Neo4j

the graph database

The Neo4j Manual v1.4.2

The Neo4j Team *neo4j.org* <http://neo4j.org/>
***neotechnology.com* <http://neotechnology.com/>**

The Neo4j Manual v1.4.2

by The Neo4j Team *neo4j.org* <http://neo4j.org/> *neotechnology.com* <http://neotechnology.com/>

License: Creative Commons 3.0

Table of Contents

List of Figures

List of Tables

Introduction

This is a reference manual. The material is practical, technical, and focused on answering specific questions. It addresses how things work, what to do and what to avoid to successfully run Neo4j in a production environment. After a brief introduction, each topic area assumes general familiarity as it addresses the particular details of Neo4j.

The goal is to be thumb-through and rule-of-thumb friendly.

Each section should stand on its own, so you can hop right to whatever interests you. When possible, the sections distill "rules of thumb" which you can keep in mind whenever you wander out of the house without this manual in your back pocket.

1. Who should read this

The topics should be relevant to architects, administrators, developers and operations personnel. You should already know about Neo4j and using graphs to store data. If you are completely new to Neo4j please check out http://neo4j.org first.

2. Neo4j highlights

As a robust, scalable and high-performance database, Neo4j is suitable for lightweight projects or full enterprise deployment.

It features:

- true ACID transactions
- high availability
- scales to billions of nodes and relationships
- high speed querying through traversals

Proper ACID behavior is the foundation of data reliability. Neo4j enforces that all mutating operations occur within a transaction, guaranteeing consistent data. This robustness extends from single instance embedded graphs to multi-server high availability installations. For details, see Chapter 3, *Transaction Management*.

Reliable graph storage can easily be added to any application. A property graph can scale in size and complexity as the application evolves, with little impact on performance. Whether starting new development, or augmenting existing functionality, Neo4j is only limited by physical hardware.

A single server instance can handle a graph of billions of nodes and relationships. When data throughput is insufficient, the graph database can be distributed among multiple servers in a high availability configuration. See Chapter 9, *High Availability* to learn more.

The graph database storage shines when storing richly-connected data. Querying is performed through traversals, which can perform millions of "joins" per second.

Part I. Reference Documentation

Chapter 1. Installation & Deployment

1.1. Deployment Scenarios

Neo4j can be embedded into your application, run as a standalone server or deployed on several machines to provide high availability.

Table 1.1. Neo4j deployment options

	Single Instance	Multiple Instances
Embedded	EmbeddedGraphDatabase	HighlyAvailableGraphDatabase
Standalone	Neo4j Server	Neo4j Server high availability mode

1.1.1. Server

Neo4j is normally accessed as a standalone server, either directly through a REST interface or through a language-specific driver. More information about Neo4j server is found in Chapter 5, *Neo4j Server*. For running the server in high availability mode, see Section 5.4, “Starting the Neo4j server in high availability mode”.

1.1.2. Embedded

Neo4j can be embedded directly in a server application by including the appropriate Java libraries. When programming, you can refer to the `GraphDatabaseService` `<http://components.neo4j.org/neo4j/1.4.2/apidocs/org/neo4j/graphdb/GraphDatabaseService.html>` API. To switch from a single instance to multiple highly available instances, simply switch from the concrete `EmbeddedGraphDatabase` `<http://components.neo4j.org/neo4j/1.4.2/apidocs/org/neo4j/kernel/EmbeddedGraphDatabase.html>` to the `HighlyAvailableGraphDatabase` `<http://components.neo4j.org/neo4j-enterprise/1.4.2/apidocs/org/neo4j/kernel/HighlyAvailableGraphDatabase.html>`.

1.2. System Requirements

Memory constrains graph size, disk I/O constrains read/write performance, as always.

1.2.1. CPU

Performance is generally memory or I/O bound for large graphs, and compute bound for graphs which fit in memory.

Minimum
: Intel 486

Recommended
: Intel Core i7

1.2.2. Memory

More memory allows even larger graphs, but runs the risk of inducing larger Garbage Collection operations.

Minimum
: 1GB

Recommended
: 4-8GB

1.2.3. Disk

Aside from capacity, the performance characteristics of the disk are the most important when selecting storage.

Minimum
: SCSI, EIDE

Recommended
: SSD w/ SATA

1.2.4. Filesystem

For proper ACID behavior, the filesystem must support flush (fsync, fdatasync).

Minimum
: ext3 (or similar)

Recommended
: ext4, ZFS

1.2.5. Software

Neo4j is Java-based.

Java
: 1.6+

Operating Systems
: Linux, Windows XP, Mac OS X

1.3. Installation

Neo4j can be installed as a server, running either as a headless application or system service. For Java developers, it is also possible to use Neo4j as a library, embedded in your application.

For information on installing Neo4j as a server, see Section 5.1, “Server Installation”.

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

Tip

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

Table 1.2. Neo4j editions

Edition	**Dependency**	**Description**	**License**
Community	org.neo4j:neo4j <http://search.maven.org/#search|gav|1|g%3A%22org.neo4j%22%20AND%20a%3A%22neo4j%22>	a high performance, fully ACID transactional graph database	GPLv3
Advanced	org.neo4j:neo4j-advanced <http://search.maven.org/#search|gav|1|g%3A%22org.neo4j%22%20AND%20a%3A%22neo4j-advanced%22>	adding advanced monitoring	AGPLv3
Enterprise	org.neo4j:neo4j-enterprise <http://search.maven.org/#search|gav|1|g%3A%22org.neo4j%22%20AND%20a%3A%22neo4j-enterprise%22>	adding online backup and High Availability clustering	AGPLv3

Note

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

For more information regarding licensing, see the Licensing Guide <http://neo4j.org/licensing-guide/>.

1.3.1. Embedded Installation

The latest release is always available from http://neo4j.org/download, included as part of the Neo4j download packages. After selecting the appropriate version for your platform, embed Neo4j in your Java application by including the Neo4j library jars in your build. Either take the jar files from the *lib/*

directory of the download, or directly use the artifacts available from Maven Central Repository [1]. Stable and milestone releases are available there.

For information on how to use Neo4j as a dependency with Maven and other dependency management tools, see the following table:

Note

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

Maven dependency.

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

Where `${neo4j-version}` is the intended version and the `artifactId` is one of `neo4j`, `neo4j-advanced`, `neo4j-enterprise`.

---

[1] http://repo1.maven.org/maven2/org/neo4j/

1.4. Upgrading

Normally a properly shutdown Neo4j database can be upgraded directly to a new minor version. A database can be upgraded from a minor version to the next, e.g. 1.1 –> 1.2, and 1.2 –> 1.3, but you can not jump directly from 1.1 –> 1.3. The upgrade process is a one way step; databases cannot be downgraded.

However, some upgrades make significant changes to the database store. Neo4j will refuse to start when a significant upgrade is required, requiring explicit upgrade configuration.

1.4.1. Normal Upgrade

To perform a normal upgrade (for minor changes to the database store):

1. download the newer version of Neo4j
2. cleanly shutdown the database to upgrade, if it is running
3. startup the database with the newer version of Neo4j

1.4.2. Special Upgrade

To perform a special upgrade (for significant changes to the database store):

1. make sure the database you are upgrading has been cleanly shut down
2. set the Neo4j configuration parameter "allow_store_upgrade=true"
3. start the database
4. the upgrade will happen during startup and the process is done when the database has been successfully started
5. "allow_store_upgrade=true" configuration parameter should be removed, set to "false" or commented out

1.4.3. Upgrade 1.3 –> 1.4

Warning

Upgrading from 1.3 –> 1.4 is done automatically, but will in that process upgrade Lucene indexes to Lucene version 3.1.0. That Lucene index upgrade is irreversible.

1.4.4. Upgrade 1.3.M03 –> 1.3.M04

Warning

Upgrading from 1.3.M03 –> 1.3.M04 must be done explicitly since store format has changed between those two versions.

The store format, as well as logical log format, have changed between these two versions to allow for bigger stores.

1.4.5. Upgrade 1.2 –> 1.3

Warning

Upgrading from 1.2 –> 1.3 must be done explicitly since store format has changed between those two versions.

The store format, as well as logical log format, have changed between these two versions to allow for bigger stores.

Important

Although id ranges has been increased the space used to store the database will not increase compared to the previous version.

Upgrading between these two version needs to be performed explicitly using a configuration parameter at startup (see "Special Upgrade").

Caution

Upgrade cannot be performed if either the number of relationship types or the configured block size for either the dynamic array store or string store is greater than 65534.

Caution

Indexes created using the old IndexService/LuceneIndexService are no longer accessible out of the box in 1.3 in favor of the integrated index. An automatic upgrade isn’t possible so a full rebuild of the index data into the integrated index framework is required.
For reference the legacy index can be downloaded from the Neo4j repository, http://m2.neo4j.org/org/neo4j/neo4j-legacy-index/

1.4.6. Upgrade 1.1 –> 1.2

Upgrading from Neo4j 1.1 to Neo4j 1.2 is a "normal" upgrade.

