retain more objects than you strictly need to. Large caches in particular tend to promote more objects to the old generation, thus increasing the need for expensive full garbage collections.
- Don't use internal Neo4j APIs. They are internal to Neo4j and subject to change without notice, which may break or change the behavior of your code.
- If possible, avoid using Java object serialization or reflection in your code or in any runtime dependency that you use. Otherwise, if you cannot avoid using Java object serialization and reflection, then ensure that the `-XX:+TrustFinalNonStaticFields` JVM flag is disabled in `neo4j.conf`.

1.1. Procedures

User-defined procedures are written in Java, deployed into the database, and called from Cypher.

A *procedure* is a mechanism that allows Neo4j to be extended by writing custom code which can be invoked directly from Cypher. Procedures can take arguments, perform operations on the database, and return results.

Procedures are written in Java and compiled into *jar* files. They can be deployed to the database by dropping a *jar* file into the *$NEO4J_HOME/plugins* directory on each standalone or clustered server. The database must be re-started on each server to pick up new procedures.

Procedures are the preferred means for extending Neo4j. Examples of use cases for procedures are:

1. To provide access to functionality that is not available in Cypher, such as explicit indexes and schema introspection.
2. To provide access to third party systems.
3. To perform graph-global operations, such as counting connected components or finding dense nodes.
4. To express a procedural operation that is difficult to express declaratively with Cypher.

1.1.1. Calling procedures

To call a stored procedure, use a Cypher `CALL` clause. The procedure name must be fully qualified, so a procedure named `findDenseNodes` defined in the package `org.neo4j.examples` could be called using:

```
CALL org.neo4j.examples.findDenseNodes(1000)
```

A `CALL` may be the only clause within a Cypher statement or may be combined with other clauses. Arguments can be supplied directly within the query or taken from the associated parameter set. For full details, see the Cypher documentation on the `CALL` clause.

1.1.2. Built-in procedures

This section shows the built-in procedures that are bundled with Neo4j.

Neo4j comes bundled with a number of built-in procedures. These can be used to:

- Inspect schema.
- Inspect meta data.
- Explore procedures and components.
- Monitor management data.
- Set user password.

These are listed in the table below:

Procedure name	Command to invoke procedure	What it does
`ListLabels`	`CALL db.labels()`	List all labels in the database.
`ListRelationshipTypes`	`CALL db.relationshipTypes()`	List all relationship types in the database.
`ListPropertyKeys`	`CALL db.propertyKeys()`	List all property keys in the database.
`ListIndexes`	`CALL db.indexes()`	List all indexes in the database.
`AwaitIndex`	`CALL db.awaitIndex(label, property, timeOutSeconds)`	Wait for the specified index to come online.
`AwaitIndexes`	`CALL db.awaitIndexes( timeOutSeconds)`	Wait for the all index to come online.
`ListConstraints`	`CALL db.constraints()`	List all constraints in the database.
`ListProcedures`	`CALL dbms.procedures()`	List all procedures in the DBMS.
`ListComponents`	`CALL dbms.components()`	List DBMS components and their versions.
`QueryJmx`	`CALL dbms.queryJmx(query)`	Query JMX management data by domain and name. For instance, "org.neo4j:*".
`AlterUserPassword`	`CALL dbms.changePassword(query)`	Change the user password.
`ListQueries`	`CALL dbms.listQueries()`	List all queries currently executing at this instance that are visible to the user.
`ListTransactions`	`CALL dbms.listTransactions()`	List all transactions currently executing at this instance that are visible to the user.
`ListActiveLocks`	`CALL dbms.listActiveLocks()`	List the active lock requests granted for the transaction executing the query with the given query id.

1.1.3. User-defined procedures

This section covers how to write, test and deploy a procedure for Neo4j.

Custom procedures are written in the Java programming language. Procedures are deployed via a *jar* file that contains the code itself along with any dependencies (excluding Neo4j). These files should be placed into the *plugin* directory of each standalone database or cluster member and will become available following the next database restart.

The example that follows shows the steps to create and deploy a new procedure.

The example discussed below is available as a repository on GitHub *(https://github.com/neo4j-examples/neo4j-procedure-template)*. To get started quickly you can fork the repository and work with the code as you follow along in the guide below.

Set up a new project

A project can be set up in any way that allows for compiling a procedure and producing a *jar* file. Below is an example configuration using the Maven *(https://maven.apache.org/)* build system. For readability, only excerpts from the Maven *pom.xml* file are shown here, the whole file is available from the Neo4j Procedure Template *(https://github.com/neo4j-examples/neo4j-procedure-template)* repository.

Setting up a project with Maven

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                    http://maven.apache.org/xsd/maven-4.0.0.xsd">
 <modelVersion>4.0.0</modelVersion>

 <groupId>org.neo4j.example</groupId>
 <artifactId>procedure-template</artifactId>
 <version>1.0.0-SNAPSHOT</version>

 <packaging>jar</packaging>
 <name>Neo4j Procedure Template</name>
 <description>A template project for building a Neo4j Procedure</description>

 <properties>
   <neo4j.version>3.4.0</neo4j.version>
 </properties>
```

Next, the build dependencies are defined. The following two sections are included in the *pom.xml* between `<dependencies></dependencies>` tags.

The first dependency section includes the procedure API that procedures use at runtime. The scope is set to `provided`, because once the procedure is deployed to a Neo4j instance, this dependency is provided by Neo4j. If non-Neo4j dependencies are added to the project, their scope should normally be `compile`.

```
   <dependency>
     <groupId>org.neo4j</groupId>
     <artifactId>neo4j</artifactId>
     <version>${neo4j.version}</version>
     <scope>provided</scope>
   </dependency>
```

Next, the dependencies necessary for testing the procedure are added:

- Neo4j Harness, a utility that allows for starting a lightweight Neo4j instance. It is used to start Neo4j with a specific procedure deployed, which greatly simplifies testing.
- The Neo4j Java driver, used to send cypher statements that call the procedure.
- JUnit, a common Java test framework.

```
<dependency>
  <groupId>org.neo4j.test</groupId>
  <artifactId>neo4j-harness</artifactId>
  <version>${neo4j.version}</version>
  <scope>test</scope>
</dependency>

<dependency>
  <groupId>org.neo4j.driver</groupId>
  <artifactId>neo4j-java-driver</artifactId>
  <version>{java-driver-version}</version>
  <scope>test</scope>
</dependency>

<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.12</version>
  <scope>test</scope>
</dependency>
```

Along with declaring the dependencies used by the procedure it is also necessary to define the steps that Maven will go through to build the project. The goal is first to *compile* the source, then to *package* it in a *jar* that can be deployed to a Neo4j instance.

Procedures require at least Java 8, so the version `1.8` should be defined as the *source* and *target version* in the configuration for the Maven compiler plugin.

The Maven Shade *(https://maven.apache.org/plugins/maven-shade-plugin/)* plugin is used to package the compiled procedure. It also includes all dependencies in the package, unless the dependency scope is set to *test* or *provided*.

Once the procedure has been deployed to the *plugins* directory of each Neo4j instance and the instances have restarted, the procedure is available for use.

```
<build>
 <plugins>
   <plugin>
     <artifactId>maven-compiler-plugin</artifactId>
     <version>3.1</version>
     <configuration>
       <source>1.8</source>
       <target>1.8</target>
     </configuration>
   </plugin>
   <plugin>
     <artifactId>maven-shade-plugin</artifactId>
     <executions>
       <execution>
         <phase>package</phase>
         <goals>
           <goal>shade</goal>
         </goals>
       </execution>
     </executions>
   </plugin>
 </plugins>
</build>
```

Writing integration tests

The test dependencies include *Neo4j Harness* and *JUnit*. These can be used to write integration tests for procedures.

First, we decide what the procedure should do, then we write a test that proves that it does it right. Finally we write a procedure that passes the test.

Below is a template for testing a procedure that accesses Neo4j's full-text indexes from Cypher.

Writing tests for procedures

```
package example;

import org.junit.Rule;
import org.junit.Test;
import org.neo4j.driver.v1.*;
import org.neo4j.graphdb.factory.GraphDatabaseSettings;
import org.neo4j.harness.junit.Neo4jRule;

import static org.hamcrest.core.IsEqual.equalTo;
import static org.junit.Assert.assertThat;
import static org.neo4j.driver.v1.Values.parameters;

public class ManualFullTextIndexTest
{
    // This rule starts a Neo4j instance for us
    @Rule
    public Neo4jRule neo4j = new Neo4jRule()

            // This is the Procedure we want to test
            .withProcedure( FullTextIndex.class );

    @Test
    public void shouldAllowIndexingAndFindingANode() throws Throwable
    {
        // In a try-block, to make sure we close the driver after the test
        try( Driver driver = GraphDatabase.driver( neo4j.boltURI() , Config.build().withEncryptionLevel(
Config.EncryptionLevel.NONE ).toConfig() ) )
        {

            // Given I've started Neo4j with the FullTextIndex procedure class
            //       which my 'neo4j' rule above does.
            Session session = driver.session();

            // And given I have a node in the database
            long nodeId = session.run( "CREATE (p:User {name:'Brookreson'}) RETURN id(p)" )
                    .single()
                    .get( 0 ).asLong();

            // When I use the index procedure to index a node
            session.run( "CALL example.index({id}, ['name'])", parameters( "id", nodeId ) );

            // Then I can search for that node with lucene query syntax
            StatementResult result = session.run( "CALL example.search('User', 'name:Brook*')" );
            assertThat( result.single().get( "nodeId" ).asLong(), equalTo( nodeId ) );
        }
    }
}
```

Writing a procedure

With the test in place, we write a procedure procedure that fulfils the expectations of the test. The full example is available in the Neo4j Procedure Template *(https://github.com/neo4j-examples/neo4j-procedure-template)* repository.

Particular things to note:

- All procedures are annotated `@Procedure`. Procedures that write to the database specify `mode = WRITE`.
- The *context* of the procedure, which is the same as each resource that the procedure wants to use, is annotated `@Context`.
- The *input* and *output* to and from a procedure must be one of the supported types. In Java this corresponds to `String`, `Long`, `Double`, `Boolean`, `org.neo4j.graphdb.Node`, `org.neo4j.graphdb.Relationship`, `org.neo4j.graphdb.Path`, and `org.neo4j.graphdb.spatial.Point`. Composite types are also supported via `List<T>` where `T` is one the supported types and `Map<String, Object>` where the values in the map must have one of the supported types. For the

case where the type is not known beforehand we also support `Object`, however note that the actual value must still have one of the aforementioned types.

For more details, see the API documentation for procedures *(https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/procedure/Procedure.html)*.

The correct way to signal an error from within a procedure is to throw a `RuntimeException`.

```
package example;

import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.stream.Stream;

import org.neo4j.graphdb.GraphDatabaseService;
import org.neo4j.graphdb.Label;
import org.neo4j.graphdb.Node;
import org.neo4j.graphdb.index.Index;
import org.neo4j.graphdb.index.IndexManager;
import org.neo4j.logging.Log;
import org.neo4j.procedure.Context;
import org.neo4j.procedure.Name;
import org.neo4j.procedure.PerformsWrites;
import org.neo4j.procedure.Procedure;

import static org.neo4j.helpers.collection.MapUtil.stringMap;
import static org.neo4j.procedure.Procedure.Mode.SCHEMA;
import static org.neo4j.procedure.Procedure.Mode.WRITE;

/**
 * This is an example showing how you could expose Neo4j's full text indexes as
 * two procedures - one for updating indexes, and one for querying by label and
 * the lucene query language.
 */
public class FullTextIndex
{
    // Only static fields and @Context-annotated fields are allowed in
    // Procedure classes. This static field is the configuration we use
    // to create full-text indexes.
    private static final Map<String,String> FULL_TEXT =
            stringMap( IndexManager.PROVIDER, "lucene", "type", "fulltext" );

    // This field declares that we need a GraphDatabaseService
    // as context when any procedure in this class is invoked
    @Context
    public GraphDatabaseService db;

    // This gives us a log instance that outputs messages to the
    // standard log, `neo4j.log`
    @Context
    public Log log;

    /**
     * This declares the first of two procedures in this class - a
     * procedure that performs queries in a manual index.
     *
     * It returns a Stream of Records, where records are
     * specified per procedure. This particular procedure returns
     * a stream of {@link SearchHit} records.
     *
     * The arguments to this procedure are annotated with the
     * {@link Name} annotation and define the position, name
     * and type of arguments required to invoke this procedure.
     * There is a limited set of types you can use for arguments,
     * these are as follows:
     *
     * <ul>
     *     <li>{@link String}</li>
     *     <li>{@link Long} or {@code long}</li>
     *     <li>{@link Double} or {@code double}</li>
     *     <li>{@link Number}</li>
     *     <li>{@link Boolean} or {@code boolean}</li>
     *     <li>{@link org.neo4j.graphdb.Node}</li>
```

```
     *     <li>{@link org.neo4j.graphdb.Relationship}</li>
     *     <li>{@link org.neo4j.graphdb.Path}</li>
     *     <li>{@link java.util.Map} with key {@link String} and value of any type in this list, including
{@link java.util.Map}</li>
     *     <li>{@link java.util.List} of elements of any valid field type, including {@link
java.util.List}</li>
     *     <li>{@link Object}, meaning any of the types above</li>
     *
     * @param label the label name to query by
     * @param query the lucene query, for instance `name:Brook*` to
     *              search by property `name` and find any value starting
     *              with `Brook`. Please refer to the Lucene Query Parser
     *              documentation for full available syntax.
     * @return the nodes found by the query
     */
    @Procedure( name = "example.search", mode = WRITE )
    public Stream<SearchHit> search( @Name("label") String label,
                                     @Name("query") String query )
    {
        String index = indexName( label );

        // Avoid creating the index, if it's not there we won't be
        // finding anything anyway!
        if( !db.index().existsForNodes( index ))
        {
            // Just to show how you'd do logging
            log.debug( "Skipping index query since index does not exist: `%s`", index );
            return Stream.empty();
        }

        // If there is an index, do a lookup and convert the result
        // to our output record.
        return db.index()
                .forNodes( index )
                .query( query )
                .stream()
                .map( SearchHit::new );
    }

    /**
     * This is the second procedure defined in this class, it is used to update the
     * index with nodes that should be queryable. You can send the same node multiple
     * times, if it already exists in the index the index will be updated to match
     * the current state of the node.
     *
     * This procedure works largely the same as {@link #search(String, String)},
     * with three notable differences. One, it is annotated with `mode = SCHEMA`,
     * which is <i>required</i> if you want to perform updates to the graph in your
     * procedure.
     *
     * Two, it returns {@code void} rather than a stream. This is simply a short-hand
     * for saying our procedure always returns an empty stream of empty records.
     *
     * Three, it uses a default value for the property list, in this way you can call
     * the procedure by simply invoking {@code CALL index(nodeId)}. Default values are
     * are provided as the Cypher string representation of the given type, e.g.
     * {@code {default: true}}, {@code null}, or {@code -1}.
     *
     * @param nodeId the id of the node to index
     * @param propKeys a list of property keys to index, only the ones the node
     *                 actually contains will be added
     */
    @Procedure( name = "example.index", mode = SCHEMA )
    public void index( @Name("nodeId") long nodeId,
                       @Name(value = "properties", defaultValue = "[]") List<String> propKeys )
    {
        Node node = db.getNodeById( nodeId );

        // Load all properties for the node once and in bulk,
        // the resulting set will only contain those properties in `propKeys`
        // that the node actually contains.
        Set<Map.Entry<String,Object>> properties =
                node.getProperties( propKeys.toArray( new String[0] ) ).entrySet();

        // Index every label (this is just as an example, we could filter which labels to index)
        for ( Label label : node.getLabels() )
        {
            Index<Node> index = db.index().forNodes( indexName( label.name() ), FULL_TEXT );

            // In case the node is indexed before, remove all occurrences of it so
```

```
            // we don't get old or duplicated data
            index.remove( node );

            // And then index all the properties
            for ( Map.Entry<String,Object> property : properties )
            {
                index.add( node, property.getKey(), property.getValue() );
            }
        }
    }


    /**
     * This is the output record for our search procedure. All procedures
     * that return results return them as a Stream of Records, where the
     * records are defined like this one - customized to fit what the procedure
     * is returning.
     *
     * The fields must be one of the following types:
     *
     * <ul>
     *     <li>{@link String}</li>
     *     <li>{@link Long} or {@code long}</li>
     *     <li>{@link Double} or {@code double}</li>
     *     <li>{@link Number}</li>
     *     <li>{@link Boolean} or {@code boolean}</li>
     *     <li>{@link org.neo4j.graphdb.Node}</li>
     *     <li>{@link org.neo4j.graphdb.Relationship}</li>
     *     <li>{@link org.neo4j.graphdb.Path}</li>
     *     <li>{@link java.util.Map} with key {@link String} and value {@link Object}</li>
     *     <li>{@link java.util.List} of elements of any valid field type, including {@link
java.util.List}</li>
     *     <li>{@link Object}, meaning any of the valid field types</li>
     * </ul>
     */
    public static class SearchHit
    {
        // This records contain a single field named 'nodeId'
        public long nodeId;

        public SearchHit( Node node )
        {
            this.nodeId = node.getId();
        }
    }

    private String indexName( String label )
    {
        return "label-" + label;
    }
}
```

1.2. Unmanaged server extensions

Sometimes you'll want finer grained control over your application's interactions with Neo4j than cypher provides. For these situations you can use the unmanaged extension API.

This is a sharp tool, allowing users to deploy arbitrary JAX-RS *(https://en.wikipedia.org/wiki/JAX-RS)* classes to the server so be careful when using this. In particular it's easy to consume lots of heap space on the server and degrade performance. If in doubt, please ask for help via one of the community channels.

1.2.1. Introduction to unmanaged extensions

The first step when writing an unmanaged extension is to create a project which includes dependencies to the JAX-RS and Neo4j core jars. In Maven this would be achieved by adding the following lines to the pom file:

```
<dependency>
    <groupId>javax.ws.rs</groupId>
    <artifactId>javax.ws.rs-api</artifactId>
    <version>2.0</version>
    <scope>provided</scope>
</dependency>
```

```
<dependency>
    <groupId>org.neo4j</groupId>
    <artifactId>neo4j</artifactId>
    <version>3.4.0</version>
    <scope>provided</scope>
</dependency>
```

Now we're ready to write our extension.

In our code we'll interact with the database using `GraphDatabaseService` which we can get access to by using the `@Context` annotation. The following example serves as a template which you can base your extension on:

Unmanaged extension example

```
@Path( "/helloworld" )
public class HelloWorldResource
{
    private final GraphDatabaseService database;

    public HelloWorldResource( @Context GraphDatabaseService database )
    {
        this.database = database;
    }

    @GET
    @Produces( MediaType.TEXT_PLAIN )
    @Path( "/{nodeId}" )
    public Response hello( @PathParam( "nodeId" ) long nodeId )
    {
        // Do stuff with the database
        return Response.status( Status.OK ).entity( UTF8.encode( "Hello World, nodeId=" + nodeId ) ).
build();
    }
}
```

The full source code is found here: HelloWorldResource.java *(https://github.com/neo4j/neo4j-documentation/blob/3.4/server-examples/src/main/java/org/neo4j/examples/server/unmanaged/HelloWorldResource.java)*

Having built your code, the resulting jar file (and any custom dependencies) should be placed in the *$NEO4J_SERVER_HOME/plugins* directory. We also need to tell Neo4j where to look for the extension by adding some configuration in *neo4j.conf*:

```
#Comma separated list of JAXRS packages containing JAXRS Resource, one package name for each mountpoint.
dbms.unmanaged_extension_classes=org.neo4j.examples.server.unmanaged=/examples/unmanaged
```

Our hello method will now respond to `GET` requests at the URI: `http://{neo4j_server}:{neo4j_port}/examples/unmanaged/helloworld/{nodeId}`. e.g.

```
curl http://localhost:7474/examples/unmanaged/helloworld/123
```

which results in

```
Hello World, nodeId=123
```

1.2.2. Streaming JSON responses

When writing unmanaged extensions we have greater control over the amount of memory that our Neo4j queries use. If we keep too much state around it can lead to more frequent full Garbage Collection and subsequent unresponsiveness by the Neo4j server.

A common way that state can creep in is the creation of JSON objects to represent the result of a query which we then send back to our application. Neo4j's Transactional Cypher HTTP endpoint (see Developer manual HTTP API transactional endpoint) streams responses back to the client and we should follow in its footsteps.

For example, the following unmanaged extension streams an array of a person's colleagues:

Unmanaged extension streaming example

```
@Path("/colleagues")
public class ColleaguesResource
{
    private GraphDatabaseService graphDb;
    private final ObjectMapper objectMapper;

    private static final RelationshipType ACTED_IN = RelationshipType.withName( "ACTED_IN" );
    private static final Label PERSON = Label.label( "Person" );

    public ColleaguesResource( @Context GraphDatabaseService graphDb )
    {
        this.graphDb = graphDb;
        this.objectMapper = new ObjectMapper();
    }

    @GET
    @Path("/{personName}")
    public Response findColleagues( @PathParam("personName") final String personName )
    {
        StreamingOutput stream = new StreamingOutput()
        {
            @Override
            public void write( OutputStream os ) throws IOException, WebApplicationException
            {
                JsonGenerator jg = objectMapper.getJsonFactory().createJsonGenerator( os, JsonEncoding
.UTF8 );
                jg.writeStartObject();
                jg.writeFieldName( "colleagues" );
                jg.writeStartArray();

                try ( Transaction tx = graphDb.beginTx();
                      ResourceIterator<Node> persons = graphDb.findNodes( PERSON, "name", personName ) )
                {
                    while ( persons.hasNext() )
                    {
                        Node person = persons.next();
                        for ( Relationship actedIn : person.getRelationships( ACTED_IN, OUTGOING ) )
                        {
                            Node endNode = actedIn.getEndNode();
                            for ( Relationship colleagueActedIn : endNode.getRelationships( ACTED_IN,
INCOMING ) )
                            {
                                Node colleague = colleagueActedIn.getStartNode();
                                if ( !colleague.equals( person ) )
                                {
                                    jg.writeString( colleague.getProperty( "name" ).toString() );
                                }
                            }
                        }
                    }
                    tx.success();
                }

                jg.writeEndArray();
                jg.writeEndObject();
                jg.flush();
                jg.close();
            }
        };

        return Response.ok().entity( stream ).type( MediaType.APPLICATION_JSON ).build();
    }
}
```

The full source code is found here: ColleaguesResource.java *(https://github.com/neo4j/neo4j-documentation/blob/3.4/server-examples/src/main/java/org/neo4j/examples/server/unmanaged/ColleaguesResource.java)*

As well as depending on JAX-RS API this example also uses Jackson — a Java JSON library. You'll need to add the following dependency to your Maven POM file (or equivalent):

```
<dependency>
    <groupId>org.codehaus.jackson</groupId>
    <artifactId>jackson-mapper-asl</artifactId>
    <version>1.9.7</version>
</dependency>
```

Our `findColleagues` method will now respond to `GET` requests at the URI: `http://{neo4j_server}:{neo4j_port}/examples/unmanaged/colleagues/{personName}`. For example:

```
curl http://localhost:7474/examples/unmanaged/colleagues/Keanu%20Reeves
```

which results in

```
{"colleagues":["Hugo Weaving","Carrie-Anne Moss","Laurence Fishburne"]}
```

1.2.3. Using Cypher in an unmanaged extension

You can execute Cypher queries by using the `GraphDatabaseService` that is injected into the extension. For example, the following unmanaged extension retrieves a person's colleagues using Cypher:

Unmanaged extension Cypher execution example

```
@Path("/colleagues-cypher-execution")
public class ColleaguesCypherExecutionResource
{
    private final ObjectMapper objectMapper;
    private GraphDatabaseService graphDb;

    public ColleaguesCypherExecutionResource( @Context GraphDatabaseService graphDb )
    {
        this.graphDb = graphDb;
        this.objectMapper = new ObjectMapper();
    }

    @GET
    @Path("/{personName}")
    public Response findColleagues( @PathParam("personName") final String personName )
    {
        final Map<String, Object> params = MapUtil.map( "personName", personName );

        StreamingOutput stream = new StreamingOutput()
        {
            @Override
            public void write( OutputStream os ) throws IOException, WebApplicationException
            {
                JsonGenerator jg = objectMapper.getJsonFactory().createJsonGenerator( os, JsonEncoding
.UTF8 );
                jg.writeStartObject();
                jg.writeFieldName( "colleagues" );
                jg.writeStartArray();

                try ( Transaction tx = graphDb.beginTx();
                      Result result = graphDb.execute( colleaguesQuery(), params ) )
                {
                    while ( result.hasNext() )
                    {
                        Map<String,Object> row = result.next();
                        jg.writeString( ((Node) row.get( "colleague" )).getProperty( "name" ).toString()
);
                    }
                    tx.success();
                }

                jg.writeEndArray();
                jg.writeEndObject();
                jg.flush();
                jg.close();
            }
        };

        return Response.ok().entity( stream ).type( MediaType.APPLICATION_JSON ).build();
    }

    private String colleaguesQuery()
    {
        return "MATCH (p:Person {name: $personName })-[:ACTED_IN]->()<-[:ACTED_IN]-(colleague) RETURN
colleague";
    }
}
```

The full source code is found here: ColleaguesCypherExecutionResource.java *(https://github.com/neo4j/neo4j-documentation/blob/3.4/server-examples/src/main/java/org/neo4j/examples/server/unmanaged/ColleaguesCypherExecutionResource.java)*

Our `findColleagues` method will now respond to `GET` requests at the URI: `http://{neo4j_server}:{neo4j_port}/examples/unmanaged/colleagues-cypher-execution/{personName}`. e.g.

```
curl http://localhost:7474/examples/unmanaged/colleagues-cypher-execution/Keanu%20Reeves
```

which results in

```
{"colleagues":["Hugo Weaving","Carrie-Anne Moss","Laurence Fishburne"]}
```

1.2.4. Testing your extension

Neo4j provides tools to help you write integration tests for your extensions. You can access this toolkit by adding the following test dependency to your project:

```
<dependency>
   <groupId>org.neo4j.test</groupId>
   <artifactId>neo4j-harness</artifactId>
   <version>3.4.0</version>
   <scope>test</scope>
</dependency>
```

The test toolkit provides a mechanism to start a Neo4j instance with custom configuration and with extensions of your choice. It also provides mechanisms to specify data fixtures to include when starting Neo4j.

Usage example

```
@Path("")
public static class MyUnmanagedExtension
{
    @GET
    public Response myEndpoint()
    {
        return Response.ok().build();
    }
}

@Test
public void testMyExtension() throws Exception
{
    // Given
    try ( ServerControls server = getServerBuilder()
            .withExtension( "/myExtension", MyUnmanagedExtension.class )
            .newServer() )
    {
        // When
        HTTP.Response response = HTTP.GET(
                HTTP.GET( server.httpURI().resolve( "myExtension" ).toString() ).location() );

        // Then
        assertEquals( 200, response.status() );
    }
}

@Test
public void testMyExtensionWithFunctionFixture() throws Exception
{
    // Given
    try ( ServerControls server = getServerBuilder()
            .withExtension( "/myExtension", MyUnmanagedExtension.class )
            .withFixture( new Function<GraphDatabaseService, Void>()
            {
                @Override
                public Void apply( GraphDatabaseService graphDatabaseService ) throws RuntimeException
                {
                    try ( Transaction tx = graphDatabaseService.beginTx() )
                    {
                        graphDatabaseService.createNode( Label.label( "User" ) );
                        tx.success();
                    }
                    return null;
                }
            } )
            .newServer() )
    {
        // When
        Result result = server.graph().execute( "MATCH (n:User) return n" );

        // Then
        assertEquals( 1, count( result ) );
    }
}
```

The full source code of the example is found here: ExtensionTestingDocIT.java *(https://github.com/neo4j/neo4j-documentation/blob/3.4/neo4j-harness-test/src/test/java/org/neo4j/harness/doc/ExtensionTestingDocIT.java)*

Note the use of `server.httpURI().resolve( "myExtension" )` to ensure that the correct base URI is used.

If you are using the JUnit test framework, there is a JUnit rule available as well.

JUnit example

```
@Rule
public Neo4jRule neo4j = new Neo4jRule()
        .withFixture( "CREATE (admin:Admin)" )
        .withConfig( LegacySslPolicyConfig.certificates_directory.name(),
                getRelativePath( getSharedTestTemporaryFolder(), LegacySslPolicyConfig
.certificates_directory ) )
        .withConfig( ServerSettings.script_enabled.name(), Settings.TRUE )
        .withFixture( new Function<GraphDatabaseService, Void>()
        {
            @Override
            public Void apply( GraphDatabaseService graphDatabaseService ) throws RuntimeException
            {
                try (Transaction tx = graphDatabaseService.beginTx())
                {
                    graphDatabaseService.createNode( Label.label( "Admin" ) );
                    tx.success();
                }
                return null;
            }
        } );

@Test
public void shouldWorkWithServer() throws Exception
{
    // Given
    URI serverURI = neo4j.httpURI();

    // When I access the server
    HTTP.Response response = HTTP.GET( serverURI.toString() );

    // Then it should reply
    assertEquals(200, response.status());

    // and we have access to underlying GraphDatabaseService
    try (Transaction tx = neo4j.getGraphDatabaseService().beginTx()) {
        assertEquals( 2, count(neo4j.getGraphDatabaseService().findNodes( Label.label( "Admin" ) ) ));
        tx.success();
    }
}
```

1.3. Installing Procedures and Extensions in Neo4j Desktop

Extensions can be be deployed also when using Neo4j Desktop. Neo4j Desktop will add all jars in *%ProgramFiles%\Neo4j Community\plugins* to the classpath, but please note that nested directories for plugins are not supported.

Otherwise extensions are subject to the same rules as usual. Please note when configuring server extensions that *neo4j.conf* for Neo4j Desktop lives in *%APPDATA%\Neo4j Community*.

Chapter 2. REST API authorization rules

This section discusses authorization rules for restricting access to Neo4j through the REST API.

Administrators may require more fine-grained security policies in addition to the basic authorization and/or IP-level restrictions on the Web server. Neo4j supports administrators in allowing or disallowing access the specific aspects of the database based on credentials that users or applications provide.

To facilitate domain-specific authorization policies in Neo4j when using the REST API, security rules can be implemented and registered with the server. This makes scenarios such as user and role-based security and authentication against external lookup services possible. See `org.neo4j.server.rest.security.SecurityRule` in the javadocs downloadable from Maven Central (org.neo4j.app:neo4j-server) *(http://search.maven.org/#search%7Cgav%7C1%7Cg%3A%22org.neo4j.app%22%20AND%20a%3A%22neo4j-server%22)*.

The use of REST API Authorization Rules may interact unexpectedly with the built-in authentication and authorization (see the relevant sections in the Neo4j Operations Manual), if enabled.

2.1. Enforcing server authorization rules

In this example, a (dummy) failing security rule is registered to deny access to all URIs to the server by listing the rules class in *neo4j.conf*:

```
dbms.security.http_authorization_classes=my.rules.PermanentlyFailingSecurityRule
```

with the rule source code of:

```
public class PermanentlyFailingSecurityRule implements SecurityRule
{

    public static final String REALM = "WallyWorld"; // as per RFC2617 :-)

    @Override
    public boolean isAuthorized( HttpServletRequest request )
    {
        return false; // always fails - a production implementation performs
                      // deployment-specific authorization logic here
    }

    @Override
    public String forUriPath()
    {
        return "/*";
    }

    @Override
    public String wwwAuthenticateHeader()
    {
        return SecurityFilter.basicAuthenticationResponse(REALM);
    }
}
```

With this rule registered, any access to the server will be denied. In a production-quality implementation the rule will likely lookup credentials/claims in a 3rd-party directory service (e.g. LDAP) or in a local database of authorized users.

Example request

- **POST** http://localhost:7474/db/data/node
- **Accept:** application/json; charset=UTF-8

Example response

- **401:** Unauthorized
- **WWW-Authenticate:** Basic realm="WallyWorld"

2.2. Using wildcards to target security rules

In this example, a security rule is registered to deny access to all URIs to the server by listing the rule(s) class(es) in *neo4j.conf*. In this case, the rule is registered using a wildcard URI path (where `characters can be used to signify any part of the path). For example /users` means the rule will be bound to any resources under the `/users` root path. Similarly `/users*type*` will bind the rule to resources matching URIs like `/users/fred/type/premium`.