1.5. Usage Data Collector

The Neo4j Usage Data Collector is a sub-system that gathers usage data, reporting it to the UDC-server at udc.neo4j.org. It is easy to disable, and does not collect any data that is confidential. For more information about what is being sent, see below.

The Neo4j team uses this information as a form of automatic, effortless feedback from the Neo4j community. We want to verify that we are doing the right thing by matching download statistics with usage statistics. After each release, we can see if there is a larger retention span of the server software.

The data collected is clearly stated here. If any future versions of this system collect additional data, we will clearly announce those changes.

The Neo4j team is very concerned about your privacy. We do not disclose any personally identifiable information.

1.5.1. Technical Information

To gather good statistics about Neo4j usage, UDC collects this information:

- Kernel version - the build number, and if there are any modifications to the kernel.
- Store id - it is a randomized globally unique id created at the same time a database is created.
- Ping count - UDC holds an internal counter which is incremented for every ping, and reset for every restart of the kernel.
- Source - this is either "neo4j" or "maven". If you downloaded Neo4j from the Neo4j website, it’s "neo4j", if you are using Maven to get Neo4j, it will be "maven".
- Java version - the referrer string shows which version of Java is being used.

After startup, UDC waits for ten minutes before sending the first ping. It does this for two reasons; first, we don’t want the startup to be slower because of UDC, and secondly, we want to keep pings from automatic tests to a minimum. The ping to the UDC servers is done with a HTTP GET.

1.5.2. How to disable UDC

We’ve tried to make it extremely easy to disable UDC. In fact, the code for UDC is not even included in the kernel jar but as a completely separate component.

There are three ways you can disable UDC:

1. The easiest way is to just remove the neo4j-udc-*.jar file. By doing this, the kernel will not load UDC, and no pings will be sent.
2. If you are using Maven, and want to make sure that UDC is never installed in your system, a dependency element like this will do that:

```
<dependency>
  <groupId>org.neo4j</groupId>
  <artifactId>neo4j</artifactId>
  <version>${neo4j-version}</version>
  <exclusions>
    <exclusion>
      <groupId>org.neo4j</groupId>
      <artifactId>neo4j-udc</artifactId>
    </exclusion>
  </exclusions>
</dependency>
```

Where ${neo4j-version} is the Neo4j version in use.

3. Lastly, if you are using a packaged version of Neo4j, and do not want to make any change to the jars, a system property setting like this will also make sure that UDC is never activated: `-Dneo4j.ext.udc.disable=true`.

Chapter 2. Configuration & Performance

In order to get optimum performance out of Neo4j for your application there are a few parameters that can be tweaked. The two main components that can be configured are the Neo4j caches and the JVM that Neo4j runs in. The following sections describe how to tune these.

2.1. Caches in Neo4j

Neo4j utilizes two different types of caches: A file buffer cache and an object cache. The file buffer cache caches the storage file data in the same format as it is stored on the durable storage media. The object cache caches the nodes, relationships and properties in a format that is optimized for high traversal speeds and transactional mutation.

2.1.1. File buffer cache

Quick info

- The file buffer cache is sometimes called *low level cache* or *file system cache*.
- It caches the Neo4j data as stored on the durable media.
- It uses the operating system memory mapping features when possible.
- Neo4j will configure the cache automatically as long as the heap size of the JVM is configured properly.

The file buffer cache caches the Neo4j data in the same format as it is represented on the durable storage media. The purpose of this cache layer is to improve both read and write performance. The file buffer cache improves write performance by writing to the cache and deferring durable write until the logical log is rotated. This behavior is safe since all transactions are always durably written to the logical log, which can be used to recover the store files in the event of a crash.

Since the operation of the cache is tightly related to the data it stores, a short description of the Neo4j durable representation format is necessary background. Neo4j stores data in multiple files and relies on the underlying file system to handle this efficiently. Each Neo4j storage file contains uniform fixed size records of a particular type:

Store file	Record size	Contents
nodestore	`9 B`	Nodes
relstore	`33 B`	Relationships
propstore	`25 B`	Properties for nodes and relationships
stringstore	`133 B`	Values of string properties
arraystore	`133 B`	Values of array properties

For strings and arrays, where data can be of variable length, data is stored in one or more 120B chunks, with 13B record overhead. The sizes of these blocks can actually be configured when the store is created using the `string_block_size` and `array_block_size` parameters. The size of each record type can also be used to calculate the storage requirements of a Neo4j graph or the appropriate cache size for each file buffer cache. Note that some strings can be stored without using the string store, see Section 2.4, “Compressed storage of short strings”.

Neo4j uses multiple file buffer caches, one for each different storage file. Each file buffer cache divides its storage file into a number of equally sized windows. Each cache window contains an even number of storage records. The cache holds the most active cache windows in memory and tracks hit vs. miss ratio for the windows. When the hit ratio of an uncached window gets higher than the miss ratio of a cached window, the cached window gets evicted and the previously uncached window is cached instead.

Configuration

Parameter	Possible values	Effect
`use_memory_mapped_buffers`	`true` or `false`	If set to `true` Neo4j will use the operating systems memory mapping functionality for the file buffer cache windows. If set to `false` Neo4j will use its own buffer implementation. In this case the buffers will reside in the JVM heap which needs to be increased accordingly. The default value for this parameter is `true`, except on Windows.
`neostore.nodestore.db.mapped_memory`	The maximum amount of memory to use for memory mapped buffers for this file buffer cache. The default unit is `MiB`, for other units use any of the following suffixes: `B`, `k`, `M` or `G`.	The maximum amount of memory to use for the file buffer cache of the node storage file.
`neostore.relationshipstore.db.mapped_memory`		The maximum amount of memory to use for the file buffer cache of the relationship store file.
`neostore.propertystore.db.index.keys.mapped_memory`		The maximum amount of memory to use for the file buffer cache of the something-something file.
`neostore.propertystore.db.index.mapped_memory`		The maximum amount of memory to use for the file buffer cache of the something-something file.
`neostore.propertystore.db.mapped_memory`		The maximum amount of memory to use for the file buffer cache of the property storage file.
`neostore.propertystore.db.strings.mapped_memory`		The maximum amount of memory to use for the file buffer cache of the string property storage file.
`neostore.propertystore.db.arrays.mapped_memory`		The maximum amount of memory to use for the file buffer cache of the array property storage file.
`string_block_size`	The number of bytes per block.	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 a block size of 120 (the default size) will hold a 60 character long string

Parameter	Possible values	Effect
		before overflowing into a second block. Also note that each block carries an overhead of 13 bytes. This means that if the block size is 120, the size of the stored records will be 133 bytes.
`array_block_size`		Specifies the block size for storing arrays. This parameter is only honored when the store is created, otherwise it is ignored. The default block size is 120 bytes, and the overhead of each block is the same as for string blocks, i.e., 13 bytes.
`dump_configuration`	`true` or `false`	If set to `true` the current configuration settings will be written to the default system output, mostly the console or the logfiles.

When memory mapped buffers are used (`use_memory_mapped_buffers = true`) the heap size of the JVM must be smaller than the total available memory of the computer, minus the total amount of memory used for the buffers. When heap buffers are used (`use_memory_mapped_buffers = false`) the heap size of the JVM must be large enough to contain all the buffers, plus the runtime heap memory requirements of the application and the object cache.

When reading the configuration parameters on startup Neo4j will automatically configure the parameters that are not specified. The cache sizes will be configured based on the available memory on the computer, how much is used by the JVM heap, and how large the storage files are.

2.1.2. Object cache

Quick info

- The object cache is sometimes called *high level cache*.
- It caches the Neo4j data in a form optimized for fast traversal.

The object cache caches individual nodes and relationships and their properties in a form that is optimized for fast traversal of the graph. The content of this cache are objects with a representation geared towards supporting the Neo4j object API and graph traversals. Reading from this cache is 5 to 10 times faster than reading from the file buffer cache. This cache is contained in the heap of the JVM and the size is adapted to the current amount of available heap memory.

Nodes and relationships are added to the object cache as soon as they are accessed. The cached objects are however populated lazily. The properties for a node or relationship are not loaded until properties are accessed for that node or relationship. String (and array) properties are not loaded until that particular property is accessed. The relationships for a particular node is also not loaded until the relationships are accessed for that node. Eviction from the cache happens in an LRU manner when the memory is needed.

Configuration

The main configuration parameter for the object cache is the `cache_type` parameter. This specifies which cache implementation to use for the object cache. The available cache types are:

`cache_type`	Description
`none`	Do not use a high level cache. No objects will be cached.
`soft`	Provides optimal utilization of the available memory. Suitable for high performance traversal. May run into GC issues under high load if the frequently accessed parts of the graph does not fit in the cache. This is the default cache implementation.
`weak`	Provides short life span for cached objects. Suitable for high throughput applications where a larger portion of the graph than what can fit into memory is frequently accessed.
`strong`	This cache will cache **all data** in the **entire graph**. It will never release memory held by the cache. Provides optimal performance if your graph is small enough to fit in memory.