```
dbms.security.http_authorization_classes=my.rules.PermanentlyFailingSecurityRuleWithWildcardPath
```

with the rule source code of:

```
public String forUriPath()
{
    return "/protected/*";
}
```

With this rule registered, any access to URIs under /protected/ will be denied by the server. Using wildcards allows flexible targeting of security rules to arbitrary parts of the server's API, including any unmanaged extensions or managed plugins that have been registered.

Example request

- **GET** http://localhost:7474/protected/tree/starts/here/dummy/more/stuff
- **Accept:** application/json

Example response

- **401:** Unauthorized
- **WWW-Authenticate:** Basic realm="WallyWorld"

2.3. Using complex wildcards to target security rules

In this example, a security rule is registered to deny access to all URIs matching a complex pattern. The config looks like this:

```
dbms.security.http_authorization_classes=my.rules.PermanentlyFailingSecurityRuleWithComplexWildcardPath
```

with the rule source code of:

```
public class PermanentlyFailingSecurityRuleWithComplexWildcardPath implements SecurityRule
{

    public static final String REALM = "WallyWorld"; // as per RFC2617 :-)

    @Override
    public boolean isAuthorized( HttpServletRequest request )
    {
        return false;
    }

    @Override
    public String forUriPath()
    {
        return "/protected/*/something/else/*/final/bit";
    }

    @Override
    public String wwwAuthenticateHeader()
    {
        return SecurityFilter.basicAuthenticationResponse(REALM);
    }
}
```

Example request

- **GET** http://localhost:7474/protected/wildcard_replacement/x/y/z/something/else/more_wildcard_replacement/a/b/c/final/bit/more/stuff
- **Accept:** application/json

Example response

- **401:** Unauthorized
- **WWW-Authenticate:** Basic realm="WallyWorld"

Chapter 3. Setup for remote debugging

In order to configure the Neo4j server for remote debugging sessions, the Java debugging parameters need to be passed to the Java process through the configuration. They live in the *conf/neo4j.conf* file.

In order to specify the parameters, add a line for the additional Java arguments like this:

```
dbms.jvm.additional=-agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=5005
```

This configuration will start a Neo4j server ready for remote debugging attachement at localhost and port `5005`. Use these parameters to attach to the process from Eclipse, IntelliJ or your remote debugger of choice after starting the server.

Chapter 4. Using Neo4j embedded in Java applications

It's easy to use Neo4j embedded in Java applications. In this chapter you will find everything needed — from setting up the environment to doing something useful with your data.

When running your own code and Neo4j in the same JVM, there are a few things you should keep in mind:

- Don't create or retain more objects than you strictly need to. Large caches in particular tend to promote more objects to the old generation, thus increasing the need for expensive full garbage collections.
- Don't use internal Neo4j APIs. They are internal to Neo4j and subject to change without notice, which may break or change the behavior of your code.
- Don't enable the `-XX:+TrustFinalNonStaticFields` JVM flag when running in embedded mode.

4.1. Include Neo4j in your project

After selecting the appropriate edition for your platform, embed Neo4j in your Java application by including the Neo4j library jars in your build. The following sections will show how to do this by either altering the build path directly or by using dependency management.

4.1.1. Add Neo4j to the build path

Get the Neo4j libraries from one of these sources:

- Extract a Neo4j zip/tarball *(https://neo4j.com/download/other-releases/#releases)*, and use the *jar* files found in the *lib/* directory.
- Use the *jar* files available from Maven Central Repository *(http://search.maven.org/#search|ga|1|g%3A%22org.neo4j%22)*

Add the jar files to your project:

JDK tools

Append to `-classpath`

Eclipse

- Right-click on the project and then go *Build Path* → *Configure Build Path.* In the dialog, choose *Add External JARs*, browse to the Neo4j *lib/* directory and select all of the jar files.
- Another option is to use User Libraries *(http://help.eclipse.org/indigo/index.jsp?topic=/org.eclipse.jdt.doc.user/reference/preferences/java/buildpath/ref-preferences-user-libraries.htm)*.

IntelliJ IDEA

See Libraries, Global Libraries, and the Configure Library dialog *(http://www.jetbrains.com/help/idea/2016.1/configuring-project-and-global-libraries.html)*

NetBeans

- Right-click on the *Libraries* node of the project, choose *Add JAR/Folder*, browse to the Neo4j *lib/* directory and select all of the jar files.
- You can also handle libraries from the project node, see Managing a Project's Classpath *(http://netbeans.org/kb/docs/java/project-setup.html#projects-classpath)*.

4.1.2. Editions

The following table outlines the available editions and their names for use with dependency management tools.

Follow the links in the table for details on dependency configuration with Apache Maven, Apache Buildr, Apache Ivy, Groovy Grape, Grails, Scala SBT!

Table 1. Neo4j editions

Edition	Dependency	Description
Community	org.neo4j:neo4j *(http://search.maven.org/#search%7Cgav%7C1%7Cg%3A%22org.neo4j%22%20AND%20a%3A%22neo4j%22)*	a high performance, fully ACID transactional graph database
Enterprise	org.neo4j:neo4j-enterprise *(http://search.maven.org/#search%7Cgav%7C1%7Cg%3A%22org.neo4j%22%20AND%20a%3A%22neo4j-enterprise%22)*	adding advanced monitoring, online backup and High Availability clustering

The listed dependencies do not contain the implementation, but pulls it in transitively.

For information regarding licensing, see the Licensing Guide *(https://neo4j.com/licensing)*.

Javadocs can be downloaded packaged in jar files from Maven Central or read at javadocs *(https://neo4j.com/docs/java-reference/3.4/javadocs/)*.

4.1.3. Add Neo4j as a dependency

You can either go with the top-level artifact from the table above or include the individual components directly. The examples included here use the top-level artifact approach.

Maven

Add the dependency to your project along the lines of the snippet below. This is usually done in the *pom.xml* file found in the root directory of the project.

Maven dependency

```
<project>
...
 <dependencies>
  <dependency>
   <groupId>org.neo4j</groupId>
   <artifactId>neo4j</artifactId>
   <version>3.4.0</version>
  </dependency>
  ...
 </dependencies>
...
</project>
```

Where the `artifactId` *is found in the editions table.*

Eclipse and Maven

For development in Eclipse *(http://www.eclipse.org)*, it is recommended to install the m2e plugin *(http://www.eclipse.org/m2e/)* and let Maven manage the project build classpath instead, see above. This also adds the possibility to build your project both via the command line with Maven and have a

working Eclipse setup for development.

Ivy

Make sure to resolve dependencies from Maven Central, for example using this configuration in your *ivysettings.xml* file:

```
<ivysettings>
  <settings defaultResolver="main"/>
  <resolvers>
    <chain name="main">
      <filesystem name="local">
        <artifact pattern="${ivy.settings.dir}/repository/[artifact]-[revision].[ext]" />
      </filesystem>
      <ibiblio name="maven_central" root="http://repo1.maven.org/maven2/" m2compatible="true"/>
    </chain>
  </resolvers>
</ivysettings>
```

With that in place you can add Neo4j to the mix by having something along these lines to your 'ivy.xml' file:

```
..
<dependencies>
  ..
  <dependency org="org.neo4j" name="neo4j" rev="3.4.0"/>
  ..
</dependencies>
..
```

Where the `name` *is found in the editions table above*

Gradle

The example below shows an example gradle build script for including the Neo4j libraries.

```
def neo4jVersion = "3.4.0"
apply plugin: 'java'
repositories {
   mavenCentral()
}
dependencies {
   compile "org.neo4j:neo4j:${neo4jVersion}"
}
```

Where the coordinates (`org.neo4j:neo4j` *in the example) are found in the editions table above.*

4.1.4. Starting and stopping

To create a new database or open an existing one you instantiate a `GraphDatabaseService` (https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/graphdb/GraphDatabaseService.html).

```
graphDb = new GraphDatabaseFactory().newEmbeddedDatabase( databaseDirectory );
registerShutdownHook( graphDb );
```

The `GraphDatabaseService` instance can be shared among multiple threads. Note however that you can't create multiple instances pointing to the same database.

To stop the database, call the `shutdown()` method:

```
graphDb.shutdown();
```

To make sure Neo4j is shut down properly you can add a shutdown hook:

```
private static void registerShutdownHook( final GraphDatabaseService graphDb )
{
    // Registers a shutdown hook for the Neo4j instance so that it
    // shuts down nicely when the VM exits (even if you "Ctrl-C" the
    // running application).
    Runtime.getRuntime().addShutdownHook( new Thread()
    {
        @Override
        public void run()
        {
            graphDb.shutdown();
        }
    } );
}
```

Starting an embedded database with configuration settings

To start Neo4j with configuration settings, a Neo4j properties file can be loaded like this:

```
GraphDatabaseService graphDb = new GraphDatabaseFactory()
    .newEmbeddedDatabaseBuilder( testDirectory.graphDbDir() )
    .loadPropertiesFromFile( pathToConfig + "neo4j.conf" )
    .newGraphDatabase();
```

Configuration settings can also be applied programmatically, like so:

```
GraphDatabaseService graphDb = new GraphDatabaseFactory()
    .newEmbeddedDatabaseBuilder( testDirectory.graphDbDir() )
    .setConfig( GraphDatabaseSettings.pagecache_memory, "512M" )
    .setConfig( GraphDatabaseSettings.string_block_size, "60" )
    .setConfig( GraphDatabaseSettings.array_block_size, "300" )
    .newGraphDatabase();
```

Starting an embedded read-only instance

If you want a *read-only view* of the database, create an instance this way:

```
graphDb = new GraphDatabaseFactory().newEmbeddedDatabaseBuilder( dir )
        .setConfig( GraphDatabaseSettings.read_only, "true" )
        .newGraphDatabase();
```

Obviously the database has to already exist in this case.

Concurrent access to the same database files by multiple (read-only or write) instances is not supported.

4.2. Hello world

Learn how to create and access nodes and relationships. For information on project setup, see Include Neo4j in your project.

Remember that a Neo4j graph consists of:

- Nodes that are connected by

- Relationships, with
- Properties on both nodes and relationships.

All relationships have a type. For example, if the graph represents a social network, a relationship type could be `KNOWS`. If a relationship of the type `KNOWS` connects two nodes, that probably represents two people that know each other. A lot of the semantics (that is the meaning) of a graph is encoded in the relationship types of the application. And although relationships are directed they are equally well traversed regardless of which direction they are traversed.

The source code of this example is found here: EmbeddedNeo4j.java *(https://github.com/neo4j/neo4j-documentation/blob/3.4/embedded-examples/src/main/java/org/neo4j/examples/EmbeddedNeo4j.java)*

4.2.1. Prepare the database

Relationship types can be created by using an `enum`. In this example we only need a single relationship type. This is how to define it:

```
private static enum RelTypes implements RelationshipType
{
    KNOWS
}
```

We also prepare some variables to use:

```
GraphDatabaseService graphDb;
Node firstNode;
Node secondNode;
Relationship relationship;
```

The next step is to start the database server. Note that if the directory given for the database doesn't already exist, it will be created.

```
graphDb = new GraphDatabaseFactory().newEmbeddedDatabase( databaseDirectory );
registerShutdownHook( graphDb );
```

Note that starting a database server is an expensive operation, so don't start up a new instance every time you need to interact with the database! The instance can be shared by multiple threads. Transactions are thread confined.

As seen, we register a shutdown hook that will make sure the database shuts down when the JVM exits. Now it's time to interact with the database.

4.2.2. Wrap operations in a transaction

All operations have to be performed in a transaction. This is a conscious design decision, since we believe transaction demarcation to be an important part of working with a real enterprise database. Now, transaction handling in Neo4j is very easy:

```
try ( Transaction tx = graphDb.beginTx() )
{
    // Database operations go here
    tx.success();
}
```

For more information on transactions, see Transaction management and Java API for Transaction

(https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/graphdb/Transaction.html).

For brevity, we do not spell out wrapping of operations in a transaction throughout the manual.

4.2.3. Create a small graph

Now, let's create a few nodes. The API is very intuitive. Feel free to have a look at the Neo4j Javadocs *(https://neo4j.com/docs/java-reference/3.4/javadocs/)*. They're included in the distribution, as well. Here's how to create a small graph consisting of two nodes, connected with one relationship and some properties:

```
firstNode = graphDb.createNode();
firstNode.setProperty( "message", "Hello, " );
secondNode = graphDb.createNode();
secondNode.setProperty( "message", "World!" );

relationship = firstNode.createRelationshipTo( secondNode, RelTypes.KNOWS );
relationship.setProperty( "message", "brave Neo4j " );
```

We now have a graph that looks like this:

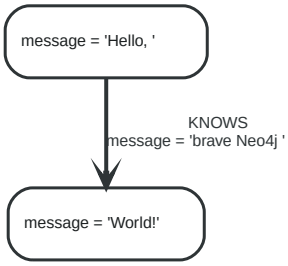


Figure 1. Hello World Graph

4.2.4. Print the result

After we've created our graph, let's read from it and print the result.

```
System.out.print( firstNode.getProperty( "message" ) );
System.out.print( relationship.getProperty( "message" ) );
System.out.print( secondNode.getProperty( "message" ) );
```

Which will output:

Hello, brave Neo4j World!

4.2.5. Remove the data

In this case we'll remove the data before committing:

```
// let's remove the data
firstNode.getSingleRelationship( RelTypes.KNOWS, Direction.OUTGOING ).delete();
firstNode.delete();
secondNode.delete();
```

Note that deleting a node which still has relationships when the transaction commits will fail. This is to make sure relationships always have a start node and an end node.

4.2.6. Shut down the database server

Finally, shut down the database server *when the application finishes:*

```
graphDb.shutdown();
```

4.3. Property values

Both nodes and relationships can have properties.

Properties are named values where the name is a string. Property values can be either a primitive or an array of one primitive type. For example `String`, `int` and `int[]` values are valid for properties.

`NULL` *is not a valid property value.*

Setting a property to `NULL` is equivalent to deleting the property.

Table 2. Property value types

Type	Description
`boolean`	
`byte`	8-bit integer
`short`	16-bit integer
`int`	32-bit integer
`long`	64-bit integer
`float`	32-bit IEEE 754 floating-point number
`double`	64-bit IEEE 754 floating-point number
`char`	16-bit unsigned integers representing Unicode characters
`String`	sequence of Unicode characters
`org.neo4j.graphdb.spatial.Point`	A 2D or 3D point object in a given coordinate system.
`java.time.LocalDate`	An instant capturing the date, but not the time, nor the timezone.
`java.time.OffsetTime`	An instant capturing the time of day, and the timezone offset, but not the date.
`java.time.LocalTime`	An instant capturing the time of day, but not the date, nor the timezone.
`java.time.ZonedDateTime`	An instant capturing the date, the time, and the timezone.
`java.time.LocalDateTime`	An instant capturing the date and the time, but not the timezone.
`java.time.temporal.TemporalAmount`	A temporal amount. This captures the difference in time between two instants.

For further details on float/double values, see Java Language Specification *(http://docs.oracle.com/javase/specs/jls/se8/html/jls-4.html#jls-4.2.3)*.

4.4. User database with indexes

You have a user database, and want to retrieve users by name using indexes.

The source code used in this example is found here: EmbeddedNeo4jWithNewIndexing.java *(https://github.com/neo4j/neo4j-documentation/blob/3.4/embedded-examples/src/main/java/org/neo4j/examples/EmbeddedNeo4jWithNewIndexing.java)*

To begin with, we start the database server:

```
GraphDatabaseService graphDb = new GraphDatabaseFactory().newEmbeddedDatabase( databaseDirectory );
```

Then we have to configure the database to index users by name. This only needs to be done once.

Schema changes and data changes are not allowed in the same transaction. Each transaction must either change the schema or the data, but not both.

```
IndexDefinition indexDefinition;
try ( Transaction tx = graphDb.beginTx() )
{
    Schema schema = graphDb.schema();
    indexDefinition = schema.indexFor( Label.label( "User" ) )
            .on( "username" )
            .create();
    tx.success();
}
```

Indexes are populated asynchronously when they are first created. It is possible to use the core API to wait for index population to complete:

```
try ( Transaction tx = graphDb.beginTx() )
{
    Schema schema = graphDb.schema();
    schema.awaitIndexOnline( indexDefinition, 10, TimeUnit.SECONDS );
}
```

It is also possible to query the progress of the index population:

```
try ( Transaction tx = graphDb.beginTx() )
{
    Schema schema = graphDb.schema();
    System.out.println( String.format( "Percent complete: %1.0f%%",
            schema.getIndexPopulationProgress( indexDefinition ).getCompletedPercentage() ) );
}
```

It's time to add the users:

```
try ( Transaction tx = graphDb.beginTx() )
{
    Label label = Label.label( "User" );

    // Create some users
    for ( int id = 0; id < 100; id++ )
    {
        Node userNode = graphDb.createNode( label );
        userNode.setProperty( "username", "user" + id + "@neo4j.org" );
    }
    System.out.println( "Users created" );
    tx.success();
}
```

Please read Managing resources when using long running transactions on how to properly close `ResourceIterators` returned from index lookups.

And here's how to find a user by id:

```
Label label = Label.label( "User" );
int idToFind = 45;
String nameToFind = "user" + idToFind + "@neo4j.org";
try ( Transaction tx = graphDb.beginTx() )
{
    try ( ResourceIterator<Node> users =
                graphDb.findNodes( label, "username", nameToFind ) )
    {
        ArrayList<Node> userNodes = new ArrayList<>();
        while ( users.hasNext() )
        {
            userNodes.add( users.next() );
        }

        for ( Node node : userNodes )
        {
            System.out.println(
                    "The username of user " + idToFind + " is " + node.getProperty( "username" ) );
        }
    }
}
```

When updating the name of a user, the index is updated as well:

```
try ( Transaction tx = graphDb.beginTx() )
{
    Label label = Label.label( "User" );
    int idToFind = 45;
    String nameToFind = "user" + idToFind + "@neo4j.org";

    for ( Node node : loop( graphDb.findNodes( label, "username", nameToFind ) ) )
    {
        node.setProperty( "username", "user" + (idToFind + 1) + "@neo4j.org" );
    }
    tx.success();
}
```

When deleting a user, it is automatically removed from the index:

```
try ( Transaction tx = graphDb.beginTx() )
{
    Label label = Label.label( "User" );
    int idToFind = 46;
    String nameToFind = "user" + idToFind + "@neo4j.org";

    for ( Node node : loop( graphDb.findNodes( label, "username", nameToFind ) ) )
    {
        node.delete();
    }
    tx.success();
}
```

In case we change our data model, we can drop the index as well:

```
try ( Transaction tx = graphDb.beginTx() )
{
    Label label = Label.label( "User" );
    for ( IndexDefinition indexDefinition : graphDb.schema()
            .getIndexes( label ) )
    {
        // There is only one index
        indexDefinition.drop();
    }

    tx.success();
}
```

4.5. User database with explicit index

Unless you have specific reasons to use the explicit indexing, see User database with indexes instead.

Please read Managing resources when using long running transactions on how to properly close `ResourceIterators` returned from index lookups.

You have a user database, and want to retrieve users by name using the explicit indexing system.

The source code used in this example is found here: EmbeddedNeo4jWithIndexing.java *(https://github.com/neo4j/neo4j-documentation/blob/3.4/embedded-examples/src/main/java/org/neo4j/examples/EmbeddedNeo4jWithIndexing.java)*

We have created two helper methods to handle user names and adding users to the database:

```
private static String idToUserName( final int id )
{
    return "user" + id + "@neo4j.org";
}

private static Node createAndIndexUser( final String username )
{
    Node node = graphDb.createNode();
    node.setProperty( USERNAME_KEY, username );
    nodeIndex.add( node, USERNAME_KEY, username );
    return node;
}
```

The next step is to start the database server:

```
graphDb = new GraphDatabaseFactory().newEmbeddedDatabase( databaseDirectory );
registerShutdownHook();
```

It's time to add the users:

```
try ( Transaction tx = graphDb.beginTx() )
{
    nodeIndex = graphDb.index().forNodes( "nodes" );
    // Create some users and index their names with the IndexService
    for ( int id = 0; id < 100; id++ )
    {
        createAndIndexUser( idToUserName( id ) );
    }
```

And here's how to find a user by Id:

```
int idToFind = 45;
String userName = idToUserName( idToFind );
Node foundUser = nodeIndex.get( USERNAME_KEY, userName ).getSingle();

System.out.println( "The username of user " + idToFind + " is "
    + foundUser.getProperty( USERNAME_KEY ) );
```

4.6. Managing resources when using long running transactions

It is necessary to always open a transaction when accessing the database. Inside a long running transaction it is good practice to ensure that any `ResourceIterator` (https://neo4j.com/docs/java-

reference/3.4/javadocs/org/neo4j/graphdb/ResourceIterator.html)s obtained inside the transaction are closed as early as possible. This is either achieved by just exhausting the iterator or by explicitly calling its close method.

What follows is an example of how to work with a `ResourceIterator`. As we don't exhaust the iterator, we will close it explicitly using the `close()` method.

```
Label label = Label.label( "User" );
int idToFind = 45;
String nameToFind = "user" + idToFind + "@neo4j.org";
try ( Transaction tx = graphDb.beginTx();
      ResourceIterator<Node> users = graphDb.findNodes( label, "username", nameToFind ) )
{
    Node firstUserNode;
    if ( users.hasNext() )
    {
        firstUserNode = users.next();
    }
    users.close();
}
```

4.7. Controlling logging

To control logging in Neo4j embedded, use the Neo4j embedded logging framework.

Neo4j embedded provides logging via its own `org.neo4j.logging.Log (https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/logging/Log.html)` layer, and does not natively use any existing Java logging framework. All logging events produced by Neo4j have a name, a level and a message. The name is a FQCN (fully qualified class name).

Neo4j uses the following log levels:

`ERROR`	For serious errors that are almost always fatal
`WARN`	For events that are serious, but not fatal
`INFO`	Informational events
`DEBUG`	Debugging events

To enable logging, an implementation of `org.neo4j.logging.LogProvider (https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/logging/LogProvider.html)` must be provided to the `GraphDatabaseFactory (https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/graphdb/factory/GraphDatabaseFactory.html)`, as follows:

```
LogProvider logProvider = new MyCustomLogProvider( output );
graphDb = new GraphDatabaseFactory().setUserLogProvider( logProvider ).newEmbeddedDatabase(
databaseDirectory );
```

Neo4j also includes a binding for SLF4J, which is available in the `neo4j-slf4j` library jar. This can be obtained via Maven:

```
<project>
...
 <dependencies>
  <dependency>
   <groupId>org.neo4j</groupId>
   <artifactId>neo4j-slf4j</artifactId>
   <version>3.4.0</version>
  </dependency>
  <dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
    <version>${slf4j-version}</version>
  </dependency>
  ...
 </dependencies>
...
</project>
```

To use this binding, simply pass an instance of `org.neo4j.logging.slf4j.Slf4jLogProvider (https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/logging/slf4j/Slf4jLogProvider.html)` to the `GraphDatabaseFactory (https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/graphdb/factory/GraphDatabaseFactory.html)`, as follows:

```
graphDb = new GraphDatabaseFactory().setUserLogProvider( new Slf4jLogProvider() )
        .newEmbeddedDatabase( databaseDirectory );
```

All log output can then be controlled via SLF4J configuration.

4.8. Basic unit testing

The basic pattern of unit testing with Neo4j is illustrated by the following example.

To access the Neo4j testing facilities you should have the `neo4j-kernel` 'tests.jar' together with the `neo4j-io` 'tests.jar' on the classpath during tests. You can download them from Maven Central: org.neo4j:neo4j-kernel *(http://search.maven.org/#search|ga|1|g%3A%22org.neo4j%22%20AND%20a%3A%22neo4j-kernel%22)* and org.neo4j:neo4j-io *(http://search.maven.org/#search|ga|1|g%3A%22org.neo4j%22%20AND%20a%3A%22neo4j-io%22)*.

Using Maven as a dependency manager you would typically add this dependency together with JUnit and Hamcrest like so:

Maven dependency

```
<project>
...
 <dependencies>
  <dependency>
   <groupId>org.neo4j</groupId>
   <artifactId>neo4j-kernel</artifactId>
   <version>3.4.0</version>
   <type>test-jar</type>
   <scope>test</scope>
  </dependency>
  <dependency>
   <groupId>org.neo4j</groupId>
   <artifactId>neo4j-io</artifactId>
   <version>3.4.0</version>
   <type>test-jar</type>
   <scope>test</scope>
  </dependency>
  <dependency>
   <groupId>junit</groupId>
   <artifactId>junit</artifactId>
   <version>4.12</version>
   <scope>test</scope>
  </dependency>
  <dependency>
   <groupId>org.hamcrest</groupId>
   <artifactId>hamcrest-all</artifactId>
   <version>1.3</version>
   <scope>test</scope>
  </dependency>
  ...
 </dependencies>
...
</project>
```

Observe that the `<type>test-jar</type>` is crucial. Without it you would get the common `neo4j-kernel` jar, not the one containing the testing facilities.

With that in place, we're ready to code our tests.

For the full source code of this example see: Neo4jBasicDocTest.java *(https://github.com/neo4j/neo4j-documentation/blob/3.4/embedded-examples/src/test/java/org/neo4j/examples/Neo4jBasicDocTest.java)*

Before each test, create a fresh database:

```
@Before
public void prepareTestDatabase()
{
    graphDb = new TestGraphDatabaseFactory().newImpermanentDatabase( testDirectory.directory() );
}
```

After the test has executed, the database should be shut down:

```
@After
public void destroyTestDatabase()
{
    graphDb.shutdown();
}
```

During a test, create nodes and check to see that they are there, while enclosing write operations in a transaction.

```
Node n = null;
try ( Transaction tx = graphDb.beginTx() )
{
    n = graphDb.createNode();
    n.setProperty( "name", "Nancy" );
    tx.success();
}

// The node should have a valid id
assertThat( n.getId(), is( greaterThan( -1L ) ) );

// Retrieve a node by using the id of the created node. The id's and
// property should match.
try ( Transaction tx = graphDb.beginTx() )
{
    Node foundNode = graphDb.getNodeById( n.getId() );
    assertThat( foundNode.getId(), is( n.getId() ) );
    assertThat( (String) foundNode.getProperty( "name" ), is( "Nancy" ) );
}
```

If you want to set configuration parameters at database creation, it's done like this:

```
GraphDatabaseService db = new TestGraphDatabaseFactory()
    .newImpermanentDatabaseBuilder()
    .setConfig( GraphDatabaseSettings.pagecache_memory, "512M" )
    .setConfig( GraphDatabaseSettings.string_block_size, "60" )
    .setConfig( GraphDatabaseSettings.array_block_size, "300" )
    .newGraphDatabase();
```

4.9. Traversal

For reading about traversals, see The traversal framework.

4.9.1. The Matrix

This is the first graph we want to traverse into:

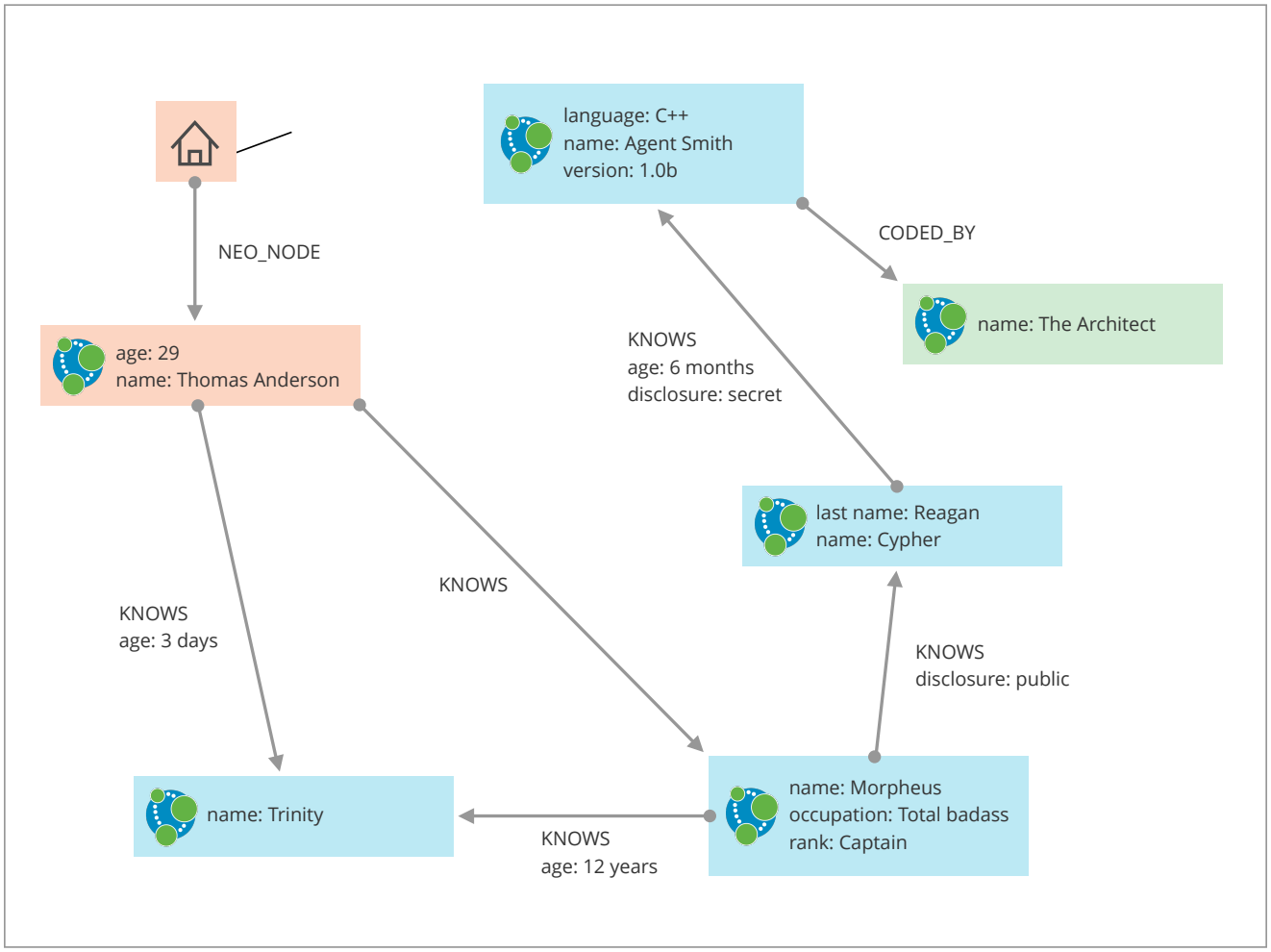


Figure 2. Matrix node space view

The source code of this example is found here: NewMatrix.java *(https://github.com/neo4j/neo4j-documentation/blob/3.4/embedded-examples/src/main/java/org/neo4j/examples/NewMatrix.java)*

Friends and friends of friends

```
private Traverser getFriends(
        final Node person )
{
    TraversalDescription td = graphDb.traversalDescription()
            .breadthFirst()
            .relationships( RelTypes.KNOWS, Direction.OUTGOING )
            .evaluator( Evaluators.excludeStartPosition() );
    return td.traverse( person );
}
```

Let's perform the actual traversal and print the results:

```
int numberOfFriends = 0;
String output = neoNode.getProperty( "name" ) + "'s friends:\n";
Traverser friendsTraverser = getFriends( neoNode );
for ( Path friendPath : friendsTraverser )
{
    output += "At depth " + friendPath.length() + " => "
              + friendPath.endNode()
                      .getProperty( "name" ) + "\n";
    numberOfFriends++;
}
output += "Number of friends found: " + numberOfFriends + "\n";
```

Which will give us the following output:

```
Thomas Anderson's friends:
At depth 1 => Morpheus
At depth 1 => Trinity
At depth 2 => Cypher
At depth 3 => Agent Smith
Number of friends found: 4
```

Who coded the Matrix?

```
private Traverser findHackers( final Node startNode )
{
    TraversalDescription td = graphDb.traversalDescription()
            .breadthFirst()
            .relationships( RelTypes.CODED_BY, Direction.OUTGOING )
            .relationships( RelTypes.KNOWS, Direction.OUTGOING )
            .evaluator(
                    Evaluators.includeWhereLastRelationshipTypeIs( RelTypes.CODED_BY ) );
    return td.traverse( startNode );
}
```

Print out the result:

```
String output = "Hackers:\n";
int numberOfHackers = 0;
Traverser traverser = findHackers( getNeoNode() );
for ( Path hackerPath : traverser )
{
    output += "At depth " + hackerPath.length() + " => "
              + hackerPath.endNode()
                      .getProperty( "name" ) + "\n";
    numberOfHackers++;
}
output += "Number of hackers found: " + numberOfHackers + "\n";
```

Now we know who coded the Matrix:

```
Hackers:
At depth 4 => The Architect
Number of hackers found: 1
```

Walking an ordered path

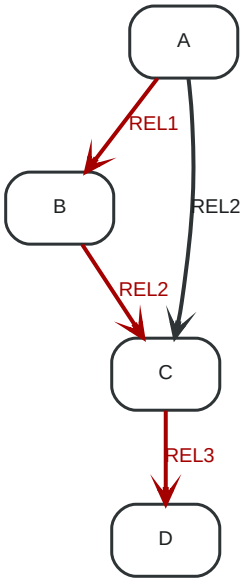
This example shows how to use a path context holding a representation of a path.

The source code of this example is found here: OrderedPath.java *(https://github.com/neo4j/neo4j-documentation/blob/3.4/embedded-examples/src/main/java/org/neo4j/examples/orderedpath/OrderedPath.java)*

Create a toy graph

```
Node A = db.createNode();
Node B = db.createNode();
Node C = db.createNode();
Node D = db.createNode();

A.createRelationshipTo( C, REL2 );
C.createRelationshipTo( D, REL3 );
A.createRelationshipTo( B, REL1 );
B.createRelationshipTo( C, REL2 );
```



Now, the order of relationships (`REL1` → `REL2` → `REL3`) is stored in an `ArrayList`. Upon traversal, the `Evaluator` can check against it to ensure that only paths are included and returned that have the predefined order of relationships:

Define how to walk the path

```
final ArrayList<RelationshipType> orderedPathContext = new ArrayList<RelationshipType>();
orderedPathContext.add( REL1 );
orderedPathContext.add( withName( "REL2" ) );
orderedPathContext.add( withName( "REL3" ) );
TraversalDescription td = db.traversalDescription()
        .evaluator( new Evaluator()
        {
            @Override
            public Evaluation evaluate( final Path path )
            {
                if ( path.length() == 0 )
                {
                    return Evaluation.EXCLUDE_AND_CONTINUE;
                }
                RelationshipType expectedType = orderedPathContext.get( path.length() - 1 );
                boolean isExpectedType = path.lastRelationship()
                        .isType( expectedType );
                boolean included = path.length() == orderedPathContext.size() && isExpectedType;
                boolean continued = path.length() < orderedPathContext.size() && isExpectedType;
                return Evaluation.of( included, continued );
            }
        } )
        .uniqueness( Uniqueness.NODE_PATH );
```

Note that we set the uniqueness to `Uniqueness.NODE_PATH` (https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/graphdb/traversal/Uniqueness.html#NODE_PATH) as we want to be able to revisit the same node during the traversal, but not the same path.

Perform the traversal and print the result

```
Traverser traverser = td.traverse( A );
PathPrinter pathPrinter = new PathPrinter( "name" );
for ( Path path : traverser )
{
    output += Paths.pathToString( path, pathPrinter );
}
```

Which will output:

```
(A)--[REL1]-->(B)--[REL2]-->(C)--[REL3]-->(D)
```

In this case we use a custom class to format the path output. This is how it's done:

```
static class PathPrinter implements Paths.PathDescriptor<Path>
{
    private final String nodePropertyKey;

    public PathPrinter( String nodePropertyKey )
    {
        this.nodePropertyKey = nodePropertyKey;
    }

    @Override
    public String nodeRepresentation( Path path, Node node )
    {
        return "(" + node.getProperty( nodePropertyKey, "" ) + ")";
    }

    @Override
    public String relationshipRepresentation( Path path, Node from, Relationship relationship )
    {
        String prefix = "--", suffix = "--";
        if ( from.equals( relationship.getEndNode() ) )
        {
            prefix = "<--";
        }
        else
        {
            suffix = "-->";
        }
        return prefix + "[" + relationship.getType().name() + "]" + suffix;
    }
}
```

4.9.2. Uniqueness of Paths in traversals

This example is demonstrating the use of node uniqueness. Below an imaginary domain graph with Principals that own pets that are descendant to other pets.

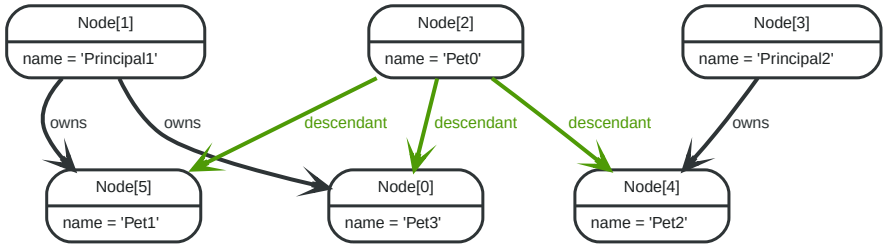


Figure 3. Descendants example graph

In order to return all descendants of `Pet0` which have the relation `owns` to `Principal1` (`Pet1` and `Pet3`), the Uniqueness of the traversal needs to be set to `NODE_PATH` rather than the default `NODE_GLOBAL`. This way nodes can be traversed more that once, and paths that have different nodes but can have some nodes in common (like the start and end node) can be returned.

```
final Node target = data.get().get( "Principal1" );
TraversalDescription td = db.traversalDescription()
        .uniqueness( Uniqueness.NODE_PATH )
        .evaluator( new Evaluator()
{
    @Override
    public Evaluation evaluate( Path path )
    {
        boolean endNodeIsTarget = path.endNode().equals( target );
        return Evaluation.of( endNodeIsTarget, !endNodeIsTarget );
    }
} );

Traverser results = td.traverse( start );
```

This will return the following paths:

```
(2)-[descendant,2]->(0)<-[owns,5]-(1)
(2)-[descendant,0]->(5)<-[owns,3]-(1)
```

In the default `path.toString()` implementation, `(1)--[knows,2]->(4)` denotes a node with ID=1 having a relationship with ID=2 or type `knows` to a node with ID=4.

Let's create a new `TraversalDescription` from the old one, having `NODE_GLOBAL` uniqueness to see the difference.

The `TraversalDescription` object is immutable, so we have to use the new instance returned with the new uniqueness setting.

```
TraversalDescription nodeGlobalTd = td.uniqueness( Uniqueness.NODE_GLOBAL );
results = nodeGlobalTd.traverse( start );
```

Now only one path is returned:

```
(2)-[descendant,2]->(0)<-[owns,5]-(1)
```

4.9.3. Social network

The following example uses the new enhanced traversal API.

Social networks (know as social graphs out on the web) are natural to model with a graph. This example shows a very simple social model that connects friends and keeps track of status updates.

The source code of the example is found here: socnet *(https://github.com/neo4j/neo4j-documentation/tree/3.4/embedded-examples/src/main/java/org/neo4j/examples/socnet)*

Simple social model

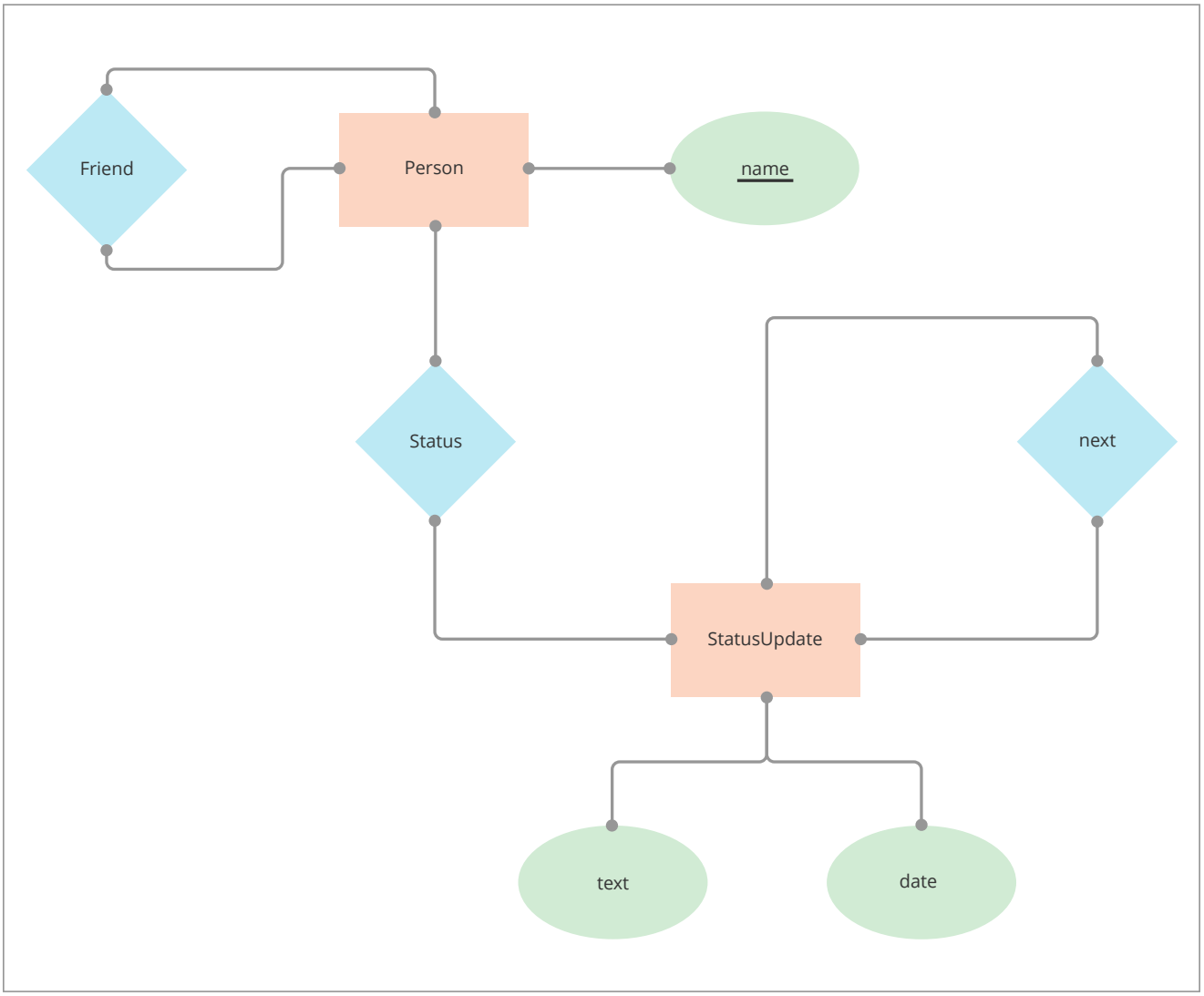


Figure 4. Social network data model

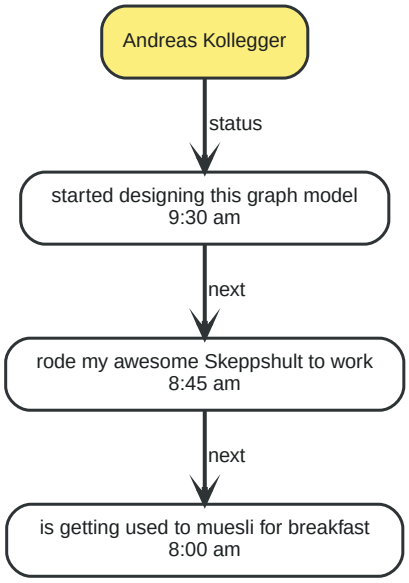
The data model for a social network is pretty simple: `Persons` with names and `StatusUpdates` with timestamped text. These entities are then connected by specific relationships.

- `Person`
  - `friend`: relates two distinct `Person` instances (no self-reference)
  - `status`: connects to the most recent `StatusUpdate`
- `StatusUpdate`
  - `next`: points to the next `StatusUpdate` in the chain, which was posted before the current one

Status graph instance

The `StatusUpdate` list for a `Person` is a linked list. The head of the list (the most recent status) is found by following `status`. Each subsequent `StatusUpdate` is connected by `next`.

Here's an example where Andreas Kollegger micro-blogged his way to work in the morning:



To read the status updates, we can create a traversal, like so:

```
TraversalDescription traversal = graphDb().traversalDescription()
        .depthFirst()
        .relationships( NEXT );
```

This gives us a traverser that will start at one `StatusUpdate`, and will follow the chain of updates until they run out. Traversers are lazy loading, so it's performant even when dealing with thousands of statuses — they are not loaded until we actually consume them.

Activity stream

Once we have friends, and they have status messages, we might want to read our friends status' messages, in reverse time order — latest first. To do this, we go through these steps:

1. Gather all friend's status update iterators in a list — latest date first.
2. Sort the list.
3. Return the first item in the list.
4. If the first iterator is exhausted, remove it from the list. Otherwise, get the next item in that iterator.
5. Go to step 2 until there are no iterators left in the list.

Animated, the sequence looks like this *(http://www.slideshare.net/systay/pattern-activity-stream)*.

The code looks like:

```
PositionedIterator<StatusUpdate> first = statuses.get(0);
StatusUpdate returnVal = first.current();

if ( !first.hasNext() )
{
    statuses.remove( 0 );
}
else
{
    first.next();
    sort();
}

return returnVal;
```

4.10. Domain entities

This page demonstrates one way to handle domain entities when using Neo4j. The principle at use is to wrap the entities around a node (the same approach can be used with relationships as well).

The source code of the examples is found here: Person.java *(https://github.com/neo4j/neo4j-documentation/blob/3.4/embedded-examples/src/main/java/org/neo4j/examples/socnet/Person.java)*

First off, store the node and make it accessible inside the package:

```
private final Node underlyingNode;

Person( Node personNode )
{
    this.underlyingNode = personNode;
}

protected Node getUnderlyingNode()
{
    return underlyingNode;
}
```

Delegate attributes to the node:

```
public String getName()
{
    return (String)underlyingNode.getProperty( NAME );
}
```

Make sure to override these methods:

```
@Override
public int hashCode()
{
    return underlyingNode.hashCode();
}

@Override
public boolean equals( Object o )
{
    return o instanceof Person &&
            underlyingNode.equals( ( (Person)o ).getUnderlyingNode() );
}

@Override
public String toString()
{
    return "Person[" + getName() + "]";
}
```

4.11. Graph algorithm examples

For details on the graph algorithm usage, see the Javadocs *(https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/graphalgo/GraphAlgoFactory.html)*.

The source code used in the example is found here: PathFindingDocTest.java *(https://github.com/neo4j/neo4j-documentation/blob/3.4/embedded-examples/src/test/java/org/neo4j/examples/PathFindingDocTest.java)*

Calculating the shortest path (least number of relationships) between two nodes:

```
Node startNode = graphDb.createNode();
Node middleNode1 = graphDb.createNode();
Node middleNode2 = graphDb.createNode();
Node middleNode3 = graphDb.createNode();
Node endNode = graphDb.createNode();
createRelationshipsBetween( startNode, middleNode1, endNode );
createRelationshipsBetween( startNode, middleNode2, middleNode3, endNode );

// Will find the shortest path between startNode and endNode via
// "MY_TYPE" relationships (in OUTGOING direction), like f.ex:
//
// (startNode)-->(middleNode1)-->(endNode)
//
PathFinder<Path> finder = GraphAlgoFactory.shortestPath(
    PathExpanders.forTypeAndDirection( ExampleTypes.MY_TYPE, Direction.OUTGOING ), 15 );
Iterable<Path> paths = finder.findAllPaths( startNode, endNode );
```

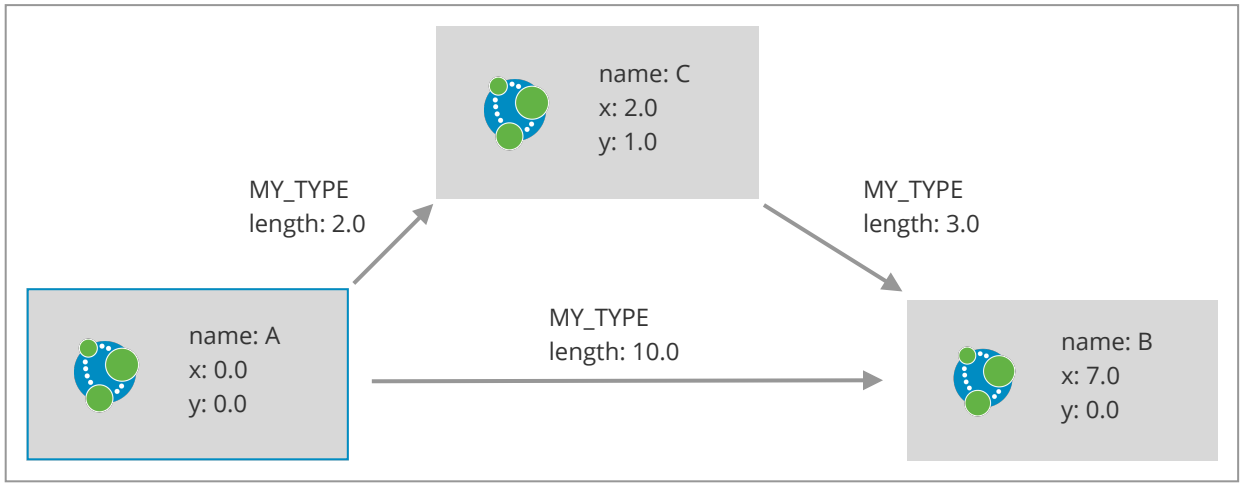
Using Dijkstra's algorithm *(https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm)* to calculate cheapest path between node A and B where each relationship can have a weight (i.e. cost) and the path(s) with least cost are found.

```
PathFinder<WeightedPath> finder = GraphAlgoFactory.dijkstra(
    PathExpanders.forTypeAndDirection( ExampleTypes.MY_TYPE, Direction.BOTH ), "cost" );

WeightedPath path = finder.findSinglePath( nodeA, nodeB );

// Get the weight for the found path
path.weight();
```

Using A* *(https://en.wikipedia.org/wiki/A*_search_algorithm)* to calculate the cheapest path between node A and B, where cheapest is for example the path in a network of roads which has the shortest length between node A and B. Here's our example graph:



```
Node nodeA = createNode( "name", "A", "x", 0d, "y", 0d );
Node nodeB = createNode( "name", "B", "x", 7d, "y", 0d );
Node nodeC = createNode( "name", "C", "x", 2d, "y", 1d );
Relationship relAB = createRelationship( nodeA, nodeC, "length", 2d );
Relationship relBC = createRelationship( nodeC, nodeB, "length", 3d );
Relationship relAC = createRelationship( nodeA, nodeB, "length", 10d );

EstimateEvaluator<Double> estimateEvaluator = new EstimateEvaluator<Double>()
{
    @Override
    public Double getCost( final Node node, final Node goal )
    {
        double dx = (Double) node.getProperty( "x" ) - (Double) goal.getProperty( "x" );
        double dy = (Double) node.getProperty( "y" ) - (Double) goal.getProperty( "y" );
        double result = Math.sqrt( Math.pow( dx, 2 ) + Math.pow( dy, 2 ) );
        return result;
    }
};
PathFinder<WeightedPath> astar = GraphAlgoFactory.aStar(
        PathExpanders.allTypesAndDirections(),
        CommonEvaluators.doubleCostEvaluator( "length" ), estimateEvaluator );
WeightedPath path = astar.findSinglePath( nodeA, nodeB );
```

4.12. Reading a management attribute

The JmxUtils *(https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/jmx/JmxUtils.html)* class includes methods to access Neo4j management beans. The common JMX service can be used as well, but from your code you probably rather want to use the approach outlined here.

The source code of the example is found here: JmxDocTest.java *(https://github.com/neo4j/neo4j-documentation/blob/3.4/embedded-examples/src/test/java/org/neo4j/examples/JmxDocTest.java)*

This example shows how to get the start time of a database:

```
private static Date getStartTimeFromManagementBean(
        GraphDatabaseService graphDbService )
{
    ObjectName objectName = JmxUtils.getObjectName( graphDbService, "Kernel" );
    Date date = JmxUtils.getAttribute( objectName, "KernelStartTime" );
    return date;
}
```

Depending on which Neo4j edition you are using different sets of management beans are available.

- For all editions, see the org.neo4j.jmx *(https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/jmx/package-summary.html)* package.
- For the Enterprise Edition, see the org.neo4j.management *(https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/management/package-summary.html)* package as well.

4.13. How to create unique nodes

This section is about how to ensure uniqueness of a property when creating nodes. For an overview of the topic, see Creating unique nodes.

4.13.1. Get or create unique node using Cypher and unique constraints

Create a unique constraint

```
try ( Transaction tx = graphdb.beginTx() )
{
    graphdb.schema()
            .constraintFor( Label.label( "User" ) )
            .assertPropertyIsUnique( "name" )
            .create();
    tx.success();
}
```

Use MERGE to create a unique node

```
Node result = null;
ResourceIterator<Node> resultIterator = null;
try ( Transaction tx = graphDb.beginTx() )
{
    String queryString = "MERGE (n:User {name: $name}) RETURN n";
    Map<String, Object> parameters = new HashMap<>();
    parameters.put( "name", username );
    resultIterator = graphDb.execute( queryString, parameters ).columnAs( "n" );
    result = resultIterator.next();
    tx.success();
    return result;
}
```

4.13.2. Get or create unique node using an explicit index

While this is a working solution, please consider using the preferred solution at Get or create unique node using Cypher and unique constraints instead.

By using put-if-absent *(https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/graphdb/index/Index.html#putIfAbsent-T-java.lang.String-java.lang.Object-)* functionality, entity uniqueness can be guaranteed using an index.

Here the index acts as the lock and will only lock the smallest part needed to guarantee uniqueness across threads and transactions. To get the more high-level `get-or-create` functionality make use of UniqueFactory *(https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/graphdb/index/UniqueFactory.html)* as seen in the example below.

Create a factory for unique nodes at application start

```
try ( Transaction tx = graphDb.beginTx() )
{
    UniqueFactory.UniqueNodeFactory result = new UniqueFactory.UniqueNodeFactory( graphDb, "users" )
    {
        @Override
        protected void initialize( Node created, Map<String, Object> properties )
        {
            created.addLabel( Label.label( "User" ) );
            created.setProperty( "name", properties.get( "name" ) );
        }
    };
    tx.success();
    return result;
}
```

Use the unique node factory to get or create a node

```
try ( Transaction tx = graphDb.beginTx() )
{
    Node node = factory.getOrCreate( "name", username );
    tx.success();
    return node;
}
```

4.13.3. Pessimistic locking for node creation

While this is a working solution, please consider using the preferred solution at Get or create unique node using Cypher and unique constraints instead.

One might be tempted to use Java synchronization for pessimistic locking, but this is dangerous. By mixing locks in Neo4j and in the Java runtime, it is easy to produce deadlocks that are not detectable by Neo4j. As long as all locking is done by Neo4j, all deadlocks will be detected and avoided. Also, a solution using manual synchronization doesn't ensure uniqueness in an HA environment.

This example uses a single "lock node" for locking. We create it only as a place to put locks, nothing else.

Create a lock node at application start

```
try ( Transaction tx = graphDb.beginTx() )
{
    final Node lockNode = graphDb.createNode();
    tx.success();
    return lockNode;
}
```

Use the lock node to ensure nodes are not created concurrently

```
try ( Transaction tx = graphDb.beginTx() )
{
    Index<Node> usersIndex = graphDb.index().forNodes( "users" );
    Node userNode = usersIndex.get( "name", username ).getSingle();
    if ( userNode != null )
    {
        return userNode;
    }

    tx.acquireWriteLock( lockNode );
    userNode = usersIndex.get( "name", username ).getSingle();
    if ( userNode == null )
    {
        userNode = graphDb.createNode( Label.label( "User" ) );
        usersIndex.add( userNode, "name", username );
        userNode.setProperty( "name", username );
    }
    tx.success();
    return userNode;
}
```

Note that finishing the transaction will release the lock on the lock node.

4.14. Accessing Neo4j embedded via the Bolt protocol

Open a Bolt connector to your embedded instance to get GUI administration and other benefits.

The Neo4j Browser and the official Neo4j Drivers use the Bolt database protocol to communicate with Neo4j. By default, Neo4j Embedded does not expose a Bolt connector, but you can enable one. Doing so allows you to connect the services Neo4j Browser to your embedded instance.

It also gives you a way to incrementally transfer an existing Embedded application to use Neo4j Drivers instead. Migrating to Neo4j Drivers means you become free to choose to run Neo4j Embedded or Neo4j Server, without changing your application code.

To add a Bolt Connector to your embedded database, you need to add the Bolt extension to your class path. This is done by adding an additional dependency to your project.

```
<project>
...
 <dependencies>

  <dependency>
    <groupId>org.neo4j</groupId>
    <artifactId>neo4j-bolt</artifactId>
    <version>3.4.0</version>
  </dependency>
  ...
 </dependencies>
...
</project>
```

With this dependency in place, you can configure Neo4j to enable a Bolt connector.

```
GraphDatabaseSettings.BoltConnector bolt = GraphDatabaseSettings.boltConnector( "0" );

GraphDatabaseService graphDb = new GraphDatabaseFactory()
        .newEmbeddedDatabaseBuilder( DB_PATH )
        .setConfig( bolt.type, "BOLT" )
        .setConfig( bolt.enabled, "true" )
        .setConfig( bolt.address, "localhost:7687" )
        .newGraphDatabase();
```

4.15. Terminating a running transaction

Sometimes you may want to terminate (abort) a long running transaction from another thread.

The source code used in this example is found here: TerminateTransactions.java *(https://github.com/neo4j/neo4j-documentation/blob/3.4/embedded-examples/src/main/java/org/neo4j/examples/TerminateTransactions.java)*

To begin with, we start the database server:

```
GraphDatabaseService graphDb = new GraphDatabaseFactory().newEmbeddedDatabase( databaseDirectory );
```

Now we start creating an infinite binary tree of nodes in the database, as an example of a long running transaction.

```
RelationshipType relType = RelationshipType.withName( "CHILD" );
Queue<Node> nodes = new LinkedList<>();
int depth = 1;

try ( Transaction tx = graphDb.beginTx() )
{
    Node rootNode = graphDb.createNode();
    nodes.add( rootNode );

    for (; true; depth++) {
        int nodesToExpand = nodes.size();
        for (int i = 0; i < nodesToExpand; ++i) {
            Node parent = nodes.remove();

            Node left = graphDb.createNode();
            Node right = graphDb.createNode();

            parent.createRelationshipTo( left, relType );
            parent.createRelationshipTo( right, relType );

            nodes.add( left );
            nodes.add( right );
        }
    }
}
catch ( TransactionTerminatedException ignored )
{
    return String.format( "Created tree up to depth %s in 1 sec", depth );
}
```

After waiting for some time, we decide to terminate the transaction. This is done from a separate thread.

```
tx.terminate();
```

Running this will execute the long running transaction for about one second and prints the maximum depth of the tree that was created before the transaction was terminated. No changes are actually made to the data — because the transaction has been terminated, the end result is as if no operations were performed.

Example output

```
Created tree up to depth 14 in 1 sec
```

Finally, let's shut down the database again.

```
graphDb.shutdown();
```

4.16. Execute Cypher queries from Java

The full source code of the example: JavaQuery.java *(https://github.com/neo4j/neo4j-documentation/blob/3.4/cypher/cypher-docs/src/test/java/org/neo4j/cypher/example/JavaQuery.java)*

In Java, you can use the Cypher query language as per the example below. First, let's add some data.