Heap memory usage

This table can be used to calculate how much memory the data in the object cache will occupy on a 64bit JVM:

Object	Size	Comment
Node	`344 B`	*Size for each node (not counting its relationships or properties).*
	`48 B`	*Object overhead.*
	`136 B`	*Property storage (ArrayMap `48B`, HashMap `88B`).*
	`136 B`	*Relationship storage (ArrayMap `48B`, HashMap `88B`).*
	`24 B`	*Location of first / next set of relationships.*
Relationship	`208 B`	*Size for each relationship (not counting its properties).*
	`48 B`	*Object overhead.*
	`136 B`	*Property storage (ArrayMap `48B`, HashMap `88B`).*
Property	`116 B`	*Size for each property of a node or relationship.*
	`32 B`	*Data element - allows for transactional modification and keeps track of on disk location.*
	`48 B`	*Entry in the hash table where it is stored.*
	`12 B`	*Space used in hash table, accounts for normal fill ratio.*
	`24 B`	*Property key index.*
Relationships	`108 B`	*Size for each relationship type for a node that has a relationship of that type.*
	`48 B`	*Collection of the relationships of this type.*
	`48 B`	*Entry in the hash table where it is stored.*
	`12 B`	*Space used in hash table, accounts for normal fill ratio.*

Object	**Size**	**Comment**
Relationships	`8 B`	*Space used by each relationship related to a particular node (both incoming and outgoing).*
Primitive	`24 B`	*Size of a primitive property value.*
String	`64+B`	*Size of a string property value.* `64 + 2*len(string) B` *(64 bytes, plus two bytes for each character in the string).*

2.2. JVM Settings

Properly configuring memory utilization of the JVM is crucial for optimal performance. As an example, a poorly configured JVM could spend all CPU time performing garbage collection (blocking all threads from performing any work). Requirements such as latency, total throughput and available hardware have to be considered to find the right setup. In production, Neo4j should run on a multi core/CPU platform with the JVM in server mode.

2.2.1. Configuring heap size and GC

A large heap allows for larger node and relationship caches — which is a good thing — but large heaps can also lead to latency problems caused by full garbage collection. The different high level cache implementations available in Neo4j together with a suitable JVM configuration of heap size and garbage collection (GC) should be able to handle most workloads.

The default cache (soft reference based LRU cache) works best with a heap that never gets full: a graph where the most used nodes and relationships can be cached. If the heap gets too full there is a risk that a full GC will be triggered; the larger the heap, the longer it can take to determine what soft references should be cleared.

Using the strong reference cache means that *all* the nodes and relationships being used must fit in the available heap. Otherwise there is a risk of getting out-of-memory exceptions. The soft reference and strong reference caches are well suited for applications were the overal throughput is important.

The weak reference cache basically needs enough heap to handle the peak load of the application — peak load multiplied by the average memory required per request. It is well suited for low latency requirements were GC interuptions are not acceptable.

When running Neo4j on Windows, keep in mind that the memory mapped buffers are allocated on heap by default, so need to be taken into consideration when determining heap size.

Table 2.1. Guidelines for heap size

Number of primitives	RAM size	Heap configuration	Reserved RAM for the OS
10M	2GB	512MB	the rest
100M	8GB+	1-4GB	1-2GB
1B+	16GB-32GB+	4GB+	1-2GB

The recommended garbage collector to use when running Neo4j in production is the Concurrent Mark and Sweep Compactor turned on by supplying -XX:+UseConcMarkSweepGC as a JVM parameter.

2.3. File system tuning for high IO

In order to support the high IO load of small transactions from a database, the underlying file system should be tuned. Symptoms for this are low CPU load with high iowait. In this case, there are a couple of tweaks possible on Linux systems:

- Disable access-time updates: `noatime,nodiratime` flags for disk mount command or in the */etc/fstab* for the database disk volume mount.
- Tune the IO scheduler for high disk IO on the database disk.

2.4. Compressed storage of short strings

Neo4j will classify your strings and store them accordingly. If a string is classified as a short string it will be stored without indirection in the property store. This means that there will be no string records created for storing that string. Additionally, when no string record is needed to store the property, it can be read and written in a single lookup. This leads to improvements in performance and lower storage overhead.

For a string to be classified as a short string, one of the following must hold:

- It is encodable in UTF-8 or Latin-1, 7 bytes or less.
- It is alphanumerical, and 10 characters or less (9 if using accented european characters).
- It consists of only upper case, or only lower case characters, including the punctuation characters space, underscore, period, dash, colon, or slash. Then it is allowed to be up to 12 characters.
- It consists of only numerical characters, inlcuding the punctuation characters plus, comma, single quote, space, period, or dash. Then it is allowed to be up to 15 characters.

Chapter 3. 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 modifications to Neo4j data must be wrapped in transactions.
- 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.

3.1. Interaction cycle

All write operations that work with the graph 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.

When working with transactions the interaction cycle looks like this:

1. Begin a transaction.
2. Operate on the graph performing write 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 write 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.

Caution

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.

3.2. Isolation levels

By default a read operation will read the last committed value unless a local modification within the current transaction exist. The default isolation level is very similar to `READ_COMMITTED`: reads do not block or take any locks so non-repeatable reads can occur. It is possible to achieve a stronger isolation level (such as `REPETABLE_READ` and `SERIALIZABLE`) by manually acquiring read and write locks.

3.3. 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.

3.4. 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.

Important

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.

3.5. 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.

Caution

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.

Chapter 4. Cypher Query Language

Caution

This is an experimental feature.

A new query language, code-named “Cypher”, has been added to Neo4j. It allows for expressive and efficient querying of the graph store without having to write traversers in code.

Cypher is designed to be a humane query language, suitable for both developers and (importantly, we think) operations professionals who want to make ad-hoc queries on the database. Its constructs are based on English prose and neat iconography, which helps to make it (somewhat) self-explanatory.

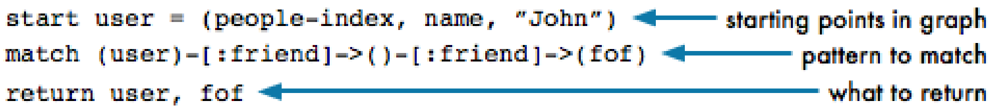
Cypher is inspired by a number of different approaches and builds upon established practices for expressive querying. Most of the keywords like `WHERE` and `ORDER BY` are inspired by SQL <http://en.wikipedia.org/wiki/SQL>. Pattern matching borrows expression approaches from SPARQL <http://en.wikipedia.org/wiki/SPARQL>. Regular expression matching is implemented using the Scala programming language <http://www.scala-lang.org/>.

Cypher is a declarative language. It focuses on the clarity of expressing *what* to retrieve from a graph, not *how* to do it, in contrast to imperative languages like Java, and scripting languages like Gremlin <http://gremlin.tinkerpop.com> (supported via the Section 6.13, “Gremlin Plugin”) and the JRuby Neo4j bindings <http://neo4j.rubyforge.org/>. This makes the concern of how to optimize queries in implementation detail not exposed to the user.

The query language is comprised of several distinct parts.

Let’s see three of them in action:

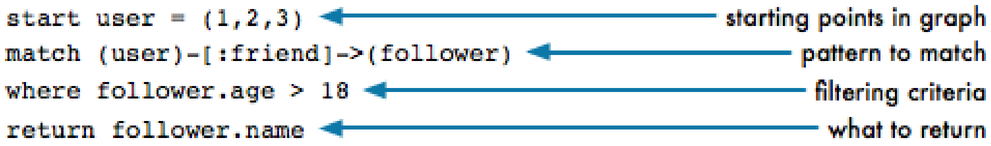
For example, here is a query which finds a user called John in an index and then traverses the graph looking for friends of Johns friends (though not his direct friends) before returning both John and any friends-of-friends that are found.



```
start user = (people-index, name, "John")
match (user)-[:friend]->()-[:friend]->(fof)
return user, fof
```

Next up we will add filtering to set all four parts in motion:

In this next example, we take a list of users (by node ID) and traverse the graph looking for those other users that have an outgoing `friend` relationship, returning only those followed users who are older than 18.



```
start user = (1,2,3)
match (user)-[:friend]->(follower)
where follower.age > 18
return follower.name
```

In Java, using the query language looks something like this:

```
db = new ImpermanentGraphDatabase();
```

```
engine = new ExecutionEngine( db );
CypherParser parser = new CypherParser();
ExecutionEngine engine = new ExecutionEngine(db);
Query query = parser.parse( "start n=(0) where 1=1 return n" );
ExecutionResult result = engine.execute( query );

assertThat( result.columns(), hasItem( "n" ) );
Iterator<Node> n_column = result.columnAs( "n" );
assertThat( asIterable( n_column ), hasItem(db.getNodeById(0)) );
assertThat( result.toString(), containsString("Node[0]") );
```

4.1. Identifiers

When you reference parts of the pattern, you do so by naming them.

```
start identifier=(0) return identifier
```

Identifiers can be lower or upper case, and may contain underscore. If other characters are needed, you can use the ` sign. The same rules apply to property names.

```
start a=(0) return a.`property.with.periods`
```

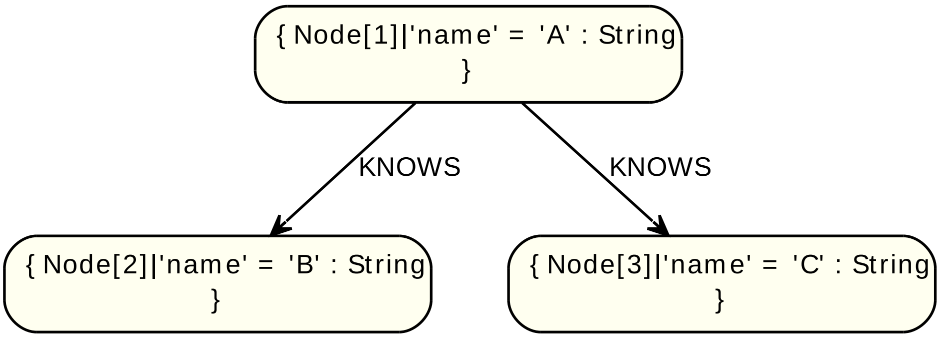
4.2. Start

Every query describes a pattern, and in that pattern one can have multiple bound points. A bound point is a relationship or a node that form the starting points for a pattern match. You can either bind points by id, or by index lookups.