```
GraphDatabaseService db = new GraphDatabaseFactory().newEmbeddedDatabase( databaseDirectory );

try ( Transaction tx = db.beginTx())
{
    Node myNode = db.createNode();
    myNode.setProperty( "name", "my node" );
    tx.success();
}
```

Execute a query:

```
try ( Transaction ignored = db.beginTx();
      Result result = db.execute( "MATCH (n {name: 'my node'}) RETURN n, n.name" ) )
{
    while ( result.hasNext() )
    {
        Map<String,Object> row = result.next();
        for ( Entry<String,Object> column : row.entrySet() )
        {
            rows += column.getKey() + ": " + column.getValue() + "; ";
        }
        rows += "\n";
    }
}
```

In the above example, we also show how to iterate over the rows of the `Result` (https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/graphdb/Result.html).

The code will generate:

```
n.name: my node; n: Node[0];
```

When using an `Result`, you should consume the entire result (iterate over all rows using `next()`, iterating over the iterator from `columnAs()` or calling for example `resultAsString()`). Failing to do so will not properly clean up resources used by the `Result` object, leading to unwanted behavior, such as leaking transactions. In case you don't want to iterate over all of the results, make sure to invoke `close()` as soon as you are done, to release the resources tied to the result.

Using a try-with-resources statement *(http://docs.oracle.com/javase/tutorial/essential/exceptions/tryResourceClose.html)* will make sure that the result is closed at the end of the statement. This is the recommended way to handle results.

You can also get a list of the columns in the result like this:

```
List<String> columns = result.columns();
```

This gives us:

```
[n, n.name]
```

To fetch the result items from a single column, do like below. In this case we'll have to read the property from the node and not from the result.

```
Iterator<Node> n_column = result.columnAs( "n" );
for ( Node node : Iterators.asIterable( n_column ) )
{
    nodeResult = node + ": " + node.getProperty( "name" );
}
```

In this case there's only one node in the result:

```
Node[0]: my node
```

Only use this if the result only contains a single column, or you are only interested in a single column of the result.

`resultAsString()`, `writeAsStringTo()`, `columnAs()` cannot be called more than once on the same `Result` object, as they consume the result. In the same way, part of the result gets consumed for every call to `next()`. You should instead use only one and if you need the facilities of the other methods on the same query result instead create a new `Result`.

For more information on the Java interface to Cypher, see the Java API *(https://neo4j.com/docs/java-reference/3.4/javadocs/index.html)*.

For more information and examples for Cypher, see Developer Manual Cypher.

4.17. Query parameters

For more information on parameters see the Developer Manual.

Below follows example of how to use parameters when executing Cypher queries from Java.

Node id

```
Map<String, Object> params = new HashMap<>();
params.put( "id", 0 );
String query = "MATCH (n) WHERE id(n) = $id RETURN n.name";
Result result = db.execute( query, params );
```

Node object

```
Map<String, Object> params = new HashMap<>();
params.put( "node", andreasNode );
String query = "MATCH (n:Person) WHERE n = $node RETURN n.name";
Result result = db.execute( query, params );
```

Multiple node ids

```
Map<String, Object> params = new HashMap<>();
params.put( "ids", Arrays.asList( 0, 1, 2 ) );
String query = "MATCH (n) WHERE id(n) IN $ids RETURN n.name";
Result result = db.execute( query, params );
```

String literal

```
Map<String, Object> params = new HashMap<>();
params.put( "name", "Johan" );
String query = "MATCH (n:Person) WHERE n.name = $name RETURN n";
Result result = db.execute( query, params );
```

Index value

```
Map<String, Object> params = new HashMap<>();
params.put( "value", "Michaela" );
String query = "START n=node:people(name = $value) RETURN n";
Result result = db.execute( query, params );
```

Index query

```
Map<String, Object> params = new HashMap<>();
params.put( "query", "name:Andreas" );
String query = "START n=node:people($query) RETURN n";
Result result = db.execute( query, params );
```

Numeric parameters for `SKIP` and `LIMIT`

```
Map<String, Object> params = new HashMap<>();
params.put( "s", 1 );
params.put( "l", 1 );
String query = "MATCH (n:Person) RETURN n.name SKIP $s LIMIT $l";
Result result = db.execute( query, params );
```

Regular expression

```
Map<String, Object> params = new HashMap<>();
params.put( "regex", ".*h.*" );
String query = "MATCH (n:Person) WHERE n.name =~ $regex RETURN n.name";
Result result = db.execute( query, params );
```

Create node with properties

```
Map<String, Object> props = new HashMap<>();
props.put( "name", "Andres" );
props.put( "position", "Developer" );

Map<String, Object> params = new HashMap<>();
params.put( "props", props );
String query = "CREATE ($props)";
db.execute( query, params );
```

Create multiple nodes with properties

```
Map<String, Object> n1 = new HashMap<>();
n1.put( "name", "Andres" );
n1.put( "position", "Developer" );
n1.put( "awesome", true );

Map<String, Object> n2 = new HashMap<>();
n2.put( "name", "Michael" );
n2.put( "position", "Developer" );
n2.put( "children", 3 );

Map<String, Object> params = new HashMap<>();
List<Map<String, Object>> maps = Arrays.asList( n1, n2 );
params.put( "props", maps );
String query = "UNWIND $props AS properties CREATE (n:Person) SET n = properties RETURN n";
db.execute( query, params );
```

Setting all properties on node

```
Map<String, Object> n1 = new HashMap<>();
n1.put( "name", "Andres" );
n1.put( "position", "Developer" );

Map<String, Object> params = new HashMap<>();
params.put( "props", n1 );

String query = "MATCH (n:Person) WHERE n.name='Michaela' SET n = $props";
db.execute( query, params );
```

Chapter 5. The traversal framework

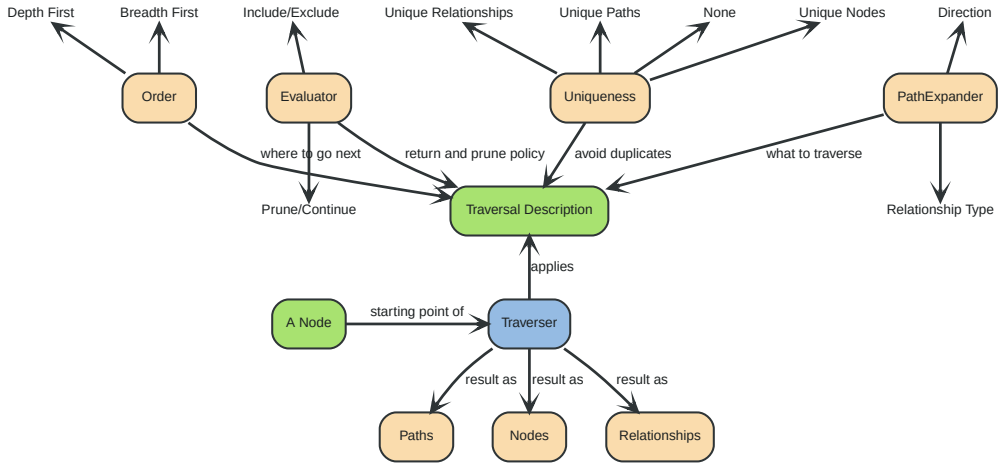
The Neo4j Traversal API *(https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/graphdb/traversal/package-summary.html)* is a callback based, lazily executed way of specifying desired movements through a graph in Java. Some traversal examples are collected under Traversal.

You can also use the Cypher query language as a powerful declarative way to query the graph.

5.1. Main concepts

Here follows a short explanation of all different methods that can modify or add to a traversal description.

- *Pathexpanders* — define what to traverse, typically in terms of relationship direction and type.
- *Order* — for example depth-first or breadth-first.
- *Uniqueness* — visit nodes (relationships, paths) only once.
- *Evaluator* — decide what to return and whether to stop or continue traversal beyond the current position.
- *Starting nodes* where the traversal will begin.



See Traversal framework Java API for more details.

5.2. Traversal framework Java API

The traversal framework consists of a few main interfaces in addition to `Node` and `Relationship`: `TraversalDescription`, `Evaluator`, `Traverser` and `Uniqueness` are the main ones. The `Path` interface also has a special purpose in traversals, since it is used to represent a position in the graph when evaluating that position. Furthermore the `PathExpander` (replacing `RelationshipExpander` and `Expander`) interface is central to traversals, but users of the API rarely need to implement it. There are also a set of interfaces for advanced use, when explicit control over the traversal order is required: `BranchSelector`, `BranchOrderingPolicy` and `TraversalBranch`.

5.2.1. TraversalDescription

The TraversalDescription *(https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/graphdb/traversal/TraversalDescription.html)* is the main interface used for defining and initializing traversals. It is not meant to be implemented by users of the traversal framework, but rather to be provided by the implementation of the traversal framework as a way for the user to describe traversals. `TraversalDescription` instances are immutable and its methods returns a new `TraversalDescription` that is modified compared to the object the method was invoked

on with the arguments of the method.

Relationships

Adds a relationship type to the list of relationship types to traverse. By default that list is empty and it means that it will traverse *all relationships*, regardless of type. If one or more relationships are added to this list *only the added* types will be traversed. There are two methods, one including direction *(https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/graphdb/traversal/TraversalDescription.html#relationships-org.neo4j.graphdb.RelationshipType-org.neo4j.graphdb.Direction-)* and another one excluding direction *(https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/graphdb/traversal/TraversalDescription.html#relationships-org.neo4j.graphdb.RelationshipType-)*, where the latter traverses relationships in both directions *(https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/graphdb/Direction.html#BOTH)*.

5.2.2. Evaluator

Evaluator *(https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/graphdb/traversal/Evaluator.html)*s are used for deciding, at each position (represented as a `Path`): should the traversal continue, and/or should the node be included in the result. Given a `Path`, it asks for one of four actions for that branch of the traversal:

- `Evaluation.INCLUDE_AND_CONTINUE`: Include this node in the result and continue the traversal
- `Evaluation.INCLUDE_AND_PRUNE`: Include this node in the result, but don't continue the traversal
- `Evaluation.EXCLUDE_AND_CONTINUE`: Exclude this node from the result, but continue the traversal
- `Evaluation.EXCLUDE_AND_PRUNE`: Exclude this node from the result and don't continue the traversal

More than one evaluator can be added. Note that evaluators will be called for all positions the traverser encounters, even for the start node.

5.2.3. Traverser

The Traverser *(https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/graphdb/traversal/Traverser.html)* object is the result of invoking traverse() *(https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/graphdb/traversal/TraversalDescription.html#traverse-org.neo4j.graphdb.Node-)* of a TraversalDescription object. It represents a traversal positioned in the graph, and a specification of the format of the result. The actual traversal is performed lazily each time the `next()`-method of the iterator of the `Traverser` is invoked.

5.2.4. Uniqueness

Sets the rules for how positions can be revisited during a traversal as stated in Uniqueness *(https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/graphdb/traversal/Uniqueness.html)*. Default if not set is NODE_GLOBAL *(https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/graphdb/traversal/Uniqueness.html#NODE_GLOBAL)*.

A Uniqueness can be supplied to the TraversalDescription to dictate under what circumstances a traversal may revisit the same position in the graph. The various uniqueness levels that can be used in Neo4j are:

- `NONE`: Any position in the graph may be revisited.
- `NODE_GLOBAL` uniqueness: No node in the entire graph may be visited more than once. This could potentially consume a lot of memory since it requires keeping an in-memory data structure remembering all the visited nodes.
- `RELATIONSHIP_GLOBAL` uniqueness: no relationship in the entire graph may be visited more than

once. For the same reasons as `NODE_GLOBAL` uniqueness, this could use up a lot of memory. But since graphs typically have a larger number of relationships than nodes, the memory overhead of this uniqueness level could grow even quicker.

- `NODE_PATH` uniqueness: A node may not occur previously in the path reaching up to it.
- `RELATIONSHIP_PATH` uniqueness: A relationship may not occur previously in the path reaching up to it.
- `NODE_RECENT` uniqueness: Similar to `NODE_GLOBAL` uniqueness in that there is a global collection of visited nodes each position is checked against. This uniqueness level does however have a cap on how much memory it may consume in the form of a collection that only contains the most recently visited nodes. The size of this collection can be specified by providing a number as the second argument to the TraversalDescription.uniqueness()-method along with the uniqueness level.
- `RELATIONSHIP_RECENT` uniqueness: Works like `NODE_RECENT` uniqueness, but with relationships instead of nodes.

Depth first / Breadth first

These are convenience methods for setting preorder depth-first *(https://en.wikipedia.org/wiki/Depth-first_search)*/ breadth-first *(https://en.wikipedia.org/wiki/Breadth-first_search)* `BranchSelector`|`ordering` policies. The same result can be achieved by calling the order *(https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/graphdb/traversal/TraversalDescription.html#order-org.neo4j.graphdb.traversal.BranchOrderingPolicy-)* method with ordering policies from BranchOrderingPolicies *(https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/graphdb/traversal/BranchOrderingPolicies.html)*, or to write your own BranchSelector/BranchOrderingPolicy and pass in.

5.2.5. Order — How to move through branches?

A more generic version of depthFirst/breadthFirst methods in that it allows an arbitrary BranchOrderingPolicy *(https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/graphdb/traversal/BranchOrderingPolicy.html)* to be injected into the description.

5.2.6. BranchSelector

A `BranchSelector`/`BranchOrderingPolicy` is used for selecting which branch of the traversal to attempt next. This is used for implementing traversal orderings. The traversal framework provides a few basic ordering implementations:

- `BranchOrderingPolicies.PREORDER_DEPTH_FIRST`: Traversing depth first, visiting each node before visiting its child nodes.
- `BranchOrderingPolicies.POSTORDER_DEPTH_FIRST`: Traversing depth first, visiting each node after visiting its child nodes.
- `BranchOrderingPolicies.PREORDER_BREADTH_FIRST`: Traversing breadth first, visiting each node before visiting its child nodes.
- `BranchOrderingPolicies.POSTORDER_BREADTH_FIRST`: Traversing breadth first, visiting each node after visiting its child nodes.

Please note that breadth first traversals have a higher memory overhead than depth first traversals.

`BranchSelector`s carries state and hence needs to be uniquely instantiated for each traversal. Therefore it is supplied to the `TraversalDescription` through a `BranchOrderingPolicy` interface, which

is a factory of `BranchSelector` instances.

A user of the Traversal framework rarely needs to implement his own `BranchSelector` or `BranchOrderingPolicy`, it is provided to let graph algorithm implementors provide their own traversal orders. The Neo4j Graph Algorithms package contains for example a `BestFirst` order `BranchSelector` /`BranchOrderingPolicy` that is used in BestFirst search algorithms such as A* and Dijkstra.

BranchOrderingPolicy

A factory for creating `BranchSelector`s to decide in what order branches are returned (where a branch's position is represented as a Path *(https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/graphdb/Path.html)* from the start node to the current node). Common policies are depth-first *(https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/graphdb/traversal/TraversalDescription.html#depthFirst--)* and breadth-first *(https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/graphdb/traversal/TraversalDescription.html#breadthFirst--)* and that's why there are convenience methods for those. For example, calling TraversalDescription#depthFirst() *(https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/graphdb/traversal/TraversalDescription.html#depthFirst--)* is equivalent to:

```
description.order( BranchOrderingPolicies.PREORDER_DEPTH_FIRST );
```

TraversalBranch

An object used by the BranchSelector to get more branches from a certain branch. In essence these are a composite of a Path and a RelationshipExpander that can be used to get new TraversalBranch *(https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/graphdb/traversal/TraversalBranch.html)*es from the current one.

5.2.7. Path

A Path *(https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/graphdb/Path.html)* is a general interface that is part of the Neo4j API. In the traversal API of Neo4j the use of Paths are twofold. Traversers can return their results in the form of the Paths of the visited positions in the graph that are marked for being returned. Path objects are also used in the evaluation of positions in the graph, for determining if the traversal should continue from a certain point or not, and whether a certain position should be included in the result set or not.

5.2.8. PathExpander/RelationshipExpander

The traversal framework use `PathExpander`s (replacing `RelationshipExpander`) to discover the relationships that should be followed from a particular path to further branches in the traversal.

5.2.9. Expander

A more generic version of relationships where a `RelationshipExpander` is injected, defining all relationships to be traversed for any given node.

The `Expander` interface is an extension of the `RelationshipExpander` interface that makes it possible to build customized versions of an `Expander`. The implementation of `TraversalDescription` uses this to provide methods for defining which relationship types to traverse, this is the usual way a user of the API would define a `RelationshipExpander` — by building it internally in the `TraversalDescription`.

All the RelationshipExpanders provided by the Neo4j traversal framework also implement the Expander interface. For a user of the traversal API it is easier to implement the PathExpander/RelationshipExpander interface, since it only contains one method — the method for

getting the relationships from a path/node, the methods that the Expander interface adds are just for building new Expanders.

5.2.10. How to use the Traversal framework

A traversal description *(https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/graphdb/traversal/TraversalDescription.html)* is built using a fluent interface and such a description can then spawn traversers *(https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/graphdb/traversal/Traverser.html)*.

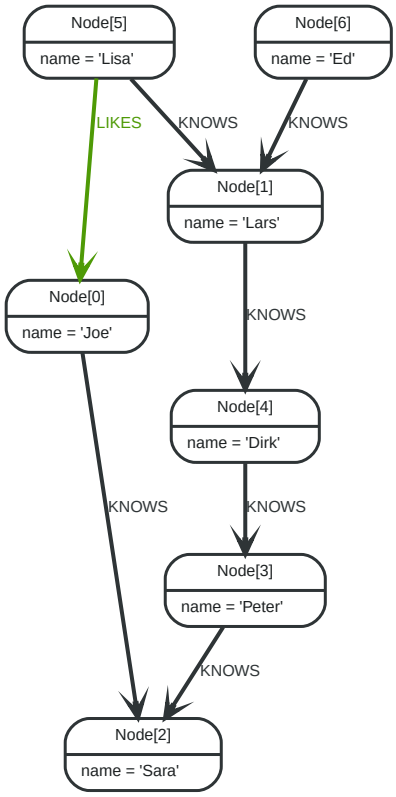


Figure 5. Traversal Example Graph

With the definition of the `RelationshipTypes` as

```
private enum Rels implements RelationshipType
{
    LIKES, KNOWS
}
```

The graph can be traversed with for example the following traverser, starting at the ``Joe'' node:

```
for ( Path position : db.traversalDescription()
        .depthFirst()
        .relationships( Rels.KNOWS )
        .relationships( Rels.LIKES, Direction.INCOMING )
        .evaluator( Evaluators.toDepth( 5 ) )
        .traverse( node ) )
{
    output += position + "\n";
}
```

The traversal will output:

```
(0)
(0)<-[LIKES,1]-(5)
(0)<-[LIKES,1]-(5)-[KNOWS,6]->(1)
(0)<-[LIKES,1]-(5)-[KNOWS,6]->(1)<-[KNOWS,5]-(6)
(0)<-[LIKES,1]-(5)-[KNOWS,6]->(1)-[KNOWS,4]->(4)
(0)<-[LIKES,1]-(5)-[KNOWS,6]->(1)-[KNOWS,4]->(4)-[KNOWS,3]->(3)
(0)<-[LIKES,1]-(5)-[KNOWS,6]->(1)-[KNOWS,4]->(4)-[KNOWS,3]->(3)-[KNOWS,2]->(2)
```

Since `TraversalDescription` *(https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/graphdb/traversal/TraversalDescription.html)*s are immutable it is also useful to create template descriptions which holds common settings shared by different traversals. For example, let's start with this traverser:

```
friendsTraversal = db.traversalDescription()
        .depthFirst()
        .relationships( Rels.KNOWS )
        .uniqueness( Uniqueness.RELATIONSHIP_GLOBAL );
```

This traverser would yield the following output (we will keep starting from the ``Joe'' node):

```
(0)
(0)-[KNOWS,0]->(2)
(0)-[KNOWS,0]->(2)<-[KNOWS,2]-(3)
(0)-[KNOWS,0]->(2)<-[KNOWS,2]-(3)<-[KNOWS,3]-(4)
(0)-[KNOWS,0]->(2)<-[KNOWS,2]-(3)<-[KNOWS,3]-(4)<-[KNOWS,4]-(1)
(0)-[KNOWS,0]->(2)<-[KNOWS,2]-(3)<-[KNOWS,3]-(4)<-[KNOWS,4]-(1)<-[KNOWS,6]-(5)
(0)-[KNOWS,0]->(2)<-[KNOWS,2]-(3)<-[KNOWS,3]-(4)<-[KNOWS,4]-(1)<-[KNOWS,5]-(6)
```

Now let's create a new traverser from it, restricting depth to three:

```
for ( Path path : friendsTraversal
        .evaluator( Evaluators.toDepth( 3 ) )
        .traverse( node ) )
{
    output += path + "\n";
}
```

This will give us the following result:

```
(0)
(0)-[KNOWS,0]->(2)
(0)-[KNOWS,0]->(2)<-[KNOWS,2]-(3)
(0)-[KNOWS,0]->(2)<-[KNOWS,2]-(3)<-[KNOWS,3]-(4)
```

Or how about from depth two to four? That's done like this:

```
for ( Path path : friendsTraversal
        .evaluator( Evaluators.fromDepth( 2 ) )
        .evaluator( Evaluators.toDepth( 4 ) )
        .traverse( node ) )
{
    output += path + "\n";
}
```

This traversal gives us:

```
(0)-[KNOWS,0]->(2)<-[KNOWS,2]-(3)
(0)-[KNOWS,0]->(2)<-[KNOWS,2]-(3)<-[KNOWS,3]-(4)
(0)-[KNOWS,0]->(2)<-[KNOWS,2]-(3)<-[KNOWS,3]-(4)<-[KNOWS,4]-(1)
```

For various useful evaluators, see the Evaluators *(https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/graphdb/traversal/Evaluators.html)* Java API or simply implement the Evaluator *(https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/graphdb/traversal/Evaluator.html)* interface yourself.

If you're not interested in the `Path` *(https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/graphdb/Path.html)*s, but the `Node` *(https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/graphdb/Node.html)*s you can transform the traverser into an iterable of nodes *(https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/graphdb/traversal/Traverser.html#nodes--)* like this:

```
for ( Node currentNode : friendsTraversal
        .traverse( node )
        .nodes() )
{
    output += currentNode.getProperty( "name" ) + "\n";
}
```

In this case we use it to retrieve the names:

```
Joe
Sara
Peter
Dirk
Lars
Lisa
Ed
```

Relationships *(https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/graphdb/traversal/Traverser.html#relationships--)* are fine as well, here's how to get them:

```
for ( Relationship relationship : friendsTraversal
        .traverse( node )
        .relationships() )
{
    output += relationship.getType().name() + "\n";
}
```

Here the relationship types are written, and we get:

```
KNOWS
KNOWS
KNOWS
KNOWS
KNOWS
KNOWS
```

The source code for the traversers in this example is available at: TraversalExample.java *(https://github.com/neo4j/neo4j/blob/3.4/community/embedded-examples/src/main/java/org/neo4j/examples/TraversalExample.java)*

Chapter 6. Manual indexing

This chapter focuses on how to use the Explicit Indexes. As of Neo4j 2.0, this is not the favored method of indexing data in Neo4j, instead we recommend defining indexes in the database schema.

However, support for explicit indexes remains, because certain features, such as full-text search, are not yet handled by the new indexes.

6.1. Introduction

Explicit indexing operations are part of the Neo4j index API *(https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/graphdb/index/package-summary.html)*.

Each index is tied to a unique, user-specified name (for example "first_name" or "books") and can index either nodes *(https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/graphdb/Node.html)* or relationships *(https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/graphdb/Relationship.html)*.

The default index implementation is provided by the `neo4j-lucene-index` component, which is included in the standard Neo4j download. It can also be downloaded separately from http://repo1.maven.org/maven2/org/neo4j/neo4j-lucene-index/ . For Maven users, the `neo4j-lucene-index` component has the coordinates `org.neo4j:neo4j-lucene-index` and should be used with the same version of `org.neo4j:neo4j-kernel`. Different versions of the index and kernel components are not compatible in the general case. Both components are included transitively by the `org.neo4j:neo4j:pom` artifact which makes it simple to keep the versions in sync.

Transactions

All modifying index operations must be performed inside a transaction, as with any modifying operation in Neo4j.

6.2. Create

An index is created if it doesn't exist when you ask for it. Unless you give it a custom configuration, it will be created with default configuration and backend.

To set the stage for our examples, let's create some indexes to begin with:

```
IndexManager index = graphDb.index();
Index<Node> actors = index.forNodes( "actors" );
Index<Node> movies = index.forNodes( "movies" );
RelationshipIndex roles = index.forRelationships( "roles" );
```

This will create two node indexes and one relationship index with default configuration. See Relationship indexes for more information specific to relationship indexes.

See Configuration and fulltext indexes for how to create *fulltext* indexes.

You can also check if an index exists like this:

```
IndexManager index = graphDb.index();
boolean indexExists = index.existsForNodes( "actors" );
```

6.3. Delete

Indexes can be deleted. When deleting, the entire contents of the index will be removed as well as its

associated configuration. An index can be created with the same name at a later point in time.

```
IndexManager index = graphDb.index();
Index<Node> actors = index.forNodes( "actors" );
actors.delete();
```

Note that the actual deletion of the index is made during the commit of *the surrounding transaction*. Calls made to such an index instance after delete() *(https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/graphdb/index/Index.html#delete--)* has been called are invalid inside that transaction as well as outside (if the transaction is successful), but will become valid again if the transaction is rolled back.

6.4. Add

Each index supports associating any number of key-value pairs with any number of entities (nodes or relationships), where each association between entity and key-value pair is performed individually. To begin with, let's add a few nodes to the indexes:

```
// Actors
Node reeves = graphDb.createNode();
reeves.setProperty( "name", "Keanu Reeves" );
actors.add( reeves, "name", reeves.getProperty( "name" ) );
Node bellucci = graphDb.createNode();
bellucci.setProperty( "name", "Monica Bellucci" );
actors.add( bellucci, "name", bellucci.getProperty( "name" ) );
// multiple values for a field, in this case for search only
// and not stored as a property.
actors.add( bellucci, "name", "La Bellucci" );
// Movies
Node theMatrix = graphDb.createNode();
theMatrix.setProperty( "title", "The Matrix" );
theMatrix.setProperty( "year", 1999 );
movies.add( theMatrix, "title", theMatrix.getProperty( "title" ) );
movies.add( theMatrix, "year", theMatrix.getProperty( "year" ) );
Node theMatrixReloaded = graphDb.createNode();
theMatrixReloaded.setProperty( "title", "The Matrix Reloaded" );
theMatrixReloaded.setProperty( "year", 2003 );
movies.add( theMatrixReloaded, "title", theMatrixReloaded.getProperty( "title" ) );
movies.add( theMatrixReloaded, "year", 2003 );
Node malena = graphDb.createNode();
malena.setProperty( "title", "Malèna" );
malena.setProperty( "year", 2000 );
movies.add( malena, "title", malena.getProperty( "title" ) );
movies.add( malena, "year", malena.getProperty( "year" ) );
```

Note that there can be multiple values associated with the same entity and key.

Next up, we'll create relationships and index them as well:

```
// we need a relationship type
RelationshipType ACTS_IN = RelationshipType.withName( "ACTS_IN" );
// create relationships
Relationship role1 = reeves.createRelationshipTo( theMatrix, ACTS_IN );
role1.setProperty( "name", "Neo" );
roles.add( role1, "name", role1.getProperty( "name" ) );
Relationship role2 = reeves.createRelationshipTo( theMatrixReloaded, ACTS_IN );
role2.setProperty( "name", "Neo" );
roles.add( role2, "name", role2.getProperty( "name" ) );
Relationship role3 = bellucci.createRelationshipTo( theMatrixReloaded, ACTS_IN );
role3.setProperty( "name", "Persephone" );
roles.add( role3, "name", role3.getProperty( "name" ) );
Relationship role4 = bellucci.createRelationshipTo( malena, ACTS_IN );
role4.setProperty( "name", "Malèna Scordia" );
roles.add( role4, "name", role4.getProperty( "name" ) );
```

After these operations, our example graph looks like this:

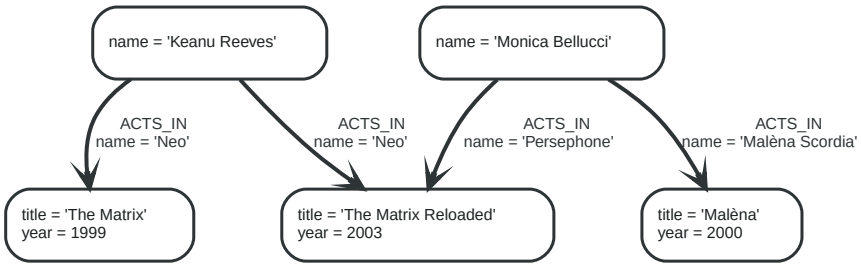


Figure 6. Movie and Actor Graph

6.5. Remove

Removing *(https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/graphdb/index/Index.html#remove-T-java.lang.String-java.lang.Object-)* from an index is similar to adding, but can be done by supplying one of the following combinations of arguments:

- entity
- entity, key
- entity, key, value

```
// completely remove bellucci from the actors index
actors.remove( bellucci );
// remove any "name" entry of bellucci from the actors index
actors.remove( bellucci, "name" );
// remove the "name" -> "La Bellucci" entry of bellucci
actors.remove( bellucci, "name", "La Bellucci" );
```

6.6. Update

To update an index entry, the old one must be removed and a new one added. For details on removing index entries, see Remove.

Remember that a node or relationship can be associated with any number of key-value pairs in an index. This means that you can index a node or relationship with many key-value pairs that have the same key. In the case where a property value changes and you'd like to update the index, it's not enough to just index the new value — you'll have to remove the old value as well.

Here's a code example that demonstrates how it's done:

```
// create a node with a property
// so we have something to update later on
Node fishburn = graphDb.createNode();
fishburn.setProperty( "name", "Fishburn" );
// index it
actors.add( fishburn, "name", fishburn.getProperty( "name" ) );
// update the index entry
// when the property value changes
actors.remove( fishburn, "name", fishburn.getProperty( "name" ) );
fishburn.setProperty( "name", "Laurence Fishburn" );
actors.add( fishburn, "name", fishburn.getProperty( "name" ) );
```

6.7. Search

An index can be searched in two ways, get *(https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/graphdb/index/ReadableIndex.html#get-java.lang.String-java.lang.Object-)* and query *(https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/graphdb/index/ReadableIndex.html#query-java.lang.Object-)*. The `get` method will return exact matches to the given key-value pair, whereas `query`

exposes querying capabilities directly from the backend used by the index. For example the Lucene query syntax *(http://lucene.apache.org/core/5_4_0/queryparser/org/apache/lucene/queryparser/classic/package-summary.html#package_description)* can be used directly with the default indexing backend.

6.7.1. Get

This is how to search for a single exact match:

```
IndexHits<Node> hits = actors.get( "name", "Keanu Reeves" );
Node reeves = hits.getSingle();
```

IndexHits *(https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/graphdb/index/IndexHits.html)* is an `Iterable` with some additional useful methods. For example getSingle() *(https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/graphdb/index/IndexHits.html#getSingle--)* returns the first and only item from the result iterator, or `null` if there isn't any hit.

Here's how to get a single relationship by exact matching and retrieve its start and end nodes:

```
Relationship persephone = roles.get( "name", "Persephone" ).getSingle();
Node actor = persephone.getStartNode();
Node movie = persephone.getEndNode();
```

Finally, we can iterate over all exact matches from a relationship index:

```
for ( Relationship role : roles.get( "name", "Neo" ) )
{
    // this will give us Reeves twice
    Node reeves = role.getStartNode();
}
```

In case you don't iterate through all the hits, IndexHits.close() *(https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/graphdb/index/IndexHits.html#close--)* must be called explicitly.

6.7.2. Query

There are two query methods, one which uses a key-value signature where the value represents a query for values with the given key only. The other method is more generic and supports querying for more than one key-value pair in the same query.

Here's an example using the key-query option:

```
for ( Node actor : actors.query( "name", "*e*" ) )
{
    // This will return Reeves and Bellucci
}
```

In the following example the query uses multiple keys:

```
for ( Node movie : movies.query( "title:*Matrix* AND year:1999" ) )
{
    // This will return "The Matrix" from 1999 only.
}
```

Beginning a wildcard search with "*" or "?" is discouraged by Lucene, but will nevertheless work.

You can't have *any whitespace* in the search term with this syntax. See Querying with Lucene query objects for how to do that.

6.8. Relationship indexes

An index for relationships is just like an index for nodes, extended by providing support to constrain a search to relationships with a specific start and/or end node. These extra methods reside in the RelationshipIndex *(https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/graphdb/index/RelationshipIndex.html)* interface which extends Index<Relationship> *(https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/graphdb/index/Index.html)*.

Example of querying a relationship index:

```
// find relationships filtering on start node
// using exact matches
IndexHits<Relationship> reevesAsNeoHits;
reevesAsNeoHits = roles.get( "name", "Neo", reeves, null );
Relationship reevesAsNeo = reevesAsNeoHits.iterator().next();
reevesAsNeoHits.close();
// find relationships filtering on end node
// using a query
IndexHits<Relationship> matrixNeoHits;
matrixNeoHits = roles.query( "name", "*eo", null, theMatrix );
Relationship matrixNeo = matrixNeoHits.iterator().next();
matrixNeoHits.close();
```

And here's an example for the special case of searching for a specific relationship type:

```
// find relationships filtering on end node
// using a relationship type.
// this is how to add it to the index:
roles.add( reevesAsNeo, "type", reevesAsNeo.getType().name() );
// Note that to use a compound query, we can't combine committed
// and uncommitted index entries, so we'll commit before querying:
tx.success();
tx.close();

// and now we can search for it:
try ( Transaction tx = graphDb.beginTx() )
{
    IndexHits<Relationship> typeHits = roles.query( "type:ACTS_IN AND name:Neo", null, theMatrix );
    Relationship typeNeo = typeHits.iterator().next();
    typeHits.close();
```

Such an index can be useful if your domain has nodes with a very large number of relationships between them, since it reduces the search time for a relationship between two nodes. A good example where this approach pays dividends is in time series data, where we have readings represented as a relationship per occurrence.

6.9. Scores

The `IndexHits` interface exposes scoring *(https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/graphdb/index/IndexHits.html#currentScore--)* so that the index can communicate scores for the hits. Note that the result is not sorted by the score unless you explicitly specify that. See Sorting for how to sort by score.