4.2.1. Node by id

Including a node as a start point is done by using parenthesis.

Graph


{ Node[1]|'name' = 'A' : String }
KNOWS
KNOWS
{ Node[2]|'name' = 'B' : String }
{ Node[3]|'name' = 'C' : String }

Query

```
start n=(1) return n
```

The reference node is returned

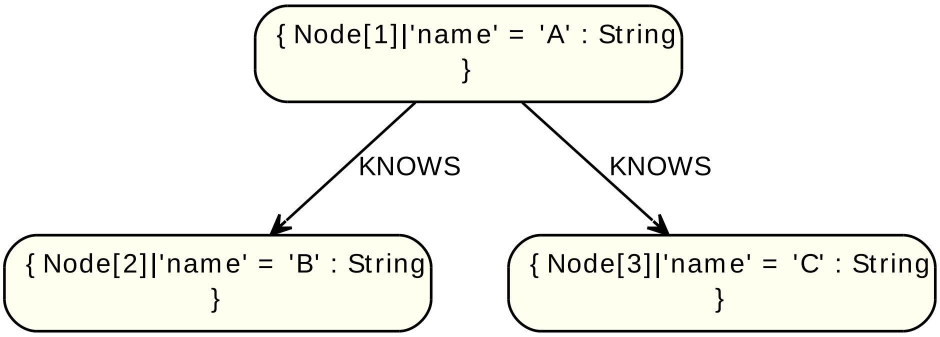
Result

```
+--------------------+
| n                  |
+--------------------+
| Node[1]{name->"A"} |
+--------------------+
1 rows, 1 ms
```

4.2.2. Multiple nodes by id

Multiple nodes are selected by listing them separated by commas.

Graph


{ Node[1]|'name' = 'A' : String }
KNOWS
KNOWS
{ Node[2]|'name' = 'B' : String }
{ Node[3]|'name' = 'C' : String }

Query

```
start n=(1, 2, 3) return n
```

The nodes listed in the START statement.

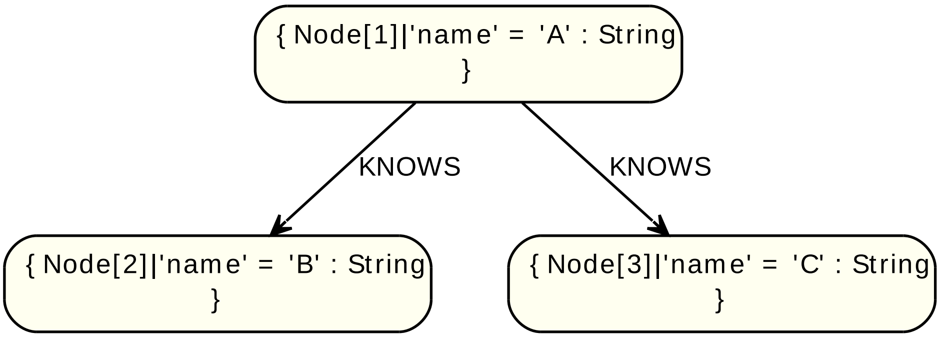
Result

```
+--------------------+
| n                  |
+--------------------+
| Node[1]{name->"A"} |
| Node[2]{name->"B"} |
| Node[3]{name->"C"} |
+--------------------+
3 rows, 0 ms
```

4.2.3. Node by index lookup

If the start point can be found by index lookups, it can be done like this: (index-name, key, "value"). Like this:

Graph


{ Node[1]|'name' = 'A' : String }
KNOWS
KNOWS
{ Node[2]|'name' = 'B' : String }
{ Node[3]|'name' = 'C' : String }

Query

```
start n=(nodes,name,"A") return n
```

The node indexed with name "A" is returned

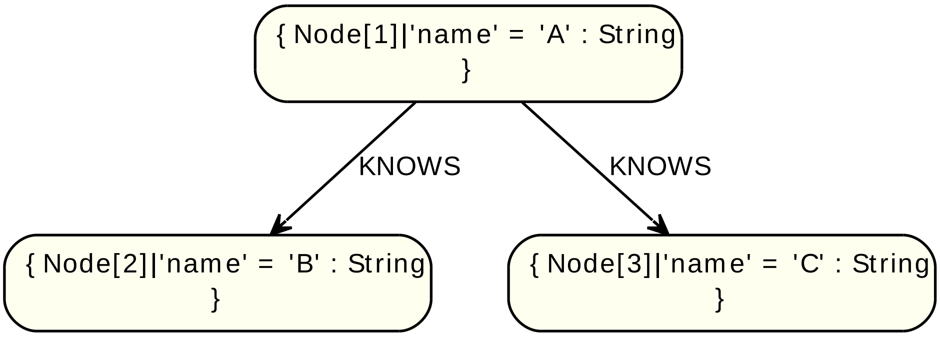
Result

```
+--------------------+
| n                  |
+--------------------+
| Node[1]{name->"A"} |
+--------------------+
1 rows, 1 ms
```

4.2.4. Node by index query

If the start point can be found by index queries, it can be done like this: (index-name, "query").This allows you to write more advanced index queries

Graph


{ Node[1]|'name' = 'A' : String }
KNOWS
KNOWS
{ Node[2]|'name' = 'B' : String }
{ Node[3]|'name' = 'C' : String }

Query

```
start n=(nodes,"name:A") return n
```

The node indexed with name "A" is returned

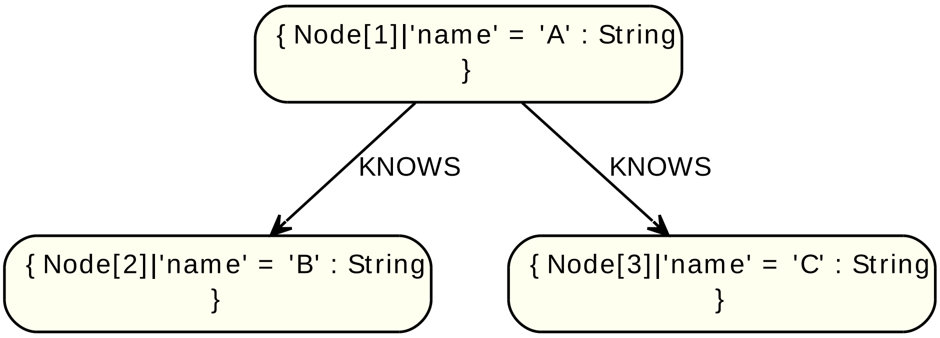
Result

```
+--------------------+
| n                  |
+--------------------+
| Node[1]{name->"A"} |
+--------------------+
1 rows, 1 ms
```

4.2.5. Multiple start points

Sometimes you want to bind multiple start points. Just list them separated by commas.