```
IndexHits<Node> hits = movies.query( "title", "The*" );
for ( Node movie : hits )
{
    System.out.println( movie.getProperty( "title" ) + " " + hits.currentScore() );
}
```

6.10. Configuration and fulltext indexes

At the time of creation extra configuration can be specified to control the behavior of the index and which backend to use. For example to create a Lucene fulltext index:

```
IndexManager index = graphDb.index();
Index<Node> fulltextMovies = index.forNodes( "movies-fulltext",
        MapUtil.stringMap( IndexManager.PROVIDER, "lucene", "type", "fulltext" ) );
fulltextMovies.add( theMatrix, "title", "The Matrix" );
fulltextMovies.add( theMatrixReloaded, "title", "The Matrix Reloaded" );
// search in the fulltext index
Node found = fulltextMovies.query( "title", "reloAdEd" ).getSingle();
```

Here's an example of how to create an exact index which is case-insensitive:

```
Index<Node> index = graphDb.index().forNodes( "exact-case-insensitive",
        MapUtil.stringMap( "type", "exact", "to_lower_case", "true" ) );
Node node = graphDb.createNode();
index.add( node, "name", "Thomas Anderson" );
assertContains( index.query( "name", "\"Thomas Anderson\"" ), node );
assertContains( index.query( "name", "\"thoMas ANDerson\"" ), node );
```

In order to search for tokenized words, the `query` method has to be used. The `get` method will only match the full string value, not the tokens.

The configuration of the index is persisted once the index has been created. The `provider` configuration key is interpreted by Neo4j, but any other configuration is passed onto the backend index (e.g. Lucene) to interpret.

Table 3. Lucene indexing configuration parameters

Parameter	Possible values	Effect
`type`	`exact`, `fulltext`	`exact` is the default and uses a Lucene keyword analyzer *(http://lucene.apache.org/core/5_4_0/analyzers-common/org/apache/lucene/analysis/core/KeywordAnalyzer.html)*. `fulltext` uses a white-space tokenizer in its analyzer.
`to_lower_case`	`true`, `false`	This parameter goes together with `type`: `fulltext` and converts values to lower case during both additions and querying, making the index case insensitive. Defaults to `true`.
`analyzer`	the full class name of an Analyzer *(http://lucene.apache.org/core/5_4_0/core/org/apache/lucene/analysis/Analyzer.html)*	Overrides the `type` so that a custom analyzer can be used. Note: `to_lower_case` still affects lowercasing of string queries. If the custom analyzer uppercases the indexed tokens, string queries will not match as expected.

6.11. Extra features for Lucene indexes

6.11.1. Numeric ranges

Lucene supports smart indexing of numbers, querying for ranges and sorting such results, and so does its backend for Neo4j. To mark a value so that it is indexed as a numeric value, we can make use of the ValueContext *(https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/index/lucene/ValueContext.html)*

class, like this:

```
movies.add( theMatrix, "year-numeric", new ValueContext( 1999 ).indexNumeric() );
movies.add( theMatrixReloaded, "year-numeric", new ValueContext( 2003 ).indexNumeric() );
movies.add( malena, "year-numeric", new ValueContext( 2000 ).indexNumeric() );

int from = 1997;
int to = 1999;
hits = movies.query( QueryContext.numericRange( "year-numeric", from, to ) );
```

The same type must be used for indexing and querying. That is, you can't index a value as a Long and then query the index using an Integer.

By giving `null` as from/to argument, an open ended query is created. In the following example we are doing that, and have added sorting to the query as well:

```
hits = movies.query(
        QueryContext.numericRange( "year-numeric", from, null )
                .sortNumeric( "year-numeric", false ) );
```

From/to in the ranges defaults to be *inclusive*, but you can change this behavior by using two extra parameters:

```
movies.add( theMatrix, "score", new ValueContext( 8.7 ).indexNumeric() );
movies.add( theMatrixReloaded, "score", new ValueContext( 7.1 ).indexNumeric() );
movies.add( malena, "score", new ValueContext( 7.4 ).indexNumeric() );

// include 8.0, exclude 9.0
hits = movies.query( QueryContext.numericRange( "score", 8.0, 9.0, true, false ) );
```

6.11.2. Sorting

Lucene performs sorting very well, and that is also exposed in the index backend, through the QueryContext *(https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/index/lucene/QueryContext.html)* class:

```
hits = movies.query( "title", new QueryContext( "*" ).sort( "title" ) );
for ( Node hit : hits )
{
    // all movies with a title in the index, ordered by title
}
// or
hits = movies.query( new QueryContext( "title:*" ).sort( "year", "title" ) );
for ( Node hit : hits )
{
    // all movies with a title in the index, ordered by year, then title
}
```

We sort the results by relevance (score) like this:

```
hits = movies.query( "title", new QueryContext( "The*" ).sortByScore() );
for ( Node movie : hits )
{
    // hits sorted by relevance (score)
}
```

6.11.3. Querying with Lucene query objects

Instead of passing in Lucene query syntax queries, you can instantiate such queries programmatically

and pass in as argument, for example:

```
// a TermQuery will give exact matches
Node actor = actors.query( new TermQuery( new Term( "name", "Keanu Reeves" ) ) ).getSingle();
```

Note that the TermQuery *(http://lucene.apache.org/core/5_4_0/core/org/apache/lucene/search/TermQuery.html)* is basically the same thing as using the `get` method on the index.

This is how to perform *wildcard* searches using Lucene query objects:

```
hits = movies.query( new WildcardQuery( new Term( "title", "The Matrix*" ) ) );
for ( Node movie : hits )
{
    System.out.println( movie.getProperty( "title" ) );
}
```

Note that this allows for whitespace in the search string.

6.11.4. Compound queries

Lucene supports querying for multiple terms in the same query, like so:

```
hits = movies.query( "title:*Matrix* AND year:1999" );
```

Compound queries can't search across committed index entries and those who haven't got committed yet at the same time.

6.11.5. Default operator

The default operator (that is whether `AND` or `OR` is used in between different terms) in a query is `OR`. Changing that behavior is also done via the QueryContext *(https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/index/lucene/QueryContext.html)* class:

```
QueryContext query = new QueryContext( "title:*Matrix* year:1999" )
        .defaultOperator( Operator.AND );
hits = movies.query( query );
```

Chapter 7. Transaction management

In order to fully maintain data integrity and ensure good transactional behavior, Neo4j supports the ACID properties:

- atomicity: If any part of a transaction fails, the database state is left unchanged.
- consistency: Any transaction will leave the database in a consistent state.
- isolation: During a transaction, modified data cannot be accessed by other operations.
- durability: The DBMS can always recover the results of a committed transaction.

Specifically:

- All database operations that access the graph, indexes, or the schema must be performed in a transaction.
- The default isolation level is `READ_COMMITTED`.
- Data retrieved by traversals is not protected from modification by other transactions.
- Non-repeatable reads may occur (i.e., only write locks are acquired and held until the end of the transaction).
- One can manually acquire write locks on nodes and relationships to achieve higher level of isolation (`SERIALIZABLE`).
- Locks are acquired at the Node and Relationship level.
- Deadlock detection is built into the core transaction management.

7.1. Interaction cycle

All database operations that access the graph, indexes, or the schema must be performed in a transaction. Transactions are thread confined and can be nested as “flat nested transactions”. Flat nested transactions means that all nested transactions are added to the scope of the top level transaction. A nested transaction can mark the top level transaction for rollback, meaning the entire transaction will be rolled back. To only rollback changes made in a nested transaction is not possible.

The interaction cycle of working with transactions looks like this:

1. Begin a transaction.
2. Perform database operations.
3. Mark the transaction as successful or not.
4. Finish the transaction.

It is very important to finish each transaction. The transaction will not release the locks or memory it has acquired until it has been finished. The idiomatic use of transactions in Neo4j is to use a try-finally block, starting the transaction and then try to perform the graph operations. The last operation in the try block should mark the transaction as successful while the finally block should finish the transaction. Finishing the transaction will perform commit or rollback depending on the success status.

All modifications performed in a transaction are kept in memory. This means that very large updates have to be split into several top level transactions to avoid running out of memory. It must be a top level transaction since splitting up the work in many nested transactions will just add all the work to the top level transaction.

In an environment that makes use of *thread pooling* other errors may occur when failing to finish a transaction properly. Consider a leaked transaction that did not get finished properly. It will be tied to

a thread and when that thread gets scheduled to perform work starting a new (what looks to be a) top level transaction it will actually be a nested transaction. If the leaked transaction state is “marked for rollback” (which will happen if a deadlock was detected) no more work can be performed on that transaction. Trying to do so will result in error on each call to a write operation.

7.2. Isolation levels

Transactions in Neo4j use a read-committed isolation level, which means they will see data as soon as it has been committed and will not see data in other transactions that have not yet been committed. This type of isolation is weaker than serialization but offers significant performance advantages whilst being sufficient for the overwhelming majority of cases.

In addition, the Neo4j Java API enables explicit locking of nodes and relationships. Using locks gives the opportunity to simulate the effects of higher levels of isolation by obtaining and releasing locks explicitly. For example, if a write lock is taken on a common node or relationship, then all transactions will serialize on that lock — giving the effect of a serialization isolation level.

7.2.1. Lost updates in Cypher

In Cypher it is possible to acquire write locks to simulate improved isolation in some cases. Consider the case where multiple concurrent Cypher queries increment the value of a property. Due to the limitations of the read-committed isolation level, the increments might not result in a deterministic final value. If there is a direct dependency, Cypher will automatically acquire a write lock before reading. A direct dependency is when the right-hand side of a `SET` has a dependent property read in the expression, or in the value of a key-value pair in a literal map.

For example, the following query, if run by one hundred concurrent clients, will very likely not increment the property `n.prop` to 100, unless a write lock is acquired before reading the property value. This is because all queries would read the value of `n.prop` within their own transaction, and would not see the incremented value from any other transaction that has not yet committed. In the worst case scenario the final value would be as low as 1, if all threads perform the read before any has committed their transaction.

Requires a write lock, and Cypher automatically acquires one.

```
MATCH (n:X {id: 42})
SET n.prop = n.prop + 1
```

Also requires a write lock, and Cypher automatically acquires one.

```
MATCH (n)
SET n += { prop: n.prop + 1 }
```

Due to the complexity of determining such a dependency in the general case, Cypher does not cover any of the below example cases:

Variable depending on results from reading the property in an earlier statement.

```
MATCH (n)
WITH n.prop as p
// ... operations depending on p, producing k
SET n.prop = k + 1
```

Circular dependency between properties read and written in the same query.

```
MATCH (n)
SET n += { propA: n.propB + 1, propB: n.propA + 1 }
```

To ensure deterministic behavior also in the more complex cases, it is necessary to explicitly acquire a write lock on the node in question. In Cypher there is no explicit support for this, but it is possible to work around this limitation by writing to a temporary property.

Acquires a write lock for the node by writing to a dummy property before reading the requested value.

```
MATCH (n:X {id: 42})
SET n._LOCK_ = true
WITH n.prop as p
// ... operations depending on p, producing k
SET n.prop = k + 1
REMOVE n._LOCK_
```

The existence of the `SET n.LOCK` statement before the read of the `n.prop` read ensures the lock is acquired before the read action, and no updates will be lost due to enforced serialization of all concurrent queries on that specific node.

7.3. Default locking behavior

- When adding, changing or removing a property on a node or relationship a write lock will be taken on the specific node or relationship.
- When creating or deleting a node a write lock will be taken for the specific node.
- When creating or deleting a relationship a write lock will be taken on the specific relationship and both its nodes.

The locks will be added to the transaction and released when the transaction finishes.

7.4. Deadlocks

7.4.1. Understanding deadlocks

Since locks are used it is possible for deadlocks to happen. Neo4j will however detect any deadlock (caused by acquiring a lock) before they happen and throw an exception. Before the exception is thrown the transaction is marked for rollback. All locks acquired by the transaction are still being held but will be released when the transaction is finished (in the finally block as pointed out earlier). Once the locks are released other transactions that were waiting for locks held by the transaction causing the deadlock can proceed. The work performed by the transaction causing the deadlock can then be retried by the user if needed.

Experiencing frequent deadlocks is an indication of concurrent write requests happening in such a way that it is not possible to execute them while at the same time live up to the intended isolation and consistency. The solution is to make sure concurrent updates happen in a reasonable way. For example given two specific nodes (A and B), adding or deleting relationships to both these nodes in random order for each transaction will result in deadlocks when there are two or more transactions doing that concurrently. One solution is to make sure that updates always happens in the same order (first A then B). Another solution is to make sure that each thread/transaction does not have any conflicting writes to a node or relationship as some other concurrent transaction. This can for example be achieved by letting a single thread do all updates of a specific type.

Deadlocks caused by the use of other synchronization than the locks managed by Neo4j can still happen. Since all operations in the Neo4j API are thread safe unless specified otherwise, there is no need for external synchronization. Other code that requires synchronization should be synchronized in such a way that it never performs any Neo4j operation in the synchronized block.

7.4.2. Deadlock handling example code

Below you'll find examples of how deadlocks can be handled in procedures, server extensions or when using Neo4j embedded.

The full source code used for the code snippets can be found at DeadlockDocTest.java *(https://github.com/neo4j/neo4j-documentation/blob/3.4/kernel/src/test/java/examples/DeadlockDocTest.java)*.

When dealing with deadlocks in code, there are several issues you may want to address:

- Only do a limited amount of retries, and fail if a threshold is reached.
- Pause between each attempt to allow the other transaction to finish before trying again.
- A retry-loop can be useful not only for deadlocks, but for other types of transient errors as well.

In the following sections you'll find example code in Java which shows how this can be implemented.

Handling deadlocks using TransactionTemplate

If you don't want to write all the code yourself, there is a class called `TransactionTemplate (https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/helpers/TransactionTemplate.html)` that will help you achieve what's needed. Below is an example of how to create, customize, and use this template for retries in transactions.

First, define the base template:

```
TransactionTemplate template = new TransactionTemplate(  ).retries( 5 ).backoff( 3, TimeUnit.SECONDS );
```

Next, specify the database to use and a function to execute:

```
Object result = template.with(graphDatabaseService).execute( transaction -> {
    Object result1 = null;
    return result1;
} );
```

The operations that could lead to a deadlock should go into the `apply` method.

The `TransactionTemplate` uses a fluent API for configuration, and you can choose whether to set everything at once, or (as in the example) provide some details just before using it. The template allows setting a predicate for what exceptions to retry on, and also allows for easy monitoring of events that take place.

Handling deadlocks using a retry loop

If you want to roll your own retry-loop code, see below for inspiration. Here's an example of what a retry block might look like:

```
Throwable txEx = null;
int RETRIES = 5;
int BACKOFF = 3000;
for ( int i = 0; i < RETRIES; i++ )
{
    try ( Transaction tx = graphDatabaseService.beginTx() )
    {
        Object result = doStuff(tx);
        tx.success();
        return result;
    }
    catch ( Throwable ex )
    {
        txEx = ex;

        // Add whatever exceptions to retry on here
        if ( !(ex instanceof DeadlockDetectedException) )
        {
            break;
        }
    }

    // Wait so that we don't immediately get into the same deadlock
    if ( i < RETRIES - 1 )
    {
        try
        {
            Thread.sleep( BACKOFF );
        }
        catch ( InterruptedException e )
        {
            throw new TransactionFailureException( "Interrupted", e );
        }
    }
}

if ( txEx instanceof TransactionFailureException )
{
    throw ((TransactionFailureException) txEx);
}
else if ( txEx instanceof Error )
{
    throw ((Error) txEx);
}
else if ( txEx instanceof RuntimeException )
{
    throw ((RuntimeException) txEx);
}
else
{
    throw new TransactionFailureException( "Failed", txEx );
}
```

The above is the gist of what such a retry block would look like, and which you can customize to fit your needs.

7.5. Delete semantics

When deleting a node or a relationship all properties for that entity will be automatically removed but the relationships of a node will not be removed.

Neo4j enforces a constraint (upon commit) that all relationships must have a valid start node and end node. In effect this means that trying to delete a node that still has relationships attached to it will throw an exception upon commit. It is however possible to choose in which order to delete the node and the attached relationships as long as no relationships exist when the transaction is committed.

The delete semantics can be summarized in the following bullets:

- All properties of a node or relationship will be removed when it is deleted.

- A deleted node can not have any attached relationships when the transaction commits.
- It is possible to acquire a reference to a deleted relationship or node that has not yet been committed.
- Any write operation on a node or relationship after it has been deleted (but not yet committed) will throw an exception
- After commit trying to acquire a new or work with an old reference to a deleted node or relationship will throw an exception.

7.6. Creating unique nodes

In many use cases, a certain level of uniqueness is desired among entities. You could for instance imagine that only one user with a certain e-mail address may exist in a system. If multiple concurrent threads naively try to create the user, duplicates will be created. There are three main strategies for ensuring uniqueness, and they all work across High Availability and single-instance deployments.

7.6.1. Single thread

By using a single thread, no two threads will even try to create a particular entity simultaneously. On High Availability, an external single-threaded client can perform the operations on the cluster.

7.6.2. Get or create

The preferred way to get or create a unique node is to use unique constraints and Cypher. See Get or create unique node using Cypher and unique constraints for more information.

By using put-if-absent *(https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/graphdb/index/Index.html#putIfAbsent-T-java.lang.String-java.lang.Object-)* functionality, entity uniqueness can be guaranteed using an explicit index. Here the explicit index acts as the lock and will only lock the smallest part needed to guarantee uniqueness across threads and transactions.

See Get or create unique node using an explicit index for how to do this using the core Java API. When using the REST API, see REST Documentation Uniqueness.

7.6.3. Pessimistic locking

While this is a working solution, please consider using the preferred method outlined above instead.

By using explicit, pessimistic locking, unique creation of entities can be achieved in a multi-threaded environment. It is most commonly done by locking on a single or a set of common nodes.

See Pessimistic locking for node creation for how to do this using the core Java API.

7.7. Transaction events

A transaction event handler can be registered to receive Neo4j transaction events. Once it has been registered at a `GraphDatabaseService` instance it receives events for transactions before they are committed. Handlers get notified about transactions that have performed any write operation, and that will be committed. If `Transaction#success()` has not been called or the transaction has been marked as failed `Transaction#failure()` it will be rolled back, and no events are sent to the Handler.

Before a transaction is committed the Handler's `beforeCommit` method is called with the entire diff of modifications made in the transaction. At this point the transaction is still running, so changes can still

be made. The method may also throw an exception, which will prevent the transaction from being committed. If the transaction is rolled back, a call to the handler's `afterRollback` method will follow.

The order in which handlers are executed is undefined — there is no guarantee that changes made by one handler will be seen by other handlers.

If `beforeCommit` is successfully executed in all registered handlers the transaction is committed and the `afterCommit` method is called with the same transaction data. This call also includes the object returned from `beforeCommit`.

In `afterCommit` the transaction has been closed and access to anything outside `TransactionData` requires a new transaction to be opened. A `TransactionEventHandler` gets notified about transactions that have any changes accessible via `TransactionData` so some indexing and schema changes will not be triggering these events.

Chapter 8. Online Backup from Java

In order to programmatically backup your data full or subsequently incremental from a JVM based program, you need to write Java code like the following:

```
OnlineBackup backup = OnlineBackup.from( "127.0.0.1" );
backup.full( backupPath.getPath() );
assertTrue( "Should be consistent", backup.isConsistent() );
backup.incremental( backupPath.getPath() );
```

For more information, please see the Javadocs for OnlineBackup *(https://neo4j.com/docs/java-reference/3.4/javadocs/org/neo4j/backup/OnlineBackup.html)*.

Chapter 9. JMX metrics

For more monitoring options, see Neo4j Operations Manual Monitoring. Most of the monitoring features are only available in the Enterprise edition of Neo4j.

In order to be able to continuously get an overview of the health of a Neo4j database, there are different levels of monitoring facilities available. Most of these are exposed through JMX *(http://www.oracle.com/technetwork/java/javase/tech/javamanagement-140525.html)*. Neo4j Enterprise also has the ability to automatically report *metrics* to commonly used monitoring systems.

9.1. Adjusting remote JMX access to the Neo4j Server

Per default, the Neo4j Enterprise Server edition does not allow remote JMX connections, since the relevant options in the *conf/neo4j.conf* configuration file are commented out. To enable this feature, you have to remove the `#` characters from the various `com.sun.management.jmxremote` options there.

When commented in, the default values are set up to allow remote JMX connections with certain roles, refer to the *conf/jmx.password*, *conf/jmx.access*, and *conf/neo4j.conf* files for details.

Make sure that *conf/jmx.password* has the correct file permissions. The owner of the file has to be the user that will run the service, and the permissions should be read only for that user. On Unix systems, this is `0600`.

On Windows, follow the tutorial at http://docs.oracle.com/javase/8/docs/technotes/guides/management/security-windows.html to set the correct permissions. If you are running the service under the Local System Account, the user that owns the file and has access to it should be SYSTEM.

With this setup, you should be able to connect to JMX monitoring of the Neo4j server using `<IP-OF-SERVER>:3637`, with the username `monitor` and the password `Neo4j`.

Note that it is possible that you have to update the permissions and/or ownership of the *conf/jmx.password* and *conf/jmx.access* files — refer to the relevant section in *conf/neo4j.conf* for details.

For maximum security, please adjust at least the password settings in *conf/jmx.password* for a production installation.

For more details, see: http://docs.oracle.com/javase/8/docs/technotes/guides/management/agent.html.

9.2. How to connect to a Neo4j instance using JMX and JConsole

First, start your Neo4j instance, for example using