Graph


{ Node[1]|'name' = 'A' : String }
KNOWS
KNOWS
{ Node[2]|'name' = 'B' : String }
{ Node[3]|'name' = 'C' : String }

Query

```
start a=(1), b=(2) return a,b
```

Both the A and the B node are returned

Result

```
+-----------------------------------------+
| a                  | b                  |
+-----------------------------------------+
| Node[1]{name->"A"} | Node[2]{name->"B"} |
+-----------------------------------------+
1 rows, 1 ms
```

4.3. Match

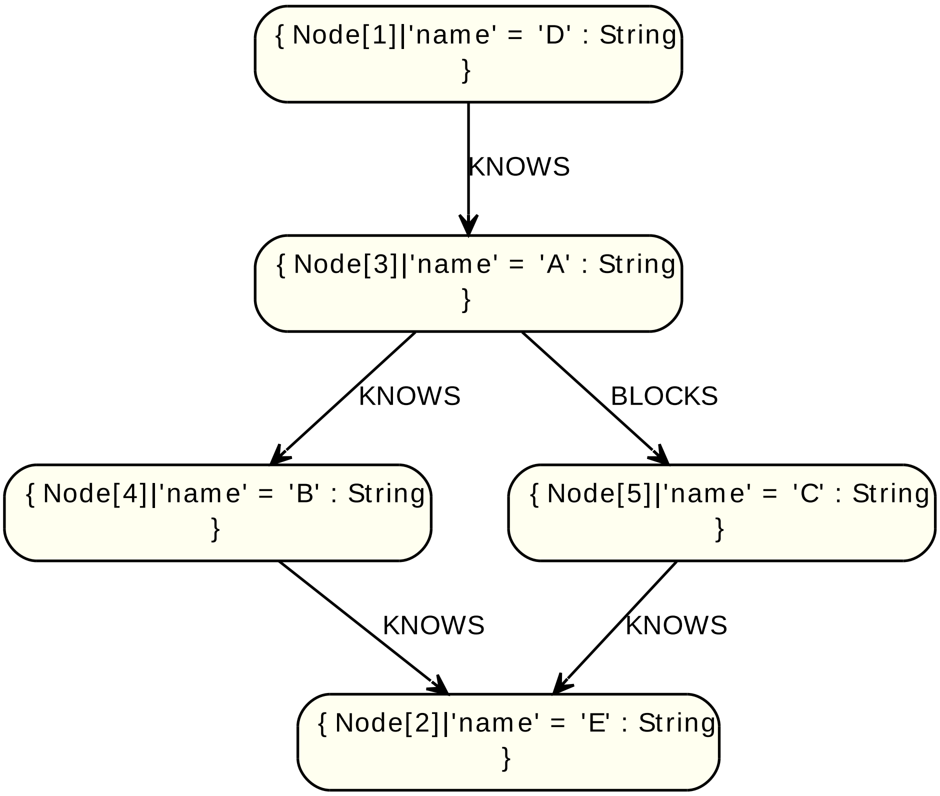
In the match part of a query, the pattern is described. The description of the pattern is made up of one or more paths, separated by commas.

All parts of the pattern must be directly or indirectly bound to a start point.

4.3.1. Related nodes

The symbol — means related to, without regard to type or direction.

Graph


{ Node[1]|'name' = 'D' : String }
KNOWS
{ Node[3]|'name' = 'A' : String }
KNOWS
BLOCKS
{ Node[4]|'name' = 'B' : String }
{ Node[5]|'name' = 'C' : String }
KNOWS
KNOWS
{ Node[2]|'name' = 'E' : String }

Query

```
start n=(3) match (n)--(x) return x
```

All nodes related to A are returned

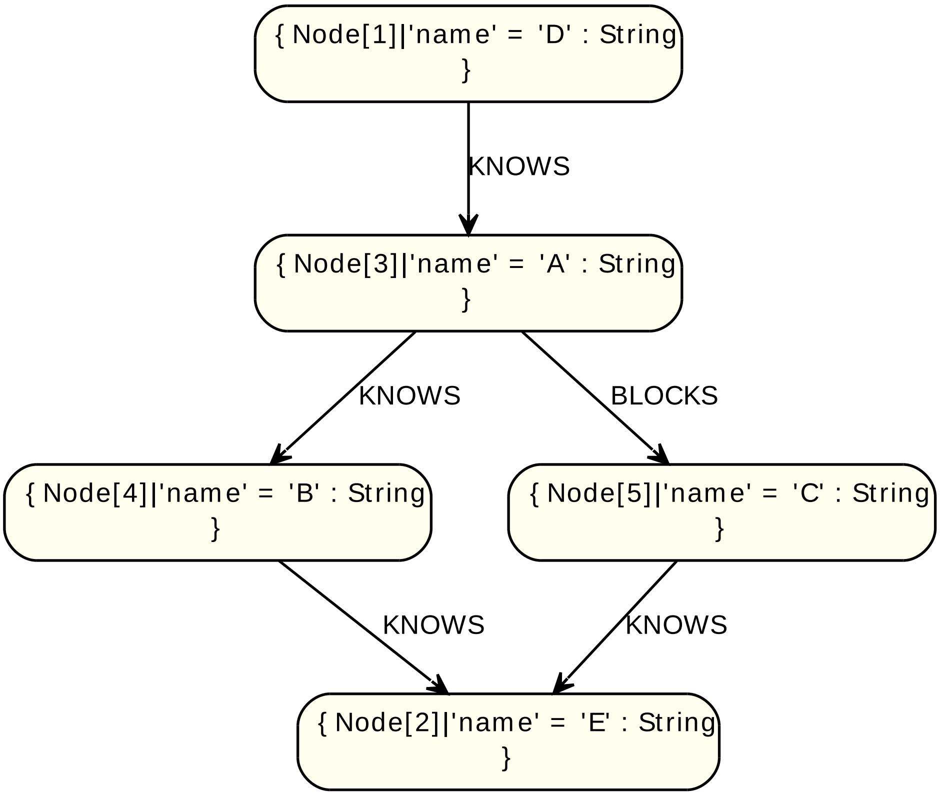
Result

```
+--------------------+
| x                  |
+--------------------+
| Node[4]{name->"B"} |
| Node[1]{name->"D"} |
| Node[5]{name->"C"} |
+--------------------+
3 rows, 2 ms
```

4.3.2. Outgoing relationships

When the direction of a relationship is interesting, it is shown by using `-->` or `<--`, like this:

Graph


{ Node[1]|'name' = 'D' : String }
KNOWS
{ Node[3]|'name' = 'A' : String }
KNOWS
BLOCKS
{ Node[4]|'name' = 'B' : String }
{ Node[5]|'name' = 'C' : String }
KNOWS
KNOWS
{ Node[2]|'name' = 'E' : String }

Query

```
start n=(3) match (n)-->(x) return x
```

All nodes that A has outgoing relationships to.

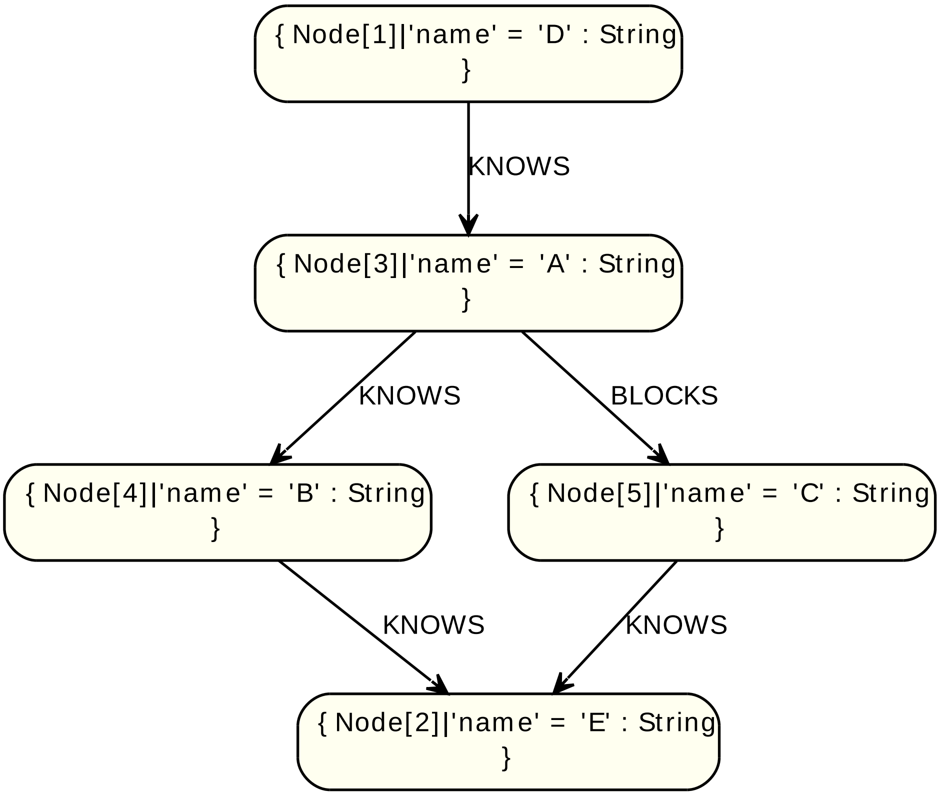
Result

```
+--------------------+
| x                  |
+--------------------+
| Node[4]{name->"B"} |
| Node[5]{name->"C"} |
+--------------------+
2 rows, 2 ms
```

4.3.3. Directed relationships and identifier

If an identifier is needed, either for filtering on properties of the relationship, or to return the relationship, this is how you introduce the identifier.

Graph


{ Node[1]|'name' = 'D' : String }
KNOWS
{ Node[3]|'name' = 'A' : String }
KNOWS
BLOCKS
{ Node[4]|'name' = 'B' : String }
{ Node[5]|'name' = 'C' : String }
KNOWS
KNOWS
{ Node[2]|'name' = 'E' : String }

Query

```
start n=(3) match (n)-[r]->() return r
```

All outgoing relationships from node A.

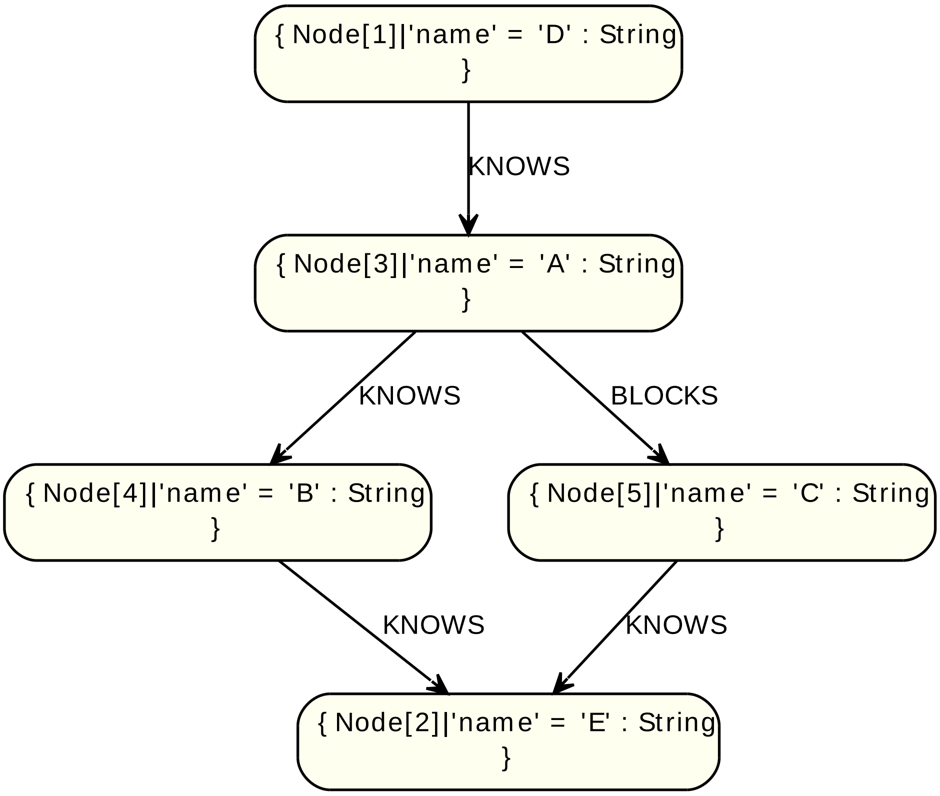
Result

```
+---------------+
| r             |
+---------------+
| :KNOWS[0] {}  |
| :BLOCKS[1] {} |
+---------------+
2 rows, 1 ms
```

4.3.4. Match by relationship type

When you know the relationship type you want to match on, you can specify it by using a colon.

Graph


{ Node[1]|'name' = 'D' : String }
KNOWS
{ Node[3]|'name' = 'A' : String }
KNOWS
BLOCKS
{ Node[4]|'name' = 'B' : String }
{ Node[5]|'name' = 'C' : String }
KNOWS
KNOWS
{ Node[2]|'name' = 'E' : String }

Query

```
start n=(3) match (n)-[:BLOCKS]->(x) return x
```

All nodes that are BLOCKed by A.

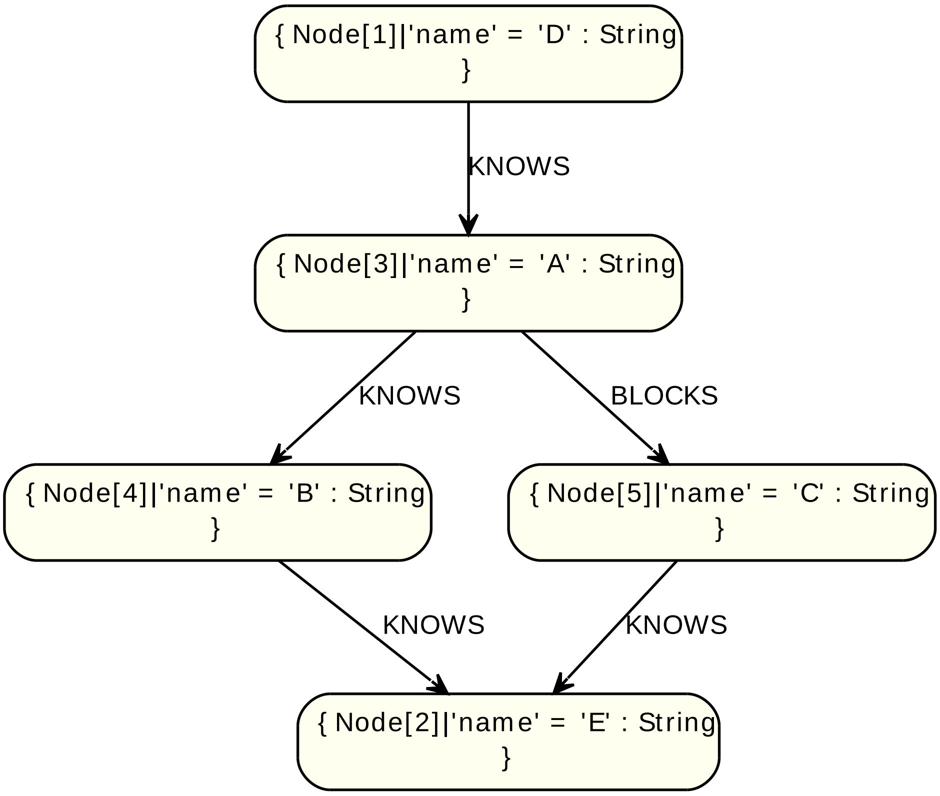
Result

```
+--------------------+
| x                  |
+--------------------+
| Node[5]{name->"C"} |
+--------------------+
1 rows, 1 ms
```

4.3.5. Match by relationship type and use an identifier

If you both want to introduce an identifier to hold the relationship, and specify the relationship type you want, just add them both, like this.

Graph


{ Node[1]|'name' = 'D' : String }
KNOWS
{ Node[3]|'name' = 'A' : String }
KNOWS
BLOCKS
{ Node[4]|'name' = 'B' : String }
{ Node[5]|'name' = 'C' : String }
KNOWS
KNOWS
{ Node[2]|'name' = 'E' : String }

Query

```
start n=(3) match (n)-[r:BLOCKS]->() return r
```

All BLOCK relationship going out from A.

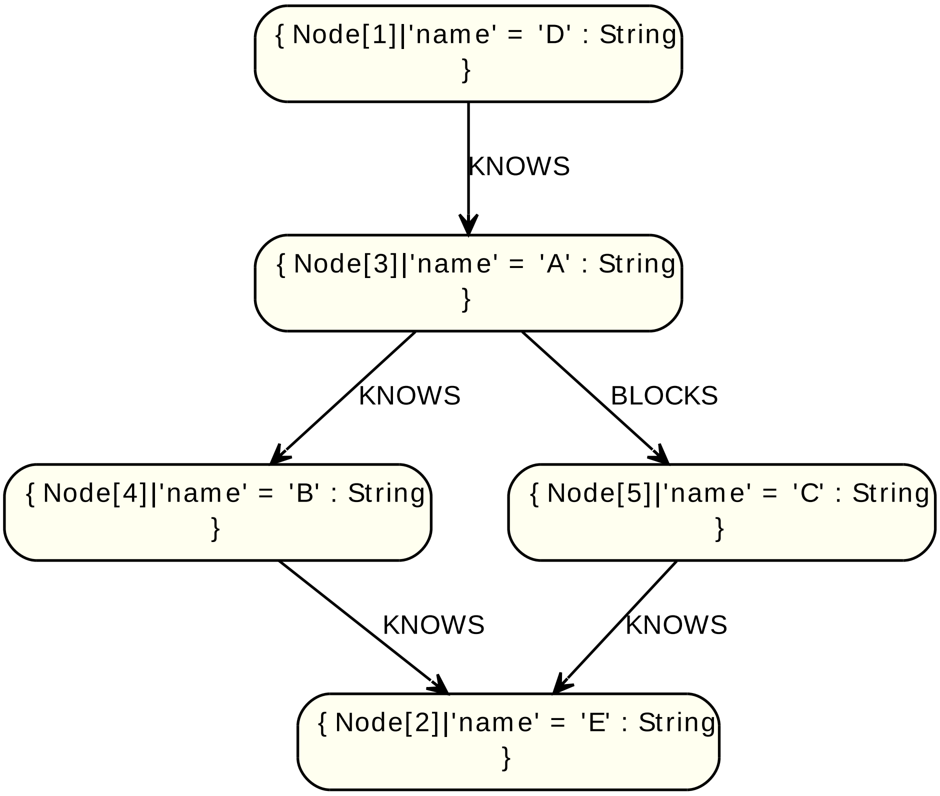
Result

```
+---------------+
| r             |
+---------------+
| :BLOCKS[1] {} |
+---------------+
1 rows, 1 ms
```

4.3.6. Multiple relationships

Relationships can be expressed by using multiple statements in the form of ()--(), or they can be stringed together, like this:

Graph


{ Node[1]|'name' = 'D' : String }
KNOWS
{ Node[3]|'name' = 'A' : String }
KNOWS
BLOCKS
{ Node[4]|'name' = 'B' : String }
{ Node[5]|'name' = 'C' : String }
KNOWS
KNOWS
{ Node[2]|'name' = 'E' : String }

Query

```
start a=(3) match (a)-[:KNOWS]->(b)-[:KNOWS]->(c) return a,b,c
```

The three nodes in the path.