```
$NEO4j_HOME/bin/neo4j start
```

Now, start JConsole with

```
$JAVA_HOME/bin/jconsole
```

Connect to the process running your Neo4j database instance:

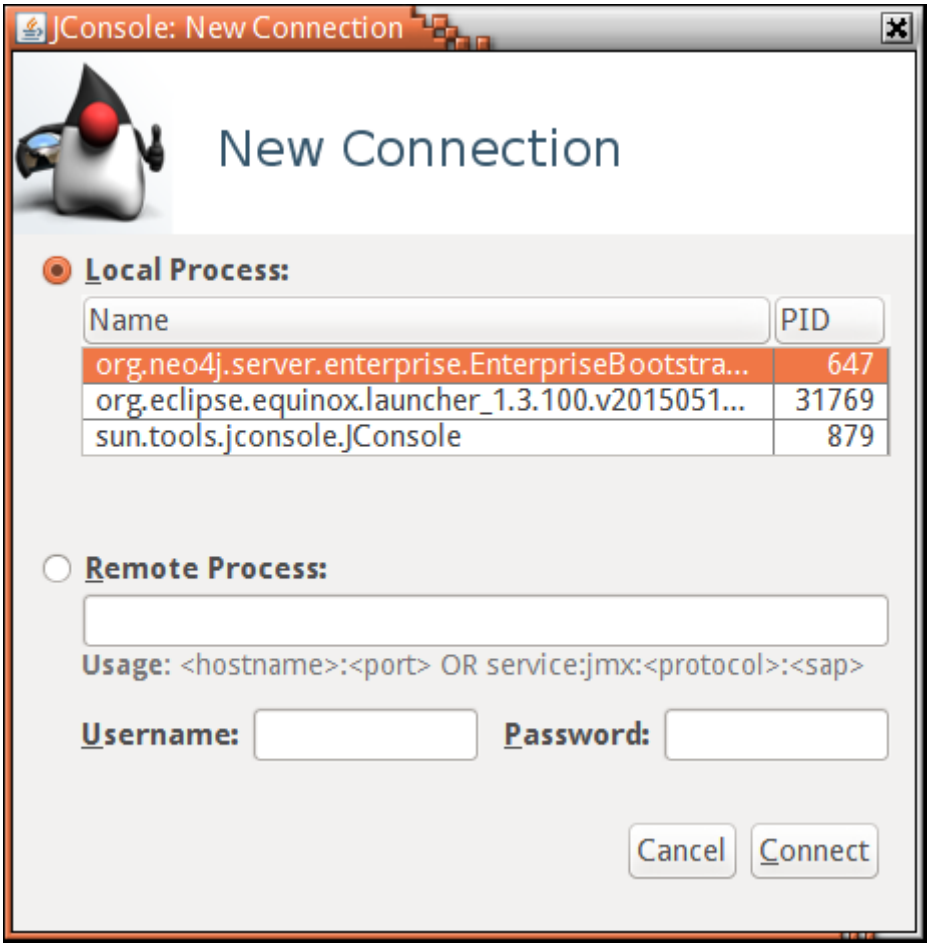


Figure 7. Connecting JConsole to the Neo4j Java process

Now, beside the MBeans exposed by the JVM, you will see an `org.neo4j` section in the MBeans tab. Under that, you will have access to all the monitoring information exposed by Neo4j.

For opening JMX to remote monitoring access, please see Adjusting remote JMX access to the Neo4j Server and the JMX documention *(http://docs.oracle.com/javase/8/docs/technotes/guides/management/agent.html)*.

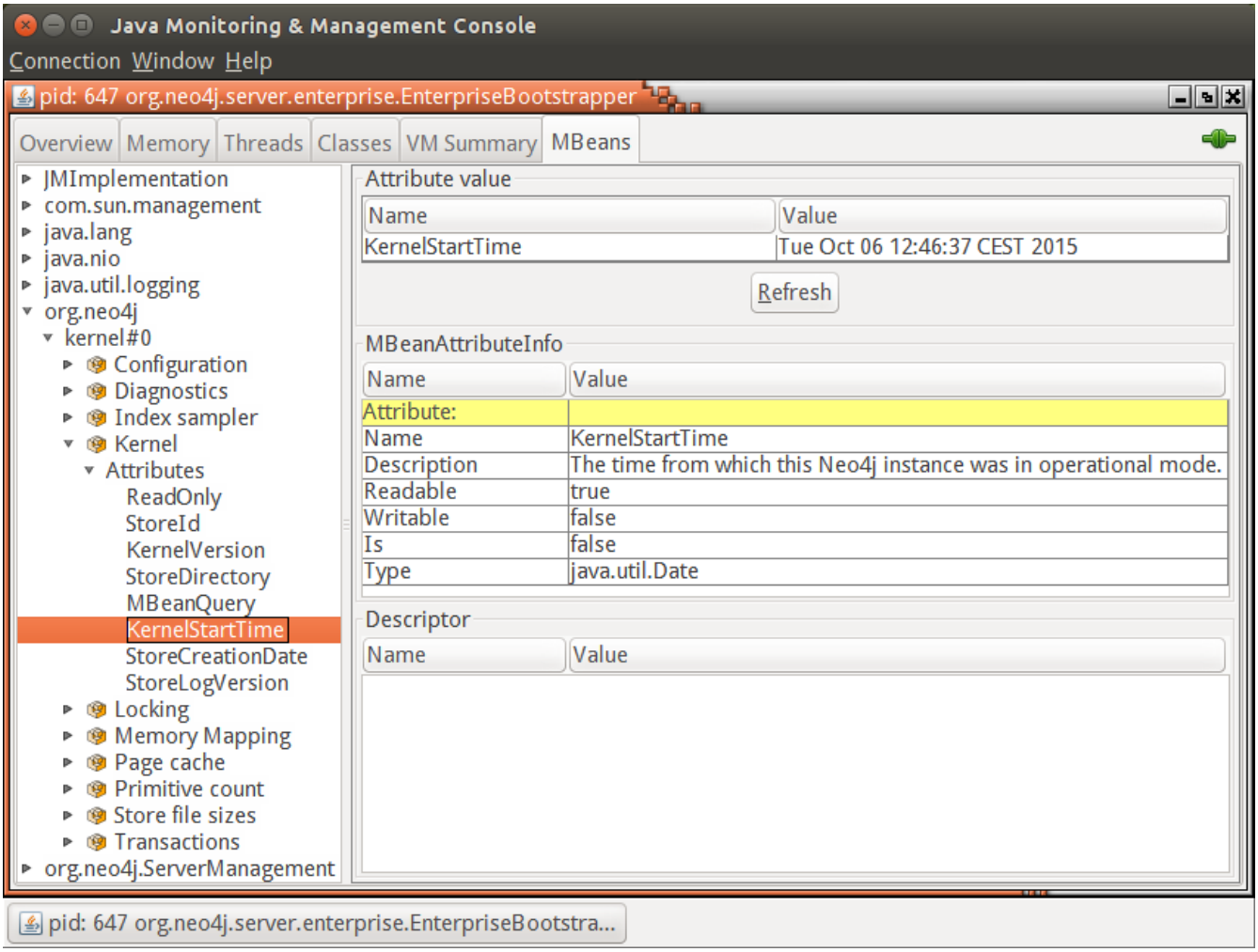


Figure 8. Neo4j MBeans View

9.3. How to connect to the JMX monitoring programmatically

In order to programmatically connect to the Neo4j JMX server, there are some convenience methods in the Neo4j Management component to help you find out the most commonly used monitoring attributes of Neo4j. See Reading a management attribute for an example.

Once you have access to this information, you can use it to for instance expose the values to SNMP *(http://en.wikipedia.org/wiki/Simple_Network_Management_Protocol)* or other monitoring systems.

9.4. Reference of supported JMX MBeans

Below is a complete reference of Neo4j JMX management beans in two parts. First part shows all beans available in Neo4j Enterprise Edition when the instance is part of a Causal Cluster. The second part shows the management beans that are uniquely available when running in High Availability mode.

For additional information on the primitive datatypes (`int`, `long` etc.) used in the JMX attributes, please see Developer Manual Introduction: Properties.

9.4.1. JMX MBeans available on instances in a Causal Cluster

The following JMX management beans are available on instances that are part of a Causal Cluster.

MBeans exposed by Neo4j

- Causal Clustering: Information about an instance participating in a causal cluster.
- Configuration: The configuration parameters used to configure Neo4j.
- Diagnostics: Diagnostics provided by Neo4j.
- Index sampler: Handle index sampling.
- Kernel: Information about the Neo4j kernel.
- Locking: Information about the Neo4j lock status.
- Memory Mapping: The status of Neo4j memory mapping.
- Page cache: Information about the Neo4j page cache. All numbers are counts and sums since the Neo4j instance was started.
- Primitive count: Estimates of the numbers of different kinds of Neo4j primitives.
- Reports: Reports operations.
- Store file sizes: This bean is deprecated, use StoreSize bean instead; Information about the sizes of the different parts of the Neo4j graph store.
- Store sizes: Information about the disk space used by different parts of the Neo4j graph store.
- Transactions: Information about the Neo4j transaction manager.

Table 4. MBean Causal Clustering (org.neo4j.management.CausalClustering) Attributes

Name	Description	Type	Read	Write
Information about an instance participating in a causal cluster				
`RaftLogSize`	The total amount of disk space used by the raft log, in bytes	`long`	yes	no
`ReplicatedStateSize`	The total amount of disk space used by the replicated states, in bytes	`long`	yes	no
`Role`	The current role this member has in the cluster	`String`	yes	no

Table 5. MBean Configuration (org.neo4j.jmx.impl.ConfigurationBean) Attributes

Name	Description	Type	Read	Write
The configuration parameters used to configure Neo4j				
`bolt.ssl_ policy`	Specify the SSL policy to use	`String`	yes	no
`causal_ clustering.array_ block_ id_ allocation_ size`	The size of the ID allocation requests Core servers will make when they run out of ARRAY_BLOCK IDs. Larger values mean less frequent requests but also result in more unused IDs (and unused disk space) in the event of a crash.	`String`	yes	no
`causal_ clustering.catch_ up_ client_ inactivity_ timeout`	The catch up protocol times out if the given duration elapses with no network activity. Every message received by the client from the server extends the time out duration.	`String`	yes	no
`causal_ clustering.catchup_ batch_ size`	The maximum batch size when catching up (in unit of entries)	`String`	yes	no
`causal_ clustering.cluster_ allow_ reads_ on_ followers`	Configure if the `dbms.cluster.routing.getServers()` procedure should include followers as read endpoints or return only read replicas. Note: if there are no read replicas in the cluster, followers are returned as read end points regardless the value of this setting. Defaults to true so that followers are available for read-only queries in a typical heterogeneous setup.	`String`	yes	no

Name	Description	Type	Read	Write
causal_ clustering.cluster_ routing_ ttl	How long drivers should cache the data from the dbms.cluster.routing.getServers() procedure.	String	yes	no
causal_ clustering.cluster_ topology_ refresh	Time between scanning the cluster to refresh current server's view of topology	String	yes	no
causal_ clustering.connect-randomly-to-server-group	Comma separated list of groups to be used by the connect-randomly-to-server-group selection strategy. The connect-randomly-to-server-group strategy is used if the list of strategies (causal_clustering.upstream_selection_strategy) includes the value connect-randomly-to-server-group.	String	yes	no
causal_ clustering.database	The name of the database being hosted by this server instance. This configuration setting may be safely ignored unless deploying a multicluster. Instances may be allocated to distinct sub-clusters by assigning them distinct database names using this setting. For instance if you had 6 instances you could form 2 sub-clusters by assigning half the database name "foo", half the name "bar". The setting value must match exactly between members of the same sub-cluster. This setting is a one-off: once an instance is configured with a database name it may not be changed in future without using neo4j-admin unbind.	String	yes	no
causal_ clustering.disable_ middleware_ logging	Prevents the network middleware from dumping its own logs. Defaults to true.	String	yes	no
causal_ clustering.discovery_ advertised_ address	Advertised cluster member discovery management communication.	String	yes	no
causal_ clustering.discovery_ listen_ address	Host and port to bind the cluster member discovery management communication.	String	yes	no
causal_ clustering.discovery_ type	Configure the discovery type used for cluster name resolution	String	yes	no
causal_ clustering.enable_ pre_ voting	Enable pre-voting extension to the Raft protocol (this is breaking and must match between the core cluster members)	String	yes	no
causal_ clustering.expected_ core_ cluster_ size	Expected number of Core machines in the cluster before startup	String	yes	no
causal_ clustering.global_ session_ tracker_ state_ size	The maximum file size before the global session tracker state file is rotated (in unit of entries)	String	yes	no
causal_ clustering.handshake_ timeout	Time out for protocol negotiation handshake	String	yes	no
causal_ clustering.id_ alloc_ state_ size	The maximum file size before the ID allocation file is rotated (in unit of entries)	String	yes	no
causal_ clustering.in_ flight_ cache.max_ bytes	The maximum number of bytes in the in-flight cache.	String	yes	no
causal_ clustering.in_ flight_ cache.max_ entries	The maximum number of entries in the in-flight cache.	String	yes	no
causal_ clustering.in_ flight_ cache.type	Type of in-flight cache.	String	yes	no
causal_ clustering.initial_ discovery_ members	A comma-separated list of other members of the cluster to join.	String	yes	no

Name	Description	Type	Read	Write
causal_ clustering.join _ catch_ up_ timeout	Time out for a new member to catch up	String	yes	no
causal_ clustering.labe l_ token_ id_ allocatio n_ size	The size of the ID allocation requests Core servers will make when they run out of LABEL_TOKEN IDs. Larger values mean less frequent requests but also result in more unused IDs (and unused disk space) in the event of a crash.	String	yes	no
causal_ clustering.labe l_ token_ name_ id_ all ocation_ size	The size of the ID allocation requests Core servers will make when they run out of LABEL_TOKEN_NAME IDs. Larger values mean less frequent requests but also result in more unused IDs (and unused disk space) in the event of a crash.	String	yes	no
causal_ clustering.last _ applied_ state_ size	The maximum file size before the storage file is rotated (in unit of entries)	String	yes	no
causal_ clustering.lead er_ election_ timeout	The time limit within which a new leader election will occur if no messages are received.	String	yes	no
causal_ clustering.load _ balancing.config	The configuration must be valid for the configured plugin and usually existsunder matching subkeys, e.g. ..config.server_policies.*This is just a top-level placeholder for the plugin-specific configuration.	String	yes	no
causal_ clustering.load _ balancing.plugin	The load balancing plugin to use.	String	yes	no
causal_ clustering.load _ balancing.shuffle	Enables shuffling of the returned load balancing result.	String	yes	no
causal_ clustering.log_ shipping_ max_ lag	The maximum lag allowed before log shipping pauses (in unit of entries)	String	yes	no
causal_ clustering.midd leware_ logging.level	The level of middleware logging	String	yes	no
causal_ clustering.mini mum_ core_ cluster_ siz e_ at_ formation	Minimum number of Core machines in the cluster at formation. The expected_core_cluster size setting is used when bootstrapping the cluster on first formation. A cluster will not form without the configured amount of cores and this should in general be configured to the full and fixed amount. When using multi-clustering (configuring multiple distinct database names across core hosts), this setting is used to define the minimum size of **each** sub-cluster at formation.	String	yes	no

Name	Description	Type	Read	Write
causal_ clustering.minimum_ core_ cluster_ size_ at_ runtime	Minimum number of Core machines required to be available at runtime. The consensus group size (core machines successfully voted into the Raft) can shrink and grow dynamically but bounded on the lower end at this number. The intention is in almost all cases for users to leave this setting alone. If you have 5 machines then you can survive failures down to 3 remaining, e.g. with 2 dead members. The three remaining can still vote another replacement member in successfully up to a total of 6 (2 of which are still dead) and then after this, one of the superfluous dead members will be immediately and automatically voted out (so you are left with 5 members in the consensus group, 1 of which is currently dead). Operationally you can now bring the last machine up by bringing in another replacement or repairing the dead one. When using multi-clustering (configuring multiple distinct database names across core hosts), this setting is used to define the minimum size of **each** sub-cluster at runtime.	String	yes	no
causal_ clustering.multi_ dc_ license	Enable multi-data center features. Requires appropriate licensing.	String	yes	no
causal_ clustering.neostore_ block_ id_ allocation_ size	The size of the ID allocation requests Core servers will make when they run out of NEOSTORE_BLOCK IDs. Larger values mean less frequent requests but also result in more unused IDs (and unused disk space) in the event of a crash.	String	yes	no
causal_ clustering.node_ id_ allocation_ size	The size of the ID allocation requests Core servers will make when they run out of NODE IDs. Larger values mean less frequent requests but also result in more unused IDs (and unused disk space) in the event of a crash.	String	yes	no
causal_ clustering.node_ labels_ id_ allocation_ size	The size of the ID allocation requests Core servers will make when they run out of NODE_LABELS IDs. Larger values mean less frequent requests but also result in more unused IDs (and unused disk space) in the event of a crash.	String	yes	no
causal_ clustering.property_ id_ allocation_ size	The size of the ID allocation requests Core servers will make when they run out of PROPERTY IDs. Larger values mean less frequent requests but also result in more unused IDs (and unused disk space) in the event of a crash.	String	yes	no
causal_ clustering.property_ key_ token_ id_ allocation_ size	The size of the ID allocation requests Core servers will make when they run out of PROPERTY_KEY_TOKEN IDs. Larger values mean less frequent requests but also result in more unused IDs (and unused disk space) in the event of a crash.	String	yes	no
causal_ clustering.property_ key_ token_ name_ id_ allocation_ size	The size of the ID allocation requests Core servers will make when they run out of PROPERTY_KEY_TOKEN_NAME IDs. Larger values mean less frequent requests but also result in more unused IDs (and unused disk space) in the event of a crash.	String	yes	no
causal_ clustering.protocol_ implementations.catchup	Catchup protocol implementation versions that this instance will allow in negotiation as a comma-separated list. Order is not relevant: the greatest value will be preferred. An empty list will allow all supported versions	String	yes	no

Name	Description	Type	Read	Write
causal_ clustering.protocol_ implementations.compression	Network compression algorithms that this instance will allow in negotiation as a comma-separated list. Listed in descending order of preference for incoming connections. An empty list implies no compression. For outgoing connections this merely specifies the allowed set of algorithms and the preference of the remote peer will be used for making the decision. Allowable values: [Gzip,Snappy,Snappy_validating,LZ4,LZ4_high_compression,LZ_validating,LZ4_high_compression_validating]	String	yes	no
causal_ clustering.protocol_ implementations.raft	Raft protocol implementation versions that this instance will allow in negotiation as a comma-separated list. Order is not relevant: the greatest value will be preferred. An empty list will allow all supported versions	String	yes	no
causal_ clustering.pull_ interval	Interval of pulling updates from cores.	String	yes	no
causal_ clustering.raft_ advertised_ address	Advertised hostname/IP address and port for the RAFT server.	String	yes	no
causal_ clustering.raft_ in_ queue_ max_ batch	Largest batch processed by RAFT	String	yes	no
causal_ clustering.raft_ in_ queue_ size	Size of the RAFT in queue	String	yes	no
causal_ clustering.raft_ listen_ address	Network interface and port for the RAFT server to listen on.	String	yes	no
causal_ clustering.raft_ log_ implementation	RAFT log implementation	String	yes	no
causal_ clustering.raft_ log_ prune_ strategy	RAFT log pruning strategy	String	yes	no
causal_ clustering.raft_ log_ pruning_ frequency	RAFT log pruning frequency	String	yes	no
causal_ clustering.raft_ log_ reader_ pool_ size	RAFT log reader pool size	String	yes	no
causal_ clustering.raft_ log_ rotation_ size	RAFT log rotation size	String	yes	no
causal_ clustering.raft_ membership_ state_ size	The maximum file size before the membership state file is rotated (in unit of entries)	String	yes	no
causal_ clustering.raft_ messages_ log_ enable	Enable or disable the dump of all network messages pertaining to the RAFT protocol	String	yes	no
causal_ clustering.raft_ messages_ log_ path	Path to RAFT messages log.	String	yes	no
causal_ clustering.raft_ term_ state_ size	The maximum file size before the term state file is rotated (in unit of entries)	String	yes	no
causal_ clustering.raft_ vote_ state_ size	The maximum file size before the vote state file is rotated (in unit of entries)	String	yes	no
causal_ clustering.read_ replica_ time_ to_ live	Time To Live before read replica is considered unavailable	String	yes	no
causal_ clustering.read_ replica_ transaction_ applier_ batch_ size	Maximum transaction batch size for read replicas when applying transactions pulled from core servers.	String	yes	no

Name	Description	Type	Read	Write
causal_ clustering.refuse_ to_ be_ leader	Prevents the current instance from volunteering to become Raft leader. Defaults to false, and should only be used in exceptional circumstances by expert users. Using this can result in reduced availability for the cluster.	String	yes	no
causal_ clustering.relationship_ group_ id_ allocation_ size	The size of the ID allocation requests Core servers will make when they run out of RELATIONSHIP_GROUP IDs. Larger values mean less frequent requests but also result in more unused IDs (and unused disk space) in the event of a crash.	String	yes	no
causal_ clustering.relationship_ id_ allocation_ size	The size of the ID allocation requests Core servers will make when they run out of RELATIONSHIP IDs. Larger values mean less frequent requests but also result in more unused IDs (and unused disk space) in the event of a crash.	String	yes	no
causal_ clustering.relationship_ type_ token_ id_ allocation_ size	The size of the ID allocation requests Core servers will make when they run out of RELATIONSHIP_TYPE_TOKEN IDs. Larger values mean less frequent requests but also result in more unused IDs (and unused disk space) in the event of a crash.	String	yes	no
causal_ clustering.relationship_ type_ token_ name_ id_ allocation_ size	The size of the ID allocation requests Core servers will make when they run out of RELATIONSHIP_TYPE_TOKEN_NAME IDs. Larger values mean less frequent requests but also result in more unused IDs (and unused disk space) in the event of a crash.	String	yes	no
causal_ clustering.replicated_ lock_ token_ state_ size	The maximum file size before the replicated lock token state file is rotated (in unit of entries)	String	yes	no
causal_ clustering.replication_ leader	The retry timeout for finding a leader for replication. Relevant during leader elections.	String	yes	no
causal_ clustering.replication_ retry_ timeout_ base	The initial timeout until replication is retried. The timeout will increase exponentially.	String	yes	no
causal_ clustering.replication_ retry_ timeout_ limit	The upper limit for the exponentially incremented retry timeout.	String	yes	no
causal_ clustering.replication_ total_ size_ limit	The maximum amount of data which can be in the replication stage concurrently.	String	yes	no
causal_ clustering.schema_ id_ allocation_ size	The size of the ID allocation requests Core servers will make when they run out of SCHEMA IDs. Larger values mean less frequent requests but also result in more unused IDs (and unused disk space) in the event of a crash.	String	yes	no
causal_ clustering.server_ groups	A list of group names for the server used when configuring load balancing and replication policies.	String	yes	no
causal_ clustering.ssl_ policy	Name of the SSL policy to be used by the clustering, as defined under the dbms.ssl.policy.* settings. If no policy is configured then the communication will not be secured.	String	yes	no
causal_ clustering.state_ machine_ apply_ max_ batch_ size	The maximum number of operations to be batched during applications of operations in the state machines	String	yes	no

Name	Description	Type	Read	Write
causal_clustering.state_machine_flush_window_size	The number of operations to be processed before the state machines flush to disk	String	yes	no
causal_clustering.store_copy_backoff_max_wait	Maximum backoff timeout for store copy requests	String	yes	no
causal_clustering.store_copy_max_retry_time_per_request	Maximum retry time per request during store copy. Regular store files and indexes are downloaded in separate requests during store copy. This configures the maximum time failed requests are allowed to resend.	String	yes	no
causal_clustering.string_block_id_allocation_size	The size of the ID allocation requests Core servers will make when they run out of STRING_BLOCK IDs. Larger values mean less frequent requests but also result in more unused IDs (and unused disk space) in the event of a crash.	String	yes	no
causal_clustering.transaction_advertised_address	Advertised hostname/IP address and port for the transaction shipping server.	String	yes	no
causal_clustering.transaction_listen_address	Network interface and port for the transaction shipping server to listen on. Please note that it is also possible to run the backup client against this port so always limit access to it via the firewall and configure an ssl policy.	String	yes	no
causal_clustering.unknown_address_logging_throttle	Throttle limit for logging unknown cluster member address	String	yes	no
causal_clustering.upstream_selection_strategy	An ordered list in descending preference of the strategy which read replicas use to choose the upstream server from which to pull transactional updates.	String	yes	no
causal_clustering.user_defined_upstream_strategy	Configuration of a user-defined upstream selection strategy. The user-defined strategy is used if the list of strategies (causal_clustering.upstream_selection_strategy) includes the value user_defined.	String	yes	no
cypher.default_language_version	Set this to specify the default parser (language version).	String	yes	no

Name	Description	Type	Read	Write
cypher.forbid_ exhaustive_ shortestpath	This setting is associated with performance optimization. Set this to true in situations where it is preferable to have any queries using the 'shortestPath' function terminate as soon as possible with no answer, rather than potentially running for a long time attempting to find an answer (even if there is no path to be found). For most queries, the 'shortestPath' algorithm will return the correct answer very quickly. However there are some cases where it is possible that the fast bidirectional breadth-first search algorithm will find no results even if they exist. This can happen when the predicates in the WHERE clause applied to 'shortestPath' cannot be applied to each step of the traversal, and can only be applied to the entire path. When the query planner detects these special cases, it will plan to perform an exhaustive depth-first search if the fast algorithm finds no paths. However, the exhaustive search may be orders of magnitude slower than the fast algorithm. If it is critical that queries terminate as soon as possible, it is recommended that this option be set to true, which means that Neo4j will never consider using the exhaustive search for shortestPath queries. However, please note that if no paths are found, an error will be thrown at run time, which will need to be handled by the application.	String	yes	no
cypher.forbid_ shortestpath_ common_ nodes	This setting is associated with performance optimization. The shortest path algorithm does not work when the start and end nodes are the same. With this setting set to false no path will be returned when that happens. The default value of true will instead throw an exception. This can happen if you perform a shortestPath search after a cartesian product that might have the same start and end nodes for some of the rows passed to shortestPath. If it is preferable to not experience this exception, and acceptable for results to be missing for those rows, then set this to false. If you cannot accept missing results, and really want the shortestPath between two common nodes, then re-write the query using a standard Cypher variable length pattern expression followed by ordering by path length and limiting to one result.	String	yes	no
cypher.hints_ error	Set this to specify the behavior when Cypher planner or runtime hints cannot be fulfilled. If true, then non-conformance will result in an error, otherwise only a warning is generated.	String	yes	no
cypher.min_ replan_ interval	The minimum time between possible cypher query replanning events. After this time, the graph statistics will be evaluated, and if they have changed by more than the value set by cypher.statistics_divergence_threshold, the query will be replanned. If the statistics have not changed sufficiently, the same interval will need to pass before the statistics will be evaluated again. Each time they are evaluated, the divergence threshold will be reduced slightly until it reaches 10% after 7h, so that even moderately changing databases will see query replanning after a sufficiently long time interval.	String	yes	no
cypher.planner	Set this to specify the default planner for the default language version.	String	yes	no

Name	Description	Type	Read	Write
`cypher.statistics_ divergence_ threshold`	The threshold when a plan is considered stale. If any of the underlying statistics used to create the plan have changed more than this value, the plan will be considered stale and will be replanned. Change is calculated as abs(a-b)/max(a,b). This means that a value of 0.75 requires the database to approximately quadruple in size. A value of 0 means replan as soon as possible, with the soonest being defined by the cypher.min_replan_interval which defaults to 10s. After this interval the divergence threshold will slowly start to decline, reaching 10% after about 7h. This will ensure that long running databases will still get query replanning on even modest changes, while not replanning frequently unless the changes are very large.	`String`	yes	no
`db.temporal.timezone`	Database timezone for temporal functions. All Time and DateTime values that are created without an explicit timezone will use this configured default timezone.	`String`	yes	no
`dbms.active_ database`	Name of the database to load	`String`	yes	no
`dbms.allow_ format_ migration`	Whether to allow a store upgrade in case the current version of the database starts against an older store version. Setting this to `true` does not guarantee successful upgrade, it just allows an upgrade to be performed.	`String`	yes	no
`dbms.allow_ upgrade`	Whether to allow an upgrade in case the current version of the database starts against an older version.	`String`	yes	no
`dbms.auto_ index.nodes. enabled`	Controls the auto indexing feature for nodes. Setting it to `false` shuts it down, while `true` enables it by default for properties listed in the dbms.auto_index.nodes.keys setting.	`String`	yes	no
`dbms.auto_ index.nodes. keys`	A list of property names (comma separated) that will be indexed by default. This applies to *nodes* only.	`String`	yes	no
`dbms.auto_ index.relationships.enabled`	Controls the auto indexing feature for relationships. Setting it to `false` shuts it down, while `true` enables it by default for properties listed in the dbms.auto_index.relationships.keys setting.	`String`	yes	no
`dbms.auto_ index.relationships.keys`	A list of property names (comma separated) that will be indexed by default. This applies to *relationships* only.	`String`	yes	no
`dbms.backup.address`	Listening server for online backups. The protocol running varies depending on deployment. In a Causal Clustering environment this is the same protocol that runs on causal_clustering.transaction_listen_address.	`String`	yes	no
`dbms.backup.enabled`	Enable support for running online backups	`String`	yes	no
`dbms.backup.ssl_ policy`	Name of the SSL policy to be used by backup, as defined under the dbms.ssl.policy.* settings. If no policy is configured then the communication will not be secured.	`String`	yes	no

Name	Description	Type	Read	Write
dbms.checkpoint	Configures the general policy for when check-points should occur. The default policy is the 'periodic' check-point policy, as specified by the 'dbms.checkpoint.interval.tx' and 'dbms.checkpoint.interval.time' settings. The Neo4j Enterprise Edition provides two alternative policies: The first is the 'continuous' check-point policy, which will ignore those settings and run the check-point process all the time. The second is the 'volumetric' check-point policy, which makes a best-effort at check-pointing often enough so that the database doesn't get too far behind on deleting old transaction logs in accordance with the 'dbms.tx_log.rotation.retention_policy' setting.	String	yes	no
dbms.checkpoint.interval.time	Configures the time interval between check-points. The database will not check-point more often than this (unless check pointing is triggered by a different event), but might check-point less often than this interval, if performing a check-point takes longer time than the configured interval. A check-point is a point in the transaction logs, from which recovery would start from. Longer check-point intervals typically means that recovery will take longer to complete in case of a crash. On the other hand, a longer check-point interval can also reduce the I/O load that the database places on the system, as each check-point implies a flushing and forcing of all the store files.	String	yes	no
dbms.checkpoint.interval.tx	Configures the transaction interval between check-points. The database will not check-point more often than this (unless check pointing is triggered by a different event), but might check-point less often than this interval, if performing a check-point takes longer time than the configured interval. A check-point is a point in the transaction logs, from which recovery would start from. Longer check-point intervals typically means that recovery will take longer to complete in case of a crash. On the other hand, a longer check-point interval can also reduce the I/O load that the database places on the system, as each check-point implies a flushing and forcing of all the store files. The default is '100000' for a check-point every 100000 transactions.	String	yes	no
dbms.checkpoint.iops.limit	Limit the number of IOs the background checkpoint process will consume per second. This setting is advisory, is ignored in Neo4j Community Edition, and is followed to best effort in Enterprise Edition. An IO is in this case a 8 KiB (mostly sequential) write. Limiting the write IO in this way will leave more bandwidth in the IO subsystem to service random-read IOs, which is important for the response time of queries when the database cannot fit entirely in memory. The only drawback of this setting is that longer checkpoint times may lead to slightly longer recovery times in case of a database or system crash. A lower number means lower IO pressure, and consequently longer checkpoint times. The configuration can also be commented out to remove the limitation entirely, and let the checkpointer flush data as fast as the hardware will go. Set this to -1 to disable the IOPS limit.	String	yes	no

Name	Description	Type	Read	Write
dbms.config.strict_ validation	A strict configuration validation will prevent the database from starting up if unknown configuration options are specified in the neo4j settings namespace (such as dbms., ha., cypher., etc). This is currently false by default but will be true by default in 4.0.	String	yes	no
dbms.connector.bolt.advertised_ address	Advertised address for this connector.	String	yes	no
dbms.connector.bolt.enabled	Enable this connector.	String	yes	no
dbms.connector.bolt.listen_ address	Address the connector should bind to.	String	yes	no
dbms.connector.bolt.type	Connector type. This setting is deprecated and its value will instead be inferred from the name of the connector.	String	yes	no
dbms.connector.http.advertised_ address	Advertised address for this connector.	String	yes	no
dbms.connector.http.enabled	Enable this connector.	String	yes	no
dbms.connector.http.listen_ address	Address the connector should bind to.	String	yes	no
dbms.connector.http.type	Connector type. This setting is deprecated and its value will instead be inferred from the name of the connector.	String	yes	no
dbms.connectors.default_ advertised_ address	Default hostname or IP address the server uses to advertise itself to its connectors. To advertise a specific hostname or IP address for a specific connector, specify the advertised_address property for the specific connector.	String	yes	no
dbms.connectors.default_ listen_ address	Default network interface to listen for incoming connections. To listen for connections on all interfaces, use "0.0.0.0". To bind specific connectors to a specific network interfaces, specify the listen_address properties for the specific connector.	String	yes	no
dbms.db.timezone	Database timezone. Among other things, this setting influences which timezone the logs and monitoring procedures use.	String	yes	no
dbms.directories.certificates	Directory for storing certificates to be used by Neo4j for TLS connections	String	yes	no
dbms.directories.data	Path of the data directory. You must not configure more than one Neo4j installation to use the same data directory.	String	yes	no
dbms.directories.import	Sets the root directory for file URLs used with the Cypher `LOAD CSV` clause. This must be set to a single directory, restricting access to only those files within that directory and its subdirectories.	String	yes	no
dbms.directories.logs	Path of the logs directory.	String	yes	no
dbms.directories.metrics	The target location of the CSV files: a path to a directory wherein a CSV file per reported field will be written.	String	yes	no
dbms.directories.plugins	Location of the database plugin directory. Compiled Java JAR files that contain database procedures will be loaded if they are placed in this directory.	String	yes	no
dbms.directories.tx_ log	Location where Neo4j keeps the logical transaction logs.	String	yes	no

Name	Description	Type	Read	Write
dbms.ids.reuse.types.override	Specified names of id types (comma separated) that should be reused. Currently only 'node' and 'relationship' types are supported.	String	yes	no
dbms.import.csv.legacy_quote_escaping	Selects whether to conform to the standard https://tools.ietf.org/html/rfc4180 for interpreting escaped quotation characters in CSV files loaded using LOAD CSV. Setting this to false will use the standard, interpreting repeated quotes """" as a single in-lined quote, while true will use the legacy convention originally supported in Neo4j 3.0 and 3.1, allowing a backslash to include quotes in-lined in fields.	String	yes	no
dbms.index.default_schema_provider	Index provider to use for newly created schema indexes. An index provider may store different value types in separate physical indexes. lucene-1.0: Store spatial and temporal value types in native indexes, remaining value types in a Lucene index. lucene+native-1.0: Store numbers in a native index and remaining value types like lucene-1.0. This improves read and write performance for non-composite indexed numbers. lucene+native-2.0: Store strings in a native index and remaining value types like lucene+native-1.0. This improves write performance for non-composite indexed strings. This version of the native string index has a value limit of 4047B, such that byte-representation of a string to index cannot be larger than that limit, or the transaction trying to index such a value will fail. This version of the native string index also has reduced performance for CONTAINS and ENDS WITH queries, due to resorting to index scan+filter internally. Native indexes generally has these benefits over Lucene: - Faster writes - Less garbage and heap presence - Less CPU resources per operation - Controllable memory usage, due to being bound by the page cache	String	yes	no
dbms.index_sampling.background_enabled	Enable or disable background index sampling	String	yes	no
dbms.index_sampling.buffer_size	Size of buffer used by index sampling. This configuration setting is no longer applicable as from Neo4j 3.0.3. Please use dbms.index_sampling.sample_size_limit instead.	String	yes	no
dbms.index_sampling.sample_size_limit	Index sampling chunk size limit	String	yes	no
dbms.index_sampling.update_percentage	Percentage of index updates of total index size required before sampling of a given index is triggered	String	yes	no
dbms.index_searcher_cache_size	The maximum number of open Lucene index searchers.	String	yes	no
dbms.jvm.additional	Additional JVM arguments.	String	yes	no
dbms.label_index	Backend to use for label -□ nodes index	String	yes	no
dbms.lock.acquisition.timeout	The maximum time interval within which lock should be acquired.	String	yes	no
dbms.logs.debug.level	Debug log level threshold.	String	yes	no
dbms.logs.debug.path	Path to the debug log file.	String	yes	no
dbms.logs.debug.rotation.delay	Minimum time interval after last rotation of the debug log before it may be rotated again.	String	yes	no

Name	Description	Type	Read	Write
dbms.logs.debug.rotation.keep_number	Maximum number of history files for the debug log.	String	yes	no
dbms.logs.debug.rotation.size	Threshold for rotation of the debug log.	String	yes	no
dbms.logs.query.allocation_logging_enabled	Log allocated bytes for the executed queries being logged. The logged number is cumulative over the duration of the query, i.e. for memory intense or long-running queries the value may be larger than the current memory allocation.	String	yes	no
dbms.logs.query.enabled	Log executed queries that take longer than the configured threshold, dbms.logs.query.threshold. Log entries are by default written to the file *query.log* located in the Logs directory. For location of the Logs directory, see [file-locations]. This feature is available in the Neo4j Enterprise Edition.	String	yes	no
dbms.logs.query.page_logging_enabled	Log page hits and page faults for the executed queries being logged.	String	yes	no
dbms.logs.query.parameter_logging_enabled	Log parameters for the executed queries being logged.	String	yes	no
dbms.logs.query.path	Path to the query log file.	String	yes	no
dbms.logs.query.rotation.keep_number	Maximum number of history files for the query log.	String	yes	no
dbms.logs.query.rotation.size	The file size in bytes at which the query log will auto-rotate. If set to zero then no rotation will occur. Accepts a binary suffix `k`, `m` or `g`.	String	yes	no
dbms.logs.query.runtime_logging_enabled	Logs which runtime that was used to run the query	String	yes	no
dbms.logs.query.threshold	If the execution of query takes more time than this threshold, the query is logged - provided query logging is enabled. Defaults to 0 seconds, that is all queries are logged.	String	yes	no
dbms.logs.query.time_logging_enabled	Log detailed time information for the executed queries being logged.	String	yes	no
dbms.logs.security.level	Security log level threshold.	String	yes	no
dbms.logs.security.path	Path to the security log file.	String	yes	no
dbms.logs.security.rotation.delay	Minimum time interval after last rotation of the security log before it may be rotated again.	String	yes	no
dbms.logs.security.rotation.keep_number	Maximum number of history files for the security log.	String	yes	no
dbms.logs.security.rotation.size	Threshold for rotation of the security log.	String	yes	no
dbms.logs.timezone	Database logs timezone.	String	yes	no
dbms.memory.heap.initial_size	Initial heap size. By default it is calculated based on available system resources.	String	yes	no
dbms.memory.heap.max_size	Maximum heap size. By default it is calculated based on available system resources.	String	yes	no

Name	Description	Type	Read	Write
dbms.memory.pagecache.size	The amount of memory to use for mapping the store files, in bytes (or kilobytes with the 'k' suffix, megabytes with 'm' and gigabytes with 'g'). If Neo4j is running on a dedicated server, then it is generally recommended to leave about 2-4 gigabytes for the operating system, give the JVM enough heap to hold all your transaction state and query context, and then leave the rest for the page cache. If no page cache memory is configured, then a heuristic setting is computed based on available system resources.	String	yes	no
dbms.memory.pagecache.swapper	Specify which page swapper to use for doing paged IO. This is only used when integrating with proprietary storage technology.	String	yes	no
dbms.mode	Configure the operating mode of the database — 'SINGLE' for stand-alone operation, 'HA' for operating as a member in an HA cluster, 'ARBITER' for a cluster member with no database in an HA cluster, 'CORE' for operating as a core member of a Causal Cluster, or 'READ_REPLICA' for operating as a read replica member of a Causal Cluster.	String	yes	no
dbms.procedures.kill_query_verbose	Specifies whether or not dbms.killQueries produces a verbose output, with information about which queries were not found	String	yes	no
dbms.query_cache_size	The number of Cypher query execution plans that are cached.	String	yes	no
dbms.read_only	Only allow read operations from this Neo4j instance. This mode still requires write access to the directory for lock purposes.	String	yes	no
dbms.record_format	Database record format. Valid values: standard, high_limit. The high_limit format is available for Enterprise Edition only. It is required if you have a graph that is larger than 34 billion nodes, 34 billion relationships, or 68 billion properties. A change of the record format is irreversible. Certain operations may suffer from a performance penalty of up to 10%, which is why this format is not switched on by default.	String	yes	no
dbms.relationship_grouping_threshold	Relationship count threshold for considering a node to be dense	String	yes	no
dbms.security.allow_csv_import_from_file_urls	Determines if Cypher will allow using file URLs when loading data using LOAD CSV. Setting this value to false will cause Neo4j to fail LOAD CSV clauses that load data from the file system.	String	yes	no
dbms.security.auth_cache_max_capacity	The maximum capacity for authentication and authorization caches (respectively).	String	yes	no
dbms.security.auth_cache_ttl	The time to live (TTL) for cached authentication and authorization info when using external auth providers (LDAP or plugin). Setting the TTL to 0 will disable auth caching. Disabling caching while using the LDAP auth provider requires the use of an LDAP system account for resolving authorization information.	String	yes	no
dbms.security.auth_cache_use_ttl	Enable time-based eviction of the authentication and authorization info cache for external auth providers (LDAP or plugin). Disabling this setting will make the cache live forever and only be evicted when dbms.security.auth_cache_max_capacity is exceeded.	String	yes	no

Name	Description	Type	Read	Write
dbms.security.auth_ enabled	Enable auth requirement to access Neo4j.	String	yes	no
dbms.security.auth_ provider	The authentication and authorization provider that contains both the users and roles. This can be one of the built-in native or ldap providers, or it can be an externally provided plugin, with a custom name prefixed by plugin-, i.e. plugin-<AUTH_PROVIDER_NAME>.	String	yes	no
dbms.security.auth_ providers	A list of security authentication and authorization providers containing the users and roles. They will be queried in the given order when login is attempted.	String	yes	no
dbms.security.causal_ clustering_ status_ auth_ enabled	Require authorization for access to the Causal Clustering status endpoints.	String	yes	no
dbms.security.ha_ status_ auth_ enabled	Require authorization for access to the HA status endpoints.	String	yes	no
dbms.security.ldap.authentication.cache_ enabled	Determines if the result of authentication via the LDAP server should be cached or not. Caching is used to limit the number of LDAP requests that have to be made over the network for users that have already been authenticated successfully. A user can be authenticated against an existing cache entry (instead of via an LDAP server) as long as it is alive (see dbms.security.auth_cache_ttl). An important consequence of setting this to true is that Neo4j then needs to cache a hashed version of the credentials in order to perform credentials matching. This hashing is done using a cryptographic hash function together with a random salt. Preferably a conscious decision should be made if this method is considered acceptable by the security standards of the organization in which this Neo4j instance is deployed.	String	yes	no
dbms.security.ldap.authentication.mechanism	LDAP authentication mechanism. This is one of simple or a SASL mechanism supported by JNDI, for example DIGEST-MD5. simple is basic username and password authentication and SASL is used for more advanced mechanisms. See RFC 2251 LDAPv3 documentation for more details.	String	yes	no
dbms.security.ldap.authentication.use_ samaccountname	Perform authentication with sAMAccountName instead of DN. Using this setting requires dbms.security.ldap.authorization.system_username and dbms.security.ldap.authorization.system_password to be used since there is no way to log in through ldap directly with the sAMAccountName, instead the login name will be resolved to a DN that will be used to log in with.	String	yes	no
dbms.security.ldap.authentication.user_ dn_ template	LDAP user DN template. An LDAP object is referenced by its distinguished name (DN), and a user DN is an LDAP fully-qualified unique user identifier. This setting is used to generate an LDAP DN that conforms with the LDAP directory's schema from the user principal that is submitted with the authentication token when logging in. The special token {0} is a placeholder where the user principal will be substituted into the DN string.	String	yes	no
dbms.security.ldap.authentication_ enabled	Enable authentication via settings configurable LDAP authentication provider.	String	yes	no

Name	Description	Type	Read	Write
dbms.security.ldap.authorization.group_membership_attributes	A list of attribute names on a user object that contains groups to be used for mapping to roles when LDAP authorization is enabled.	String	yes	no
dbms.security.ldap.authorization.group_to_role_mapping	An authorization mapping from LDAP group names to Neo4j role names. The map should be formatted as a semicolon separated list of key-value pairs, where the key is the LDAP group name and the value is a comma separated list of corresponding role names. For example: group1=role1;group2=role2;group3=role3,role4,role5 You could also use whitespaces and quotes around group names to make this mapping more readable, for example: ---- dbms.security.ldap.authorization.group_to_role_mapping=\ "cn=Neo4j Read Only,cn=users,dc=example,dc=com" = reader; \ "cn=Neo4j Read-Write,cn=users,dc=example,dc=com" = publisher; \ "cn=Neo4j Schema Manager,cn=users,dc=example,dc=com" = architect; \ "cn=Neo4j Administrator,cn=users,dc=example,dc=com" = admin ----	String	yes	no
dbms.security.ldap.authorization.system_password	An LDAP system account password to use for authorization searches when dbms.security.ldap.authorization.use_system_account is true.	String	yes	no
dbms.security.ldap.authorization.system_username	An LDAP system account username to use for authorization searches when dbms.security.ldap.authorization.use_system_account is true. Note that the dbms.security.ldap.authentication.user_dn_template will not be applied to this username, so you may have to specify a full DN.	String	yes	no
dbms.security.ldap.authorization.use_system_account	Perform LDAP search for authorization info using a system account instead of the user's own account. If this is set to false (default), the search for group membership will be performed directly after authentication using the LDAP context bound with the user's own account. The mapped roles will be cached for the duration of dbms.security.auth_cache_ttl, and then expire, requiring re-authentication. To avoid frequently having to re-authenticate sessions you may want to set a relatively long auth cache expiration time together with this option. NOTE: This option will only work if the users are permitted to search for their own group membership attributes in the directory. If this is set to true, the search will be performed using a special system account user with read access to all the users in the directory. You need to specify the username and password using the settings dbms.security.ldap.authorization.system_username and dbms.security.ldap.authorization.system_password with this option. Note that this account only needs read access to the relevant parts of the LDAP directory and does not need to have access rights to Neo4j, or any other systems.	String	yes	no
dbms.security.ldap.authorization.user_search_base	The name of the base object or named context to search for user objects when LDAP authorization is enabled. A common case is that this matches the last part of dbms.security.ldap.authentication.user_dn_template.	String	yes	no

Name	Description	Type	Read	Write
dbms.security.ldap.authorization.user_search_filter	The LDAP search filter to search for a user principal when LDAP authorization is enabled. The filter should contain the placeholder token {0} which will be substituted for the user principal.	String	yes	no
dbms.security.ldap.authorization_enabled	Enable authorization via settings configurable LDAP authorization provider.	String	yes	no
dbms.security.ldap.connection_timeout	The timeout for establishing an LDAP connection. If a connection with the LDAP server cannot be established within the given time the attempt is aborted. A value of 0 means to use the network protocol's (i.e., TCP's) timeout value.	String	yes	no
dbms.security.ldap.host	URL of LDAP server to use for authentication and authorization. The format of the setting is <protocol>://<hostname>:<port>, where hostname is the only required field. The supported values for protocol are ldap (default) and ldaps. The default port for ldap is 389 and for ldaps 636. For example: ldaps://ldap.example.com:10389. You may want to consider using STARTTLS (dbms.security.ldap.use_starttls) instead of LDAPS for secure connections, in which case the correct protocol is ldap.	String	yes	no
dbms.security.ldap.read_timeout	The timeout for an LDAP read request (i.e. search). If the LDAP server does not respond within the given time the request will be aborted. A value of 0 means wait for a response indefinitely.	String	yes	no
dbms.security.ldap.referral	The LDAP referral behavior when creating a connection. This is one of follow, ignore or throw. * follow automatically follows any referrals * ignore ignores any referrals * throw throws an exception, which will lead to authentication failure	String	yes	no
dbms.security.ldap.use_starttls	Use secure communication with the LDAP server using opportunistic TLS. First an initial insecure connection will be made with the LDAP server, and a STARTTLS command will be issued to negotiate an upgrade of the connection to TLS before initiating authentication.	String	yes	no
dbms.security.log_successful_authentication	Set to log successful authentication events to the security log. If this is set to false only failed authentication events will be logged, which could be useful if you find that the successful events spam the logs too much, and you do not require full auditing capability.	String	yes	no
dbms.security.native.authentication_enabled	Enable authentication via native authentication provider.	String	yes	no
dbms.security.native.authorization_enabled	Enable authorization via native authorization provider.	String	yes	no
dbms.security.plugin.authentication_enabled	Enable authentication via plugin authentication providers.	String	yes	no
dbms.security.plugin.authorization_enabled	Enable authorization via plugin authorization providers.	String	yes	no

Name	Description	Type	Read	Write
dbms.security.procedures.default_ allowed	The default role that can execute all procedures and user-defined functions that are not covered by the dbms.security.procedures.roles setting. If the dbms.security.procedures.default_allowed setting is the empty string (default), procedures will be executed according to the same security rules as normal Cypher statements.	String	yes	no
dbms.security.procedures.roles	This provides a finer level of control over which roles can execute procedures than the dbms.security.procedures.default_allowed setting. For example: dbms.security.procedures.roles=apoc.convert.*:reader;apoc.load.json*:writer;apoc.trigger.add:TriggerHappy will allow the role reader to execute all procedures in the apoc.convert namespace, the role writer to execute all procedures in the apoc.load namespace that starts with json and the role TriggerHappy to execute the specific procedure apoc.trigger.add. Procedures not matching any of these patterns will be subject to the dbms.security.procedures.default_allowed setting.	String	yes	no
dbms.security.procedures.unrestricted	A list of procedures and user defined functions (comma separated) that are allowed full access to the database. The list may contain both fully-qualified procedure names, and partial names with the wildcard '*'. Note that this enables these procedures to bypass security. Use with caution.	String	yes	no
dbms.security.procedures.whitelist	A list of procedures (comma separated) that are to be loaded. The list may contain both fully-qualified procedure names, and partial names with the wildcard '*'. If this setting is left empty no procedures will be loaded.	String	yes	no
dbms.security.property_ level.blacklist	An authorization mapping for property level access for roles. The map should be formatted as a semicolon separated list of key-value pairs, where the key is the role name and the value is a comma separated list of blacklisted properties. For example: role1=prop1;role2=prop2;role3=prop3,prop4,prop5 You could also use whitespaces and quotes around group names to make this mapping more readable, for example: dbms.security.property_level.blacklist=\ "role1" = ssn; \ "role2" = ssn,income; \	String	yes	no
dbms.security.property_ level.enabled	Set to true to enable property level security.	String	yes	no
dbms.shutdown_ transaction_ end_ timeout	The maximum amount of time to wait for running transactions to complete before allowing initiated database shutdown to continue	String	yes	no
dbms.ssl.policy.<policy name>.allow_ key_ generation	Allows the generation of a private key and associated self-signed certificate. Only performed when both objects cannot be found.	String	yes	no
dbms.ssl.policy.<policy name>.base_ directory	The mandatory base directory for cryptographic objects of this policy. It is also possible to override each individual configuration with absolute paths.	String	yes	no
dbms.ssl.policy.<policy name>.ciphers	Restrict allowed ciphers.	String	yes	no

Name	Description	Type	Read	Write
dbms.ssl.policy.<policy name>.client_ auth	Client authentication stance.	String	yes	no
dbms.ssl.policy.<policy name>.private_ key	Private PKCS#8 key in PEM format.	String	yes	no
dbms.ssl.policy.<policy name>.private_ key_ password	The password for the private key.	String	yes	no
dbms.ssl.policy.<policy name>.public_ certificate	X.509 certificate (chain) of this server in PEM format.	String	yes	no
dbms.ssl.policy.<policy name>.revoked_ dir	Path to directory of CRLs (Certificate Revocation Lists) in PEM format.	String	yes	no
dbms.ssl.policy.<policy name>.tls_ versions	Restrict allowed TLS protocol versions.	String	yes	no
dbms.ssl.policy.<policy name>.trust_ all	Makes this policy trust all remote parties. Enabling this is not recommended and the trusted directory will be ignored.	String	yes	no
dbms.ssl.policy.<policy name>.trusted_ dir	Path to directory of X.509 certificates in PEM format for trusted parties.	String	yes	no
dbms.track_ query_ allocation	Enables or disables tracking of how many bytes are allocated by the execution of a query.	String	yes	no
dbms.track_ query_ cpu_ time	Enables or disables tracking of how much time a query spends actively executing on the CPU.	String	yes	no
dbms.transaction.bookmark_ ready_ timeout	The maximum amount of time to wait for the database state represented by the bookmark.	String	yes	no
dbms.transaction.monitor.check.interval	Configures the time interval between transaction monitor checks. Determines how often monitor thread will check transaction for timeout.	String	yes	no
dbms.transaction.timeout	The maximum time interval of a transaction within which it should be completed.	String	yes	no
dbms.tx_ log.rotation.retention_ policy	Make Neo4j keep the logical transaction logs for being able to backup the database. Can be used for specifying the threshold to prune logical logs after. For example "10 days" will prune logical logs that only contains transactions older than 10 days from the current time, or "100k txs" will keep the 100k latest transactions and prune any older transactions.	String	yes	no
dbms.tx_ log.rotation.size	Specifies at which file size the logical log will auto-rotate. Minimum accepted value is 1M.	String	yes	no
dbms.udc.enabled	Enable the UDC extension.	String	yes	no
dbms.windows_ service_ name	Name of the Windows Service.	String	yes	no
ha.allow_ init_ cluster	Whether to allow this instance to create a cluster if unable to join.	String	yes	no
ha.branched_ data_ copying_ strategy	Strategy for how to order handling of branched data on slaves and copying of the store from the master. The default is copy_then_branch, which, when combined with the keep_last or keep_none branch handling strategies results in a safer branching strategy, as there is always a store present so store failure to copy a store (for example, because of network failure) does not leave the instance without a store.	String	yes	no

Name	Description	Type	Read	Write
ha.branched_ data_ poli cy	Policy for how to handle branched data.	String	yes	no
ha.broadcast_ timeout	Timeout for broadcasting values in cluster. Must consider end-to-end duration of Paxos algorithm. This value is the default value for the ha.join_timeout and ha.leave_timeout settings.	String	yes	no
ha.configuration_ timeo ut	Timeout for waiting for configuration from an existing cluster member during cluster join.	String	yes	no
ha.data_ chunk_ size	Max size of the data chunks that flows between master and slaves in HA. Bigger size may increase throughput, but may also be more sensitive to variations in bandwidth, whereas lower size increases tolerance for bandwidth variations.	String	yes	no
ha.default_ timeout	Default timeout used for clustering timeouts. Override specific timeout settings with proper values if necessary. This value is the default value for the ha.heartbeat_interval, ha.paxos_timeout and ha.learn_timeout settings.	String	yes	no
ha.election_ timeout	Timeout for waiting for other members to finish a role election. Defaults to ha.paxos_timeout.	String	yes	no
ha.heartbeat_ interval	How often heartbeat messages should be sent. Defaults to ha.default_timeout.	String	yes	no
ha.heartbeat_ timeout	How long to wait for heartbeats from other instances before marking them as suspects for failure. This value reflects considerations of network latency, expected duration of garbage collection pauses and other factors that can delay message sending and processing. Larger values will result in more stable masters but also will result in longer waits before a failover in case of master failure. This value should not be set to less than twice the ha.heartbeat_interval value otherwise there is a high risk of frequent master switches and possibly branched data occurrence.	String	yes	no
ha.host.coordination	Host and port to bind the cluster management communication.	String	yes	no
ha.host.data	Hostname and port to bind the HA server.	String	yes	no
ha.initial_ hosts	A comma-separated list of other members of the cluster to join.	String	yes	no
ha.internal_ role_ swit ch_ timeout	Timeout for waiting for internal conditions during state switch, like for transactions to complete, before switching to master or slave.	String	yes	no
ha.join_ timeout	Timeout for joining a cluster. Defaults to ha.broadcast_timeout. Note that if the timeout expires during cluster formation, the operator may have to restart the instance or instances.	String	yes	no
ha.learn_ timeout	Timeout for learning values. Defaults to ha.default_timeout.	String	yes	no
ha.leave_ timeout	Timeout for waiting for cluster leave to finish. Defaults to ha.broadcast_timeout.	String	yes	no
ha.max_ acceptors	Maximum number of servers to involve when agreeing to membership changes. In very large clusters, the probability of half the cluster failing is low, but protecting against any arbitrary half failing is expensive. Therefore you may wish to set this parameter to a value less than the cluster size.	String	yes	no

Name	Description	Type	Read	Write
ha.max_ channels_ per_ slave	Maximum number of connections a slave can have to the master.	String	yes	no
ha.paxos_ timeout	Default value for all Paxos timeouts. This setting controls the default value for the ha.phase1_timeout, ha.phase2_timeout and ha.election_timeout settings. If it is not given a value it defaults to ha.default_timeout and will implicitly change if ha.default_timeout changes. This is an advanced parameter which should only be changed if specifically advised by Neo4j Professional Services.	String	yes	no
ha.phase1_ timeout	Timeout for Paxos phase 1. If it is not given a value it defaults to ha.paxos_timeout and will implicitly change if ha.paxos_timeout changes. This is an advanced parameter which should only be changed if specifically advised by Neo4j Professional Services.	String	yes	no
ha.phase2_ timeout	Timeout for Paxos phase 2. If it is not given a value it defaults to ha.paxos_timeout and will implicitly change if ha.paxos_timeout changes. This is an advanced parameter which should only be changed if specifically advised by Neo4j Professional Services.	String	yes	no
ha.pull_ batch_ size	Size of batches of transactions applied on slaves when pulling from master	String	yes	no
ha.pull_ interval	Interval of pulling updates from master.	String	yes	no
ha.role_ switch_ timeout	Timeout for request threads waiting for instance to become master or slave.	String	yes	no
ha.server_ id	Id for a cluster instance. Must be unique within the cluster.	String	yes	no
ha.slave_ lock_ timeout	Timeout for taking remote (write) locks on slaves. Defaults to ha.slave_read_timeout.	String	yes	no
ha.slave_ only	Whether this instance should only participate as slave in cluster. If set to true, it will never be elected as master.	String	yes	no
ha.slave_ read_ timeout	How long a slave will wait for response from master before giving up.	String	yes	no
ha.strict_ initial_ hosts	Configuration attribute	String	yes	no
ha.tx_ push_ factor	The amount of slaves the master will ask to replicate a committed transaction.	String	yes	no
ha.tx_ push_ strategy	Push strategy of a transaction to a slave during commit.	String	yes	no
hazelcast.license_ key	Hazelcast license key	String	yes	no
metrics.bolt.messages.enabled	Enable reporting metrics about Bolt Protocol message processing.	String	yes	no
metrics.csv.enabled	Set to true to enable exporting metrics to CSV files	String	yes	no
metrics.csv.interval	The reporting interval for the CSV files. That is, how often new rows with numbers are appended to the CSV files.	String	yes	no
metrics.csv.rotation.keep_ number	Maximum number of history files for the csv files.	String	yes	no
metrics.csv.rotation.size	The file size in bytes at which the csv files will auto-rotate. If set to zero then no rotation will occur. Accepts a binary suffix k, m or g.	String	yes	no

Name	Description	Type	Read	Write
metrics.cypher.replanning.enabled	Enable reporting metrics about number of occurred replanning events.	String	yes	no
metrics.enabled	The default enablement value for all the supported metrics. Set this to false to turn off all metrics by default. The individual settings can then be used to selectively re-enable specific metrics.	String	yes	no
metrics.graphite.enabled	Set to true to enable exporting metrics to Graphite.	String	yes	no
metrics.graphite.interval	The reporting interval for Graphite. That is, how often to send updated metrics to Graphite.	String	yes	no
metrics.graphite.server	The hostname or IP address of the Graphite server	String	yes	no
metrics.jvm.buffers.enabled	Enable reporting metrics about the buffer pools.	String	yes	no
metrics.jvm.gc.enabled	Enable reporting metrics about the duration of garbage collections	String	yes	no
metrics.jvm.memory.enabled	Enable reporting metrics about the memory usage.	String	yes	no
metrics.jvm.threads.enabled	Enable reporting metrics about the current number of threads running.	String	yes	no
metrics.neo4j.causal_ clustering.enabled	Enable reporting metrics about Causal Clustering mode.	String	yes	no
metrics.neo4j.checkpointing.enabled	Enable reporting metrics about Neo4j check pointing; when it occurs and how much time it takes to complete.	String	yes	no
metrics.neo4j.cluster.enabled	Enable reporting metrics about HA cluster info.	String	yes	no
metrics.neo4j.counts.enabled	Enable reporting metrics about approximately how many entities are in the database; nodes, relationships, properties, etc.	String	yes	no
metrics.neo4j.enabled	The default enablement value for all Neo4j specific support metrics. Set this to false to turn off all Neo4j specific metrics by default. The individual metrics.neo4j.* metrics can then be turned on selectively.	String	yes	no
metrics.neo4j.logrotation.enabled	Enable reporting metrics about the Neo4j log rotation; when it occurs and how much time it takes to complete.	String	yes	no
metrics.neo4j.network.enabled	Enable reporting metrics about the network usage.	String	yes	no
metrics.neo4j.pagecache.enabled	Enable reporting metrics about the Neo4j page cache; page faults, evictions, flushes, exceptions, etc.	String	yes	no
metrics.neo4j.server.enabled	Enable reporting metrics about Server threading info.	String	yes	no
metrics.neo4j.tx.enabled	Enable reporting metrics about transactions; number of transactions started, committed, etc.	String	yes	no
metrics.prefix	A common prefix for the reported metrics field names. By default, this is either be 'neo4j', or a computed value based on the cluster and instance names, when running in an HA configuration.	String	yes	no
metrics.prometheus.enabled	Set to true to enable the Prometheus endpoint	String	yes	no

Name	Description	Type	Read	Write
metrics.prometheus.endpoint	The hostname and port to use as Prometheus endpoint	String	yes	no
tools.consistency_checker.check_graph	This setting is deprecated. See commandline arguments for neoj4-admin check-consistency instead. Perform checks between nodes, relationships, properties, types and tokens.	String	yes	no
tools.consistency_checker.check_indexes	This setting is deprecated. See commandline arguments for neoj4-admin check-consistency instead. Perform checks on indexes. Checking indexes is more expensive than checking the native stores, so it may be useful to turn off this check for very large databases.	String	yes	no
tools.consistency_checker.check_label_scan_store	This setting is deprecated. See commandline arguments for neoj4-admin check-consistency instead. Perform checks on the label scan store. Checking this store is more expensive than checking the native stores, so it may be useful to turn off this check for very large databases.	String	yes	no
tools.consistency_checker.check_property_owners	This setting is deprecated. See commandline arguments for neoj4-admin check-consistency instead. Perform optional additional checking on property ownership. This can detect a theoretical inconsistency where a property could be owned by multiple entities. However, the check is very expensive in time and memory, so it is skipped by default.	String	yes	no
unsupported.cypher.compiler_tracing	Enable tracing of compilation in cypher.	String	yes	no
unsupported.cypher.idp_solver_duration_threshold	To improve IDP query planning time, we can restrict the internal planning loop duration, triggering more frequent compaction of candidate plans. The smaller the threshold the faster the planning, but the higher the risk of sub-optimal plans.	String	yes	no
unsupported.cypher.idp_solver_table_threshold	To improve IDP query planning time, we can restrict the internal planning table size, triggering compaction of candidate plans. The smaller the threshold the faster the planning, but the higher the risk of sub-optimal plans.	String	yes	no
unsupported.cypher.morsel_size	The size of the morsels	String	yes	no
unsupported.cypher.non_indexed_label_warning_threshold	The threshold when a warning is generated if a label scan is done after a load csv where the label has no index	String	yes	no
unsupported.cypher.number_of_workers	Number of threads to allocate to Cypher worker threads. If set to 0, two workers will be started for every physical core in the system.	String	yes	no
unsupported.cypher.plan_with_minimum_cardinality_estimates	Enable using minimum cardinality estimates in the Cypher cost planner, so that cardinality estimates for logical plan operators are not allowed to go below certain thresholds even when the statistics give smaller numbers. This is especially useful for large import queries that write nodes and relationships into an empty or small database, where the generated query plan needs to be able to scale beyond the initial statistics. Otherwise, when this is disabled, the statistics on an empty or tiny database may lead the cost planner to for example pick a scan over an index seek, even when an index exists, because of a lower estimated cost.	String	yes	no

Name	Description	Type	Read	Write
unsupported.cypher.replan_ algorithm	Large databases might change slowly, and to prevent queries from never being replanned the divergence threshold set by cypher.statistics_divergence_threshold is configured to shrink over time using the algorithm set here. This will cause the threshold to reach the value set by unsupported.cypher.statistics_divergence_target once the time since the previous replanning has reached the value set in unsupported.cypher.target_replan_interval. Setting the algorithm to 'none' will cause the threshold to not decay over time.	String	yes	no
unsupported.cypher.runtime	Set this to specify the default runtime for the default language version.	String	yes	no
unsupported.cypher.statistics_ divergence_ target	Large databases might change slowly, and so to prevent queries from never being replanned the divergence threshold set by cypher.statistics_divergence_threshold is configured to shrink over time. The algorithm used to manage this change is set by unsupported.cypher.replan_algorithm and will cause the threshold to reach the value set here once the time since the previous replanning has reached unsupported.cypher.target_replan_interval. Setting this value to higher than the cypher.statistics_divergence_threshold will cause the threshold to not decay over time.	String	yes	no
unsupported.cypher.target_ replan_ interval	Large databases might change slowly, and to prevent queries from never being replanned the divergence threshold set by cypher.statistics_divergence_threshold is configured to shrink over time. The algorithm used to manage this change is set by unsupported.cypher.replan_algorithm and will cause the threshold to reach the value set by unsupported.cypher.statistics_divergence_target once the time since the previous replanning has reached the value set here. Setting this value to less than the value of cypher.min_replan_interval will cause the threshold to not decay over time.	String	yes	no
unsupported.dbms.block_ size.array_ properties	Specifies the block size for storing arrays. This parameter is only honored when the store is created, otherwise it is ignored. Also note that each block carries a ~10B of overhead so record size on disk will be slightly larger than the configured block size	String	yes	no
unsupported.dbms.block_ size.labels	Specifies the block size for storing labels exceeding in-lined space in node record. This parameter is only honored when the store is created, otherwise it is ignored. Also note that each block carries a ~10B of overhead so record size on disk will be slightly larger than the configured block size	String	yes	no