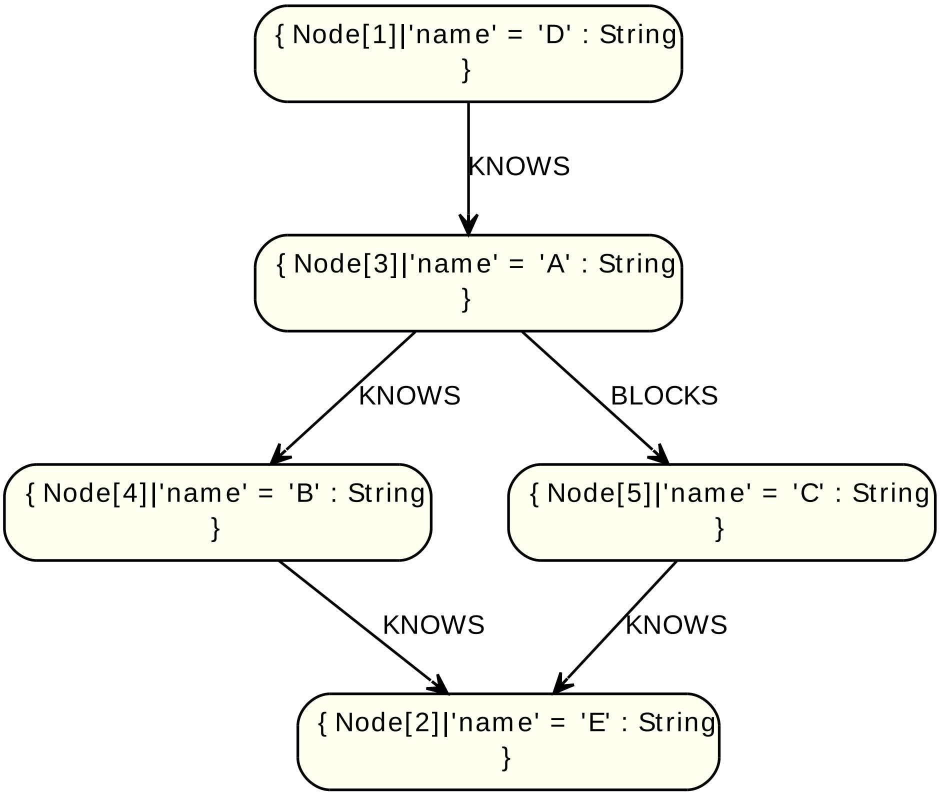
Result

```
+-------------------------------------------------------------+
| a                  | b                  | c                  |
+-------------------------------------------------------------+
| Node[3]{name->"A"} | Node[4]{name->"B"} | Node[2]{name->"E"} |
+-------------------------------------------------------------+
1 rows, 2 ms
```

4.3.7. Variable length relationships

Nodes that are variable number of relationship–>node hops can be found using the -[:TYPE^minHops..maxHops]–>.

Graph


{ Node[1]|'name' = 'D' : String }
KNOWS
{ Node[3]|'name' = 'A' : String }
KNOWS
BLOCKS
{ Node[4]|'name' = 'B' : String }
{ Node[5]|'name' = 'C' : String }
KNOWS
KNOWS
{ Node[2]|'name' = 'E' : String }

Query

```
start a=(3), x=(2, 4) match a-[:KNOWS^1..3]->x return a,x
```

The three nodes in the path.

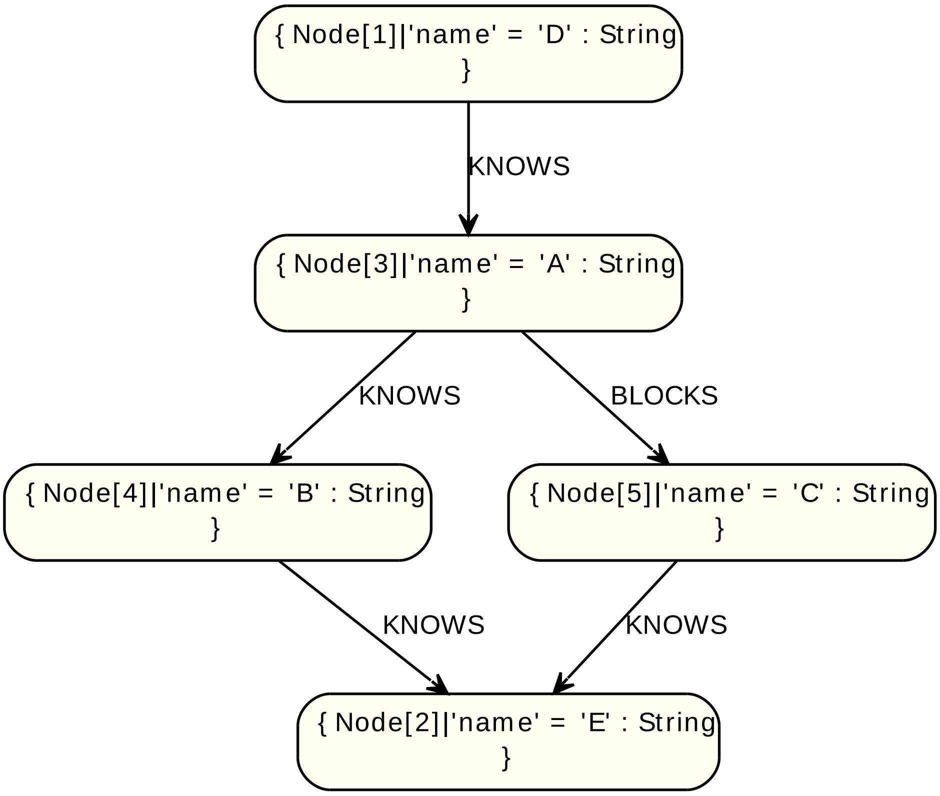
Result

```
+-----------------------------------------+
| a                  | x                  |
+-----------------------------------------+
| Node[3]{name->"A"} | Node[2]{name->"E"} |
| Node[3]{name->"A"} | Node[4]{name->"B"} |
+-----------------------------------------+
2 rows, 2 ms
```

4.3.8. Optional relationship

If a relationship is optional, it can be marked with a question mark. This similar to how a SQL outer join works, if the relationship is there, it is returned. If it’s not, null is returned in it’s place. Remember that anything hanging of an optional relation, is in turn optional, unless it is connected with a bound node some other path.

Graph


{ Node[1]|'name' = 'D' : String }
KNOWS
{ Node[3]|'name' = 'A' : String }
KNOWS
BLOCKS
{ Node[4]|'name' = 'B' : String }
{ Node[5]|'name' = 'C' : String }
KNOWS
KNOWS
{ Node[2]|'name' = 'E' : String }

Query

```
start a=(2) match a-[?]->x return a,x
```

A node, and null, since the node has no relationships.

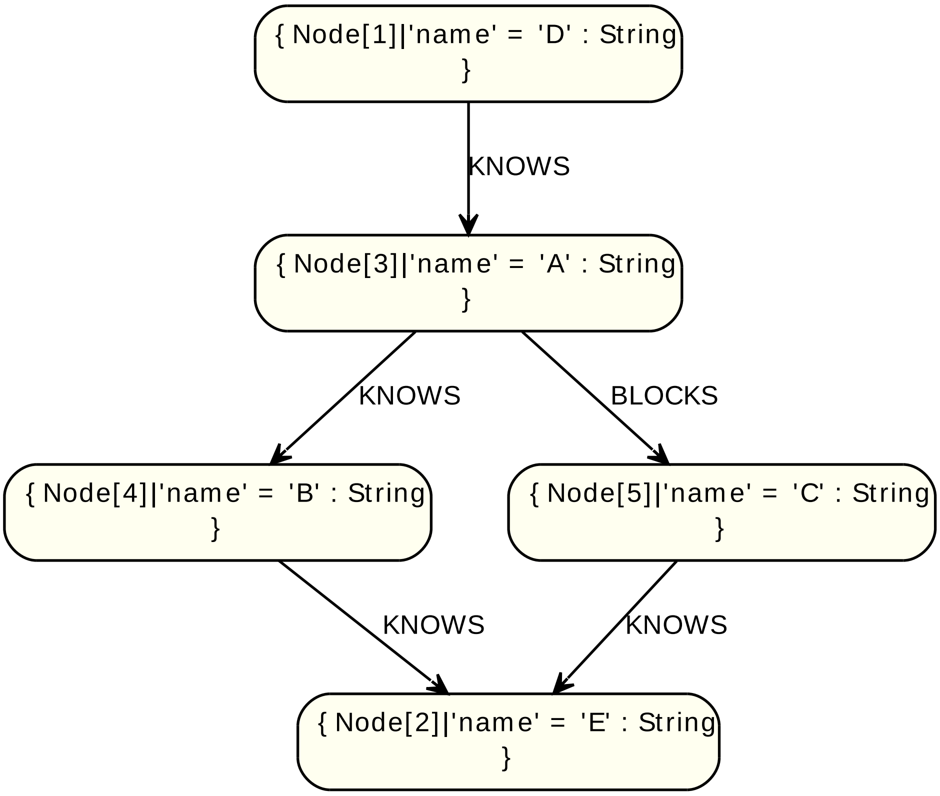
Result

```
+----------------------------+
| a                 | x      |
+----------------------------+
| Node[2]{name->"E"} | <null> |
+----------------------------+
1 rows, 1 ms
```

4.3.9. Optional typed and named relationship

Just as with a normal relationship, you can decide which identifier it goes into, and what relationship type you need.