unsupported.dbms.block_ size.strings	Specifies the block size for storing strings. This parameter is only honored when the store is created, otherwise it is ignored. Note that each character in a string occupies two bytes, meaning that e.g a block size of 120 will hold a 60 character long string before overflowing into a second block. Also note that each block carries a ~10B of overhead so record size on disk will be slightly larger than the configured block size	String	yes	no

Name	Description	Type	Read	Write
unsupported.dbms.bolt.inbound_message_throttle.high_watermark	When the number of queued inbound messages grows beyond this value, reading from underlying channel will be paused (no more inbound messages will be available) until queued number of messages drops below the configured low watermark value.	String	yes	no
unsupported.dbms.bolt.inbound_message_throttle.low_watermark	When the number of queued inbound messages, previously reached configured high watermark value, drops below this value, reading from underlying channel will be enabled and any pending messages will start queuing again.	String	yes	no
unsupported.dbms.bolt.outbound_buffer_throttle	Whether to apply network level outbound network buffer based throttling	String	yes	no
unsupported.dbms.bolt.outbound_buffer_throttle.high_watermark	When the size (in bytes) of outbound network buffers, used by bolt's network layer, grows beyond this value bolt channel will advertise itself as unwritable and will block related processing thread until it becomes writable again.	String	yes	no
unsupported.dbms.bolt.outbound_buffer_throttle.low_watermark	When the size (in bytes) of outbound network buffers, previously advertised as unwritable, gets below this value bolt channel will re-advertise itself as writable and blocked processing thread will resume execution.	String	yes	no
unsupported.dbms.bolt.outbound_buffer_throttle.max_duration	When the total time outbound network buffer based throttle lock is held exceeds this value, the corresponding bolt channel will be aborted. Setting this to 0 will disable this behaviour.	String	yes	no
unsupported.dbms.counts_store_rotation_timeout	Maximum time to wait for active transaction completion when rotating counts store	String	yes	no
unsupported.dbms.db.spatial.crs.cartesian-3d.x.max	The maximum x value for the index extents for 3D cartesian-3d spatial index. The 3Dto 1D mapping function divides all 3Dspace into discrete tiles, and orders these using a space filling curve designed to optimize the requirement that tiles that are close together in this ordered list are also close together in 3Dspace. This requires that the extents of the 3Dspace be known in advance and never changed. If you do change these settings, you need to recreate any affected index in order for the settings to apply, otherwise the index will retain the previous settings.	String	yes	no
unsupported.dbms.db.spatial.crs.cartesian-3d.x.min	The minimum x value for the index extents for 3D cartesian-3d spatial index. The 3Dto 1D mapping function divides all 3Dspace into discrete tiles, and orders these using a space filling curve designed to optimize the requirement that tiles that are close together in this ordered list are also close together in 3Dspace. This requires that the extents of the 3Dspace be known in advance and never changed. If you do change these settings, you need to recreate any affected index in order for the settings to apply, otherwise the index will retain the previous settings.	String	yes	no

Name	Description	Type	Read	Write
unsupported.dbms.db.spatial.crs.cartesian-3d.y.max	The maximum y value for the index extents for 3D cartesian-3d spatial index. The 3Dto 1D mapping function divides all 3Dspace into discrete tiles, and orders these using a space filling curve designed to optimize the requirement that tiles that are close together in this ordered list are also close together in 3Dspace. This requires that the extents of the 3Dspace be known in advance and never changed. If you do change these settings, you need to recreate any affected index in order for the settings to apply, otherwise the index will retain the previous settings.	String	yes	no
unsupported.dbms.db.spatial.crs.cartesian-3d.y.min	The minimum y value for the index extents for 3D cartesian-3d spatial index. The 3Dto 1D mapping function divides all 3Dspace into discrete tiles, and orders these using a space filling curve designed to optimize the requirement that tiles that are close together in this ordered list are also close together in 3Dspace. This requires that the extents of the 3Dspace be known in advance and never changed. If you do change these settings, you need to recreate any affected index in order for the settings to apply, otherwise the index will retain the previous settings.	String	yes	no
unsupported.dbms.db.spatial.crs.cartesian-3d.z.max	The maximum z value for the index extents for 3D cartesian-3d spatial index. The 3Dto 1D mapping function divides all 3Dspace into discrete tiles, and orders these using a space filling curve designed to optimize the requirement that tiles that are close together in this ordered list are also close together in 3Dspace. This requires that the extents of the 3Dspace be known in advance and never changed. If you do change these settings, you need to recreate any affected index in order for the settings to apply, otherwise the index will retain the previous settings.	String	yes	no
unsupported.dbms.db.spatial.crs.cartesian-3d.z.min	The minimum z value for the index extents for 3D cartesian-3d spatial index. The 3Dto 1D mapping function divides all 3Dspace into discrete tiles, and orders these using a space filling curve designed to optimize the requirement that tiles that are close together in this ordered list are also close together in 3Dspace. This requires that the extents of the 3Dspace be known in advance and never changed. If you do change these settings, you need to recreate any affected index in order for the settings to apply, otherwise the index will retain the previous settings.	String	yes	no
unsupported.dbms.db.spatial.crs.cartesian.x.max	The maximum x value for the index extents for 2D cartesian spatial index. The 2Dto 1D mapping function divides all 2Dspace into discrete tiles, and orders these using a space filling curve designed to optimize the requirement that tiles that are close together in this ordered list are also close together in 2Dspace. This requires that the extents of the 2Dspace be known in advance and never changed. If you do change these settings, you need to recreate any affected index in order for the settings to apply, otherwise the index will retain the previous settings.	String	yes	no

Name	Description	Type	Read	Write
unsupported.dbms.db.spatial.crs.cartesian.x.min	The minimum x value for the index extents for 2D cartesian spatial index. The 2Dto 1D mapping function divides all 2Dspace into discrete tiles, and orders these using a space filling curve designed to optimize the requirement that tiles that are close together in this ordered list are also close together in 2Dspace. This requires that the extents of the 2Dspace be known in advance and never changed. If you do change these settings, you need to recreate any affected index in order for the settings to apply, otherwise the index will retain the previous settings.	String	yes	no
unsupported.dbms.db.spatial.crs.cartesian.y.max	The maximum y value for the index extents for 2D cartesian spatial index. The 2Dto 1D mapping function divides all 2Dspace into discrete tiles, and orders these using a space filling curve designed to optimize the requirement that tiles that are close together in this ordered list are also close together in 2Dspace. This requires that the extents of the 2Dspace be known in advance and never changed. If you do change these settings, you need to recreate any affected index in order for the settings to apply, otherwise the index will retain the previous settings.	String	yes	no
unsupported.dbms.db.spatial.crs.cartesian.y.min	The minimum y value for the index extents for 2D cartesian spatial index. The 2Dto 1D mapping function divides all 2Dspace into discrete tiles, and orders these using a space filling curve designed to optimize the requirement that tiles that are close together in this ordered list are also close together in 2Dspace. This requires that the extents of the 2Dspace be known in advance and never changed. If you do change these settings, you need to recreate any affected index in order for the settings to apply, otherwise the index will retain the previous settings.	String	yes	no
unsupported.dbms.db.spatial.crs.wgs-84-3d.x.max	The maximum x value for the index extents for 3D wgs-84-3d spatial index. The 3Dto 1D mapping function divides all 3Dspace into discrete tiles, and orders these using a space filling curve designed to optimize the requirement that tiles that are close together in this ordered list are also close together in 3Dspace. This requires that the extents of the 3Dspace be known in advance and never changed. If you do change these settings, you need to recreate any affected index in order for the settings to apply, otherwise the index will retain the previous settings.	String	yes	no
unsupported.dbms.db.spatial.crs.wgs-84-3d.x.min	The minimum x value for the index extents for 3D wgs-84-3d spatial index. The 3Dto 1D mapping function divides all 3Dspace into discrete tiles, and orders these using a space filling curve designed to optimize the requirement that tiles that are close together in this ordered list are also close together in 3Dspace. This requires that the extents of the 3Dspace be known in advance and never changed. If you do change these settings, you need to recreate any affected index in order for the settings to apply, otherwise the index will retain the previous settings.	String	yes	no

Name	Description	Type	Read	Write
unsupported.dbms.db.spatial.crs.wgs-84-3d.y.max	The maximum y value for the index extents for 3D wgs-84-3d spatial index. The 3Dto 1D mapping function divides all 3Dspace into discrete tiles, and orders these using a space filling curve designed to optimize the requirement that tiles that are close together in this ordered list are also close together in 3Dspace. This requires that the extents of the 3Dspace be known in advance and never changed. If you do change these settings, you need to recreate any affected index in order for the settings to apply, otherwise the index will retain the previous settings.	String	yes	no
unsupported.dbms.db.spatial.crs.wgs-84-3d.y.min	The minimum y value for the index extents for 3D wgs-84-3d spatial index. The 3Dto 1D mapping function divides all 3Dspace into discrete tiles, and orders these using a space filling curve designed to optimize the requirement that tiles that are close together in this ordered list are also close together in 3Dspace. This requires that the extents of the 3Dspace be known in advance and never changed. If you do change these settings, you need to recreate any affected index in order for the settings to apply, otherwise the index will retain the previous settings.	String	yes	no
unsupported.dbms.db.spatial.crs.wgs-84-3d.z.max	The maximum z value for the index extents for 3D wgs-84-3d spatial index. The 3Dto 1D mapping function divides all 3Dspace into discrete tiles, and orders these using a space filling curve designed to optimize the requirement that tiles that are close together in this ordered list are also close together in 3Dspace. This requires that the extents of the 3Dspace be known in advance and never changed. If you do change these settings, you need to recreate any affected index in order for the settings to apply, otherwise the index will retain the previous settings.	String	yes	no
unsupported.dbms.db.spatial.crs.wgs-84-3d.z.min	The minimum z value for the index extents for 3D wgs-84-3d spatial index. The 3Dto 1D mapping function divides all 3Dspace into discrete tiles, and orders these using a space filling curve designed to optimize the requirement that tiles that are close together in this ordered list are also close together in 3Dspace. This requires that the extents of the 3Dspace be known in advance and never changed. If you do change these settings, you need to recreate any affected index in order for the settings to apply, otherwise the index will retain the previous settings.	String	yes	no
unsupported.dbms.db.spatial.crs.wgs-84.x.max	The maximum x value for the index extents for 2D wgs-84 spatial index. The 2Dto 1D mapping function divides all 2Dspace into discrete tiles, and orders these using a space filling curve designed to optimize the requirement that tiles that are close together in this ordered list are also close together in 2Dspace. This requires that the extents of the 2Dspace be known in advance and never changed. If you do change these settings, you need to recreate any affected index in order for the settings to apply, otherwise the index will retain the previous settings.	String	yes	no

Name	Description	Type	Read	Write
unsupported.dbms.db.spatial.crs.wgs-84.x.min	The minimum x value for the index extents for 2D wgs-84 spatial index. The 2Dto 1D mapping function divides all 2Dspace into discrete tiles, and orders these using a space filling curve designed to optimize the requirement that tiles that are close together in this ordered list are also close together in 2Dspace. This requires that the extents of the 2Dspace be known in advance and never changed. If you do change these settings, you need to recreate any affected index in order for the settings to apply, otherwise the index will retain the previous settings.	String	yes	no
unsupported.dbms.db.spatial.crs.wgs-84.y.max	The maximum y value for the index extents for 2D wgs-84 spatial index. The 2Dto 1D mapping function divides all 2Dspace into discrete tiles, and orders these using a space filling curve designed to optimize the requirement that tiles that are close together in this ordered list are also close together in 2Dspace. This requires that the extents of the 2Dspace be known in advance and never changed. If you do change these settings, you need to recreate any affected index in order for the settings to apply, otherwise the index will retain the previous settings.	String	yes	no
unsupported.dbms.db.spatial.crs.wgs-84.y.min	The minimum y value for the index extents for 2D wgs-84 spatial index. The 2Dto 1D mapping function divides all 2Dspace into discrete tiles, and orders these using a space filling curve designed to optimize the requirement that tiles that are close together in this ordered list are also close together in 2Dspace. This requires that the extents of the 2Dspace be known in advance and never changed. If you do change these settings, you need to recreate any affected index in order for the settings to apply, otherwise the index will retain the previous settings.	String	yes	no
unsupported.dbms.directories.auth	Configuration attribute	String	yes	no
unsupported.dbms.directories.database	Configuration attribute	String	yes	no
unsupported.dbms.directories.neo4j_ home	Root relative to which directory settings are resolved. This is set in code and should never be configured explicitly.	String	yes	no
unsupported.dbms.disconnected	Disable all protocol connectors.	String	yes	no
unsupported.dbms.edition	Configuration attribute	String	yes	no
unsupported.dbms.enable_ native_ schema_ index	Configuration attribute	String	yes	no
unsupported.dbms.ephemeral	Configuration attribute	String	yes	no
unsupported.dbms.executiontime_ limit.enabled	Please use dbms.transaction.timeout instead.	String	yes	no
unsupported.dbms.id_ generator_ fast_ rebuild_ enabled	Use a quick approach for rebuilding the ID generators. This give quicker recovery time, but will limit the ability to reuse the space of deleted entities.	String	yes	no

Name	Description	Type	Read	Write
`unsupported.dbms.id_ reuse_ safe_ zone`	Duration for which master will buffer ids and not reuse them to allow slaves read consistently. Slaves will also terminate transactions longer than this duration, when applying received transaction stream, to make sure they do not read potentially inconsistent/reused records.	`String`	yes	no
`unsupported.dbms.index. archive_ failed`	Create an archive of an index before re-creating it if failing to load on startup.	`String`	yes	no
`unsupported.dbms.index. spatial.curve.bottom_ threshold`	When searching the spatial index we need to convert a 2D range in the quad tree into a set of 1D ranges on the underlying 1D space filling curve index. There is a balance to be made between many small 1D ranges that have few false positives, and fewer, larger 1D ranges that have more false positives. The former has a more efficient filtering of false positives, while the latter will have a more efficient search of the numerical index. The maximum depth to which the quad tree is processed when mapping 2D to 1D is based on the size of the search area compared to the size of the 2D tiles at that depth. When traversing the tree to this depth, we can stop early based on when the search envelope overlaps the current tile by more than a certain threshold. The threshold is calculated based on depth, from the `top_threshold` at the top of the tree to the `bottom_threshold` at the depth calculated by the area comparison. Setting the top to 0.99 and the bottom to 0.5, for example would mean that if we reached the maximum depth, and the search area overlapped the current tile by more than 50%, we would stop traversing the tree, and return the 1D range for that entire tile to the search set. If the overlap is even higher, we would stop higher in the tree. This technique reduces the number of 1D ranges passed to the underlying space filling curve index. Setting this value to zero turns off this feature.	`String`	yes	no
`unsupported.dbms.index. spatial.curve.extra_ levels`	When searching the spatial index we need to convert a 2D range in the quad tree into a set of 1D ranges on the underlying 1D space filling curve index. There is a balance to be made between many small 1D ranges that have few false positives, and fewer, larger 1D ranges that have more false positives. The former has a more efficient filtering of false positives, while the latter will have a more efficient search of the numerical index. The maximum depth to which the quad tree is processed when mapping 2D to 1D is based on the size of the search area compared to the size of the 2D tiles at that depth. This setting will cause the algorithm to search deeper, reducing false positives.	`String`	yes	no

Name	Description	Type	Read	Write
unsupported.dbms.index.spatial.curve.max_ bits	The maximum number of bits to use for levels in the quad tree representing the spatial index. When creating the spatial index, we simulate a quad tree using a 2D (or 3D) to 1D mapping function. This requires that the extents of the index and the depth of the tree be defined in advance, so ensure the 2D to 1D mapping is deterministic and repeatable. This setting will define the maximum depth of any future spatial index created, calculated as max_bits / dimensions. For example 60 bits will define 30 levels in 2D and 20 levels in 3D. Existing indexes will not be changed, and need to be recreated if you wish to use the new value. For 2D indexes, a value of 30 is the largest supported. For 3D indexes 20 is the largest.	String	yes	no
unsupported.dbms.index.spatial.curve.top_ threshold	When searching the spatial index we need to convert a 2D range in the quad tree into a set of 1D ranges on the underlying 1D space filling curve index. There is a balance to be made between many small 1D ranges that have few false positives, and fewer, larger 1D ranges that have more false positives. The former has a more efficient filtering of false positives, while the latter will have a more efficient search of the numerical index. The maximum depth to which the quad tree is processed when mapping 2D to 1D is based on the size of the search area compared to the size of the 2D tiles at that depth. When traversing the tree to this depth, we can stop early based on when the search envelope overlaps the current tile by more than a certain threshold. The threshold is calculated based on depth, from the top_threshold at the top of the tree to the bottom_threshold at the depth calculated by the area comparison. Setting the top to 0.99 and the bottom to 0.5, for example would mean that if we reached the maximum depth, and the search area overlapped the current tile by more than 50%, we would stop traversing the tree, and return the 1D range for that entire tile to the search set. If the overlap is even higher, we would stop higher in the tree. This technique reduces the number of 1D ranges passed to the underlying space filling curve index. Setting this value to zero turns off this feature.	String	yes	no
unsupported.dbms.kernel_ id	An identifier that uniquely identifies this graph database instance within this JVM. Defaults to an auto-generated number depending on how many instance are started in this JVM.	String	yes	no
unsupported.dbms.lock_ manager	Configuration attribute	String	yes	no
unsupported.dbms.logs.bolt.enabled	Configuration attribute	String	yes	no
unsupported.dbms.logs.bolt.path	Configuration attribute	String	yes	no
unsupported.dbms.logs.debug.debug_ loggers	Debug log contexts that should output debug level logging	String	yes	no
unsupported.dbms.memory.pagecache.pagesize	Target size for pages of mapped memory. If set to 0, then a reasonable default is chosen, depending on the storage device used.	String	yes	no

Name	Description	Type	Read	Write
unsupported.dbms.memory.pagecache.warmup.enable	Page cache can be configured to perform usage sampling of loaded pages that can be used to construct active load profile. According to that profile pages can be reloaded on the restart, replication, etc. This setting allows disabling that behavior. This feature available in Neo4j Enterprise Edition.	String	yes	no
unsupported.dbms.memory.pagecache.warmup.profile.interval	The profiling frequency for the page cache. Accurate profiles allow the page cache to do active warmup after a restart, reducing the mean time to performance. This feature available in Neo4j Enterprise Edition.	String	yes	no
unsupported.dbms.multi_threaded_schema_index_population_enabled	Configuration attribute	String	yes	no
unsupported.dbms.query.snapshot	Specifies if engine should run cypher query based on a snapshot of accessed data. Query will be restarted in case if concurrent modification of data will be detected.	String	yes	no
unsupported.dbms.query.snapshot.retries	Specifies number or retries that query engine will do to execute query based on stable accessed data snapshot before giving up.	String	yes	no
unsupported.dbms.record_id_batch_size	Specifies the size of id batches local to each transaction when committing. Committing a transaction which contains changes most often results in new data records being created. For each record a new id needs to be generated from an id generator. It's more efficient to allocate a batch of ids from the contended id generator, which the transaction holds and generates ids from while creating these new records. This setting specifies how big those batches are. Remaining ids are freed back to id generator on clean shutdown.	String	yes	no
unsupported.dbms.report_configuration	Print out the effective Neo4j configuration after startup.	String	yes	no
unsupported.dbms.schema.release_lock_while_building_constraint	Whether or not to release the exclusive schema lock is while building uniqueness constraints index	String	yes	no
unsupported.dbms.security.auth_max_failed_attempts	Configuration attribute	String	yes	no
unsupported.dbms.security.auth_store.location	Configuration attribute	String	yes	no
unsupported.dbms.security.ldap.authorization.connection_pooling	Set to true if connection pooling should be used for authorization searches using the system account.	String	yes	no
unsupported.dbms.security.module	Configuration attribute	String	yes	no
unsupported.dbms.security.tls_certificate_file	Path to the X.509 public certificate to be used by Neo4j for TLS connections	String	yes	no
unsupported.dbms.security.tls_key_file	Path to the X.509 private key to be used by Neo4j for TLS connections	String	yes	no
unsupported.dbms.tracer	Configuration attribute	String	yes	no
unsupported.dbms.transaction_start_timeout	The maximum amount of time to wait for the database to become available, when starting a new transaction.	String	yes	no
unsupported.dbms.tx_state.memory_allocation	[Experimental] Defines whether memory for transaction state should allocaten on- or off-heap.	String	yes	no

Name	Description	Type	Read	Write
unsupported.dbms.udc.first_ delay	Configuration attribute	String	yes	no
unsupported.dbms.udc.host	Configuration attribute	String	yes	no
unsupported.dbms.udc.interval	Configuration attribute	String	yes	no
unsupported.dbms.udc.reg	Configuration attribute	String	yes	no
unsupported.dbms.udc.source	Configuration attribute	String	yes	no
unsupported.ha.cluster_ name	The name of a cluster.	String	yes	no
unsupported.ha.instance_ name	Configuration attribute	String	yes	no
unsupported.tools.batch_ inserter.batch_ size	Specifies number of operations that batch inserter will try to group into one batch before flushing data into underlying storage.	String	yes	no
unsupported.vm_ pause_ monitor.measurement_ duration	Configuration attribute	String	yes	no
unsupported.vm_ pause_ monitor.stall_ alert_ threshold	Configuration attribute	String	yes	no

Table 6. MBean Diagnostics (org.neo4j.management.Diagnostics) Attributes

Name	Description	Type	Read	Write
Diagnostics provided by Neo4j				
DiagnosticsProviders	A list of the ids for the registered diagnostics providers.	List (java.util.List)	yes	no

Table 7. MBean Diagnostics (org.neo4j.management.Diagnostics) Operations

Name	Description	ReturnType	Signature
dumpAll	Dump diagnostics information to JMX	String	(no parameters)
dumpToLog	Dump diagnostics information to the log.	void	(no parameters)
dumpToLog	Dump diagnostics information to the log.	void	java.lang.String
extract	Operation exposed for management	String	java.lang.String

Table 8. MBean Index sampler (org.neo4j.management.IndexSamplingManager) Operations

Name	Description	ReturnType	Signature
triggerIndexSampling	triggerIndexSampling	void	java.lang.String,java.lang.String,boolean

Table 9. MBean Kernel (org.neo4j.jmx.Kernel) Attributes

Name	Description	Type	Read	Write
Information about the Neo4j kernel				
DatabaseName	The name of the mounted database	String	yes	no
KernelStartTime	The time from which this Neo4j instance was in operational mode.	Date (java.util.Date)	yes	no
KernelVersion	The version of Neo4j	String	yes	no

Name	Description	Type	Read	Write
MBeanQuery	An ObjectName that can be used as a query for getting all management beans for this Neo4j instance.	javax.management.ObjectName	yes	no
ReadOnly	Whether this is a read only instance	boolean	yes	no
StoreCreationDate	The time when this Neo4j graph store was created.	Date (java.util.Date)	yes	no
StoreId	An identifier that, together with store creation time, uniquely identifies this Neo4j graph store.	String	yes	no
StoreLogVersion	The current version of the Neo4j store logical log.	long	yes	no

Table 10. MBean Locking (org.neo4j.management.LockManager) Attributes

Name	Description	Type	Read	Write
Information about the Neo4j lock status				
Locks	Information about all locks held by Neo4j	java.util.List<org.neo4j.kernel.info.LockInfo> as CompositeData[]	yes	no
NumberOfAvertedDeadlocks	The number of lock sequences that would have lead to a deadlock situation that Neo4j has detected and averted (by throwing DeadlockDetectedException).	long	yes	no

Table 11. MBean Locking (org.neo4j.management.LockManager) Operations

Name	Description	ReturnType	Signature
getContendedLocks	getContendedLocks	java.util.List<org.neo4j.kernel.info.LockInfo> as CompositeData[]	long

Table 12. MBean Memory Mapping (org.neo4j.management.MemoryMapping) Attributes

Name	Description	Type	Read	Write
The status of Neo4j memory mapping				
MemoryPools	Get information about each pool of memory mapped regions from store files with memory mapping enabled	org.neo4j.management.WindowPoolInfo[] as CompositeData[]	yes	no

Table 13. MBean Page cache (org.neo4j.management.PageCache) Attributes

Name	Description	Type	Read	Write
Information about the Neo4j page cache. All numbers are counts and sums since the Neo4j instance was started				
BytesRead	Number of bytes read from durable storage.	long	yes	no
BytesWritten	Number of bytes written to durable storage.	long	yes	no
EvictionExceptions	Number of exceptions caught during page eviction. This number should be zero, or at least not growing, in a healthy database. Otherwise it could indicate drive failure, storage space, or permission problems.	long	yes	no
Evictions	Number of page evictions. How many pages have been removed from memory to make room for other pages.	long	yes	no
Faults	Number of page faults. How often requested data was not found in memory and had to be loaded.	long	yes	no

Name	Description	Type	Read	Write
FileMappings	Number of files that have been mapped into the page cache.	long	yes	no
FileUnmappings	Number of files that have been unmapped from the page cache.	long	yes	no
Flushes	Number of page flushes. How many dirty pages have been written to durable storage.	long	yes	no
HitRatio	Ratio of hits to the total number of lookups in the page cache	double	yes	no
Pins	Number of page pins. How many pages have been accessed (monitoring must be enabled separately).	long	yes	no
UsageRatio	The percentage of used pages. Will return NaN if it cannot be determined.	double	yes	no

Table 14. MBean Primitive count (org.neo4j.jmx.Primitives) Attributes

Name	Description	Type	Read	Write
Estimates of the numbers of different kinds of Neo4j primitives				
NumberOf NodeIds InUse	An estimation of the number of nodes used in this Neo4j instance	long	yes	no
NumberOf PropertyIds In Use	An estimation of the number of properties used in this Neo4j instance	long	yes	no
NumberOf RelationshipId s InUse	An estimation of the number of relationships used in this Neo4j instance	long	yes	no
NumberOf RelationshipTy peIds InUse	The number of relationship types used in this Neo4j instance	long	yes	no

Table 15. MBean Reports (org.neo4j.dbms.diagnostics.jmx.Reports) Attributes

Name	Description	Type	Read	Write
Reports operations				
EnvironmentVariables	Returns a map if the current environment variables	String	yes	no

Table 16. MBean Reports (org.neo4j.dbms.diagnostics.jmx.Reports) Operations

Name	Description	ReturnType	Signature
listTransactions	List all active transactions	String	(no parameters)

Table 17. MBean Store file sizes (org.neo4j.jmx.StoreFile) Attributes

Name	Description	Type	Read	Write
This bean is deprecated, use StoreSize bean instead; Information about the sizes of the different parts of the Neo4j graph store				
ArrayStoreSize	The amount of disk space used to store array properties, in bytes.	long	yes	no
LogicalLogSize	The amount of disk space used by the current Neo4j logical log, in bytes.	long	yes	no
NodeStoreSize	The amount of disk space used to store nodes, in bytes.	long	yes	no
PropertyStoreSize	The amount of disk space used to store properties (excluding string values and array values), in bytes.	long	yes	no
RelationshipStoreSize	The amount of disk space used to store relationships, in bytes.	long	yes	no

Name	Description	Type	Read	Write
StringStoreSize	The amount of disk space used to store string properties, in bytes.	long	yes	no
TotalStoreSize	The total disk space used by this Neo4j instance, in bytes.	long	yes	no

Table 18. MBean Store sizes (org.neo4j.jmx.StoreSize) Attributes

Name	Description	Type	Read	Write
Information about the disk space used by different parts of the Neo4j graph store				
ArrayStoreSize	Disk space used to store array properties, in bytes.	long	yes	no
CountStoreSize	Disk space used to store counters, in bytes	long	yes	no
IndexStoreSize	Disk space used to store all indices, in bytes	long	yes	no
LabelStoreSize	Disk space used to store labels, in bytes	long	yes	no
NodeStoreSize	Disk space used to store nodes, in bytes.	long	yes	no
PropertyStoreSize	Disk space used to store properties (excluding string values and array values), in bytes.	long	yes	no
RelationshipStoreSize	Disk space used to store relationships, in bytes.	long	yes	no
SchemaStoreSize	Disk space used to store schemas (index and constrain declarations), in bytes	long	yes	no
StringStoreSize	Disk space used to store string properties, in bytes.	long	yes	no
TotalStoreSize	Disk space used by whole store, in bytes.	long	yes	no
TransactionLogsSize	Disk space used by the transaction logs, in bytes.	long	yes	no

Table 19. MBean Transactions (org.neo4j.management.TransactionManager) Attributes

Name	Description	Type	Read	Write
Information about the Neo4j transaction manager				
LastCommittedTxId	The id of the latest committed transaction	long	yes	no
NumberOf Committed Transactions	The total number of committed transactions	long	yes	no
NumberOf Opened Transactions	The total number started transactions	long	yes	no
NumberOf Open Transactions	The number of currently open transactions	long	yes	no
NumberOf RolledBack Transactions	The total number of rolled back transactions	long	yes	no
PeakNumberOf Concurrent Transactions	The highest number of transactions ever opened concurrently	long	yes	no

9.4.2. High Availability JMX MBeans

The following JMX management beans are unique to instances that are part of a High Availability cluster.

MBeans exposed by Neo4j in High Availability mode

- Branched Store: Information about the branched stores present in this HA cluster member.
- High Availability: Information about an instance participating in a HA cluster.

Table 20. MBean Branched Store (org.neo4j.management.BranchedStore) Attributes

Name	Description	Type	Read	Write
Information about the branched stores present in this HA cluster member				
`BranchedStores`	A list of the branched stores	`org.neo4j.management.BranchedStoreInfo[] as CompositeData[]`	yes	no

Table 21. MBean High Availability (org.neo4j.management.HighAvailability) Attributes

Name	Description	Type	Read	Write
Information about an instance participating in a HA cluster				
`Alive`	Whether this instance is alive or not	`boolean`	yes	no
`Available`	Whether this instance is available or not	`boolean`	yes	no
`InstanceId`	The identifier used to identify this server in the HA cluster	`String`	yes	no
`InstancesInCluster`	Information about all instances in this cluster	`org.neo4j.management.ClusterMemberInfo[] as CompositeData[]`	yes	no
`LastCommittedTxId`	The latest transaction id present in this instance's store	`long`	yes	no
`LastUpdateTime`	The time when the data on this instance was last updated from the master	`String`	yes	no
`Role`	The role this instance has in the cluster	`String`	yes	no

Table 22. MBean High Availability (org.neo4j.management.HighAvailability) Operations

Name	Description	ReturnType	Signature
`update`	(If this is a slave) Update the database on this instance with the latest transactions from the master	`String`	`(no parameters)`

License

ShareAlike

If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.

No additional restrictions

You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.

Notices

You do not have to comply with the license for elements of the material in the public domain or where your use is permitted by an applicable exception or limitation.

No warranties are given. The license may not give you all of the permissions necessary for your intended use. For example, other rights such as publicity, privacy, or moral rights may limit how you use the material.

See https://creativecommons.org/licenses/by-nc-sa/4.0/ for further details. The full license text is available at https://creativecommons.org/licenses/by-nc-sa/4.0/legalcode.