Graph


{ Node[1]|'name' = 'D' : String }
KNOWS
{ Node[3]|'name' = 'A' : String }
KNOWS
BLOCKS
{ Node[4]|'name' = 'B' : String }
{ Node[5]|'name' = 'C' : String }
KNOWS
KNOWS
{ Node[2]|'name' = 'E' : String }

Query

```
start a=(3) match a-[r?:LOVES]->() return a,r
```

A node, and null, since the node has no relationships.

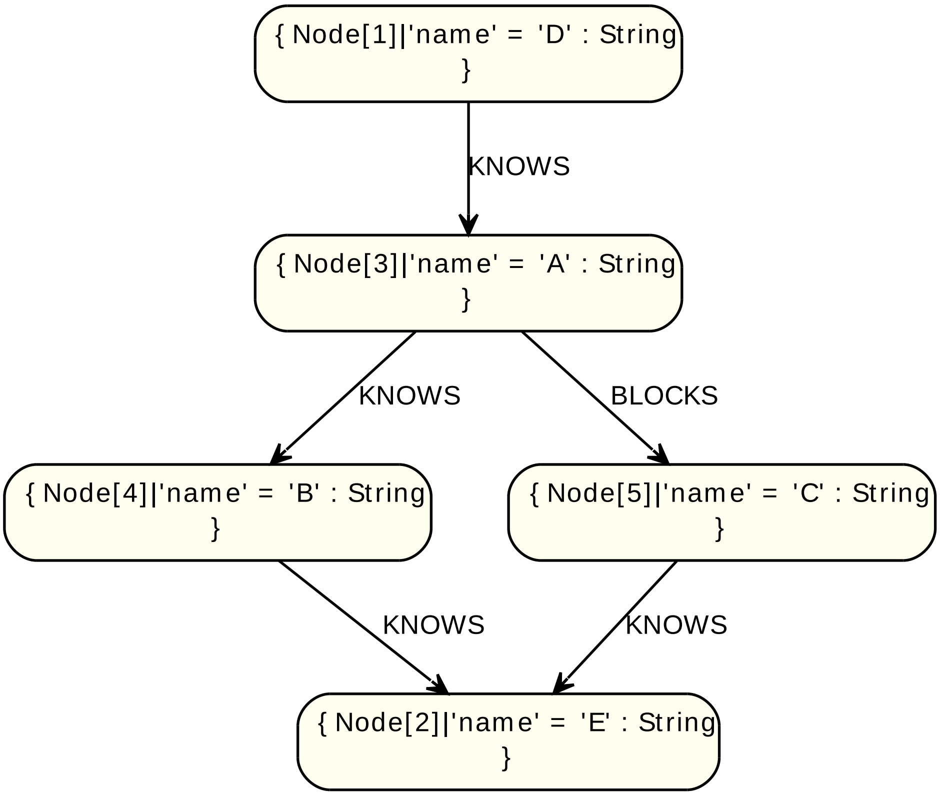
Result

```
+----------------------------+
| a                 | r      |
+----------------------------+
| Node[3]{name->"A"} | <null> |
+----------------------------+
1 rows, 1 ms
```

4.3.10. Complex matching

Using Cypher, you can also express more complex patterns to match on, like a diamond shape pattern.

Graph


{ Node[1]|'name' = 'D' : String }
KNOWS
{ Node[3]|'name' = 'A' : String }
KNOWS
BLOCKS
{ Node[4]|'name' = 'B' : String }
{ Node[5]|'name' = 'C' : String }
KNOWS
KNOWS
{ Node[2]|'name' = 'E' : String }

Query

```
start a=(3)
match (a)-[:KNOWS]->(b)-[:KNOWS]->(c), (a)-[:BLOCKS]-(d)-[:KNOWS]-(c)
return a,b,c,d
```

The four nodes in the path.

Result

```
+-----------------------------------------------------------------------------------+
| a                  | b                  | c                  | d                  |
+-----------------------------------------------------------------------------------+
| Node[3]{name->"A"} | Node[4]{name->"B"} | Node[2]{name->"E"} | Node[5]{name->"C"} |
+-----------------------------------------------------------------------------------+
1 rows, 3 ms
```

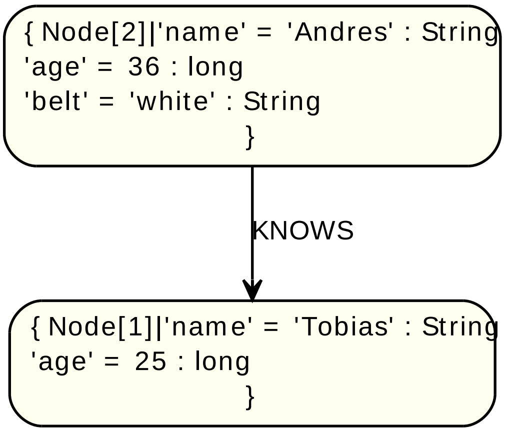
4.4. Where

If you need filtering apart from the pattern of the data that you are looking for, you can add clauses in the where part of the query.

4.4.1. Boolean operations

You can use the expected boolean operators AND and OR, and also the boolean function NOT().

Graph


{ Node[2]|'name' = 'Andres' : String
'age' = 36 : long
'belt' = 'white' : String
}
KNOWS
{ Node[1]|'name' = 'Tobias' : String
'age' = 25 : long
}

Query

```
start n=(2, 1) where (n.age < 30 and n.name = "Tobias") or not(n.name = "Tobias")  return n
```

The node.

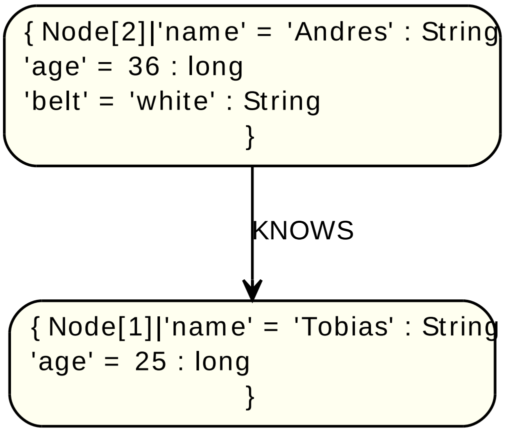
Result

```
+-----------------------------------------------+
| n                                             |
+-----------------------------------------------+
| Node[2]{name->"Andres",age->36,belt->"white"} |
| Node[1]{name->"Tobias",age->25}               |
+-----------------------------------------------+
2 rows, 0 ms
```

4.4.2. Filter on node property

To filter on a property, write your clause after the WHERE keyword.

Graph


{ Node[2]|'name' = 'Andres' : String
'age' = 36 : long
'belt' = 'white' : String
}
KNOWS
{ Node[1]|'name' = 'Tobias' : String
'age' = 25 : long
}

Query

```
start n=(2, 1) where n.age < 30 return n
```

The node.

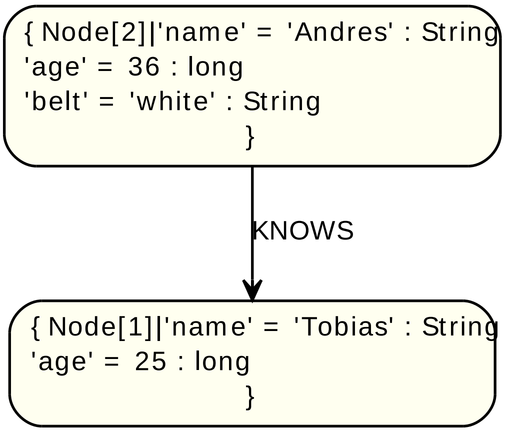
Result

```
+---------------------------------+
| n                               |
+---------------------------------+
| Node[1]{name->"Tobias",age->25} |
+---------------------------------+
1 rows, 1 ms
```

4.4.3. Regular expressions

You can match on regular expressions by using =~ /regexp/, like this:

Graph


{ Node[2]|'name' = 'Andres' : String
'age' = 36 : long
'belt' = 'white' : String
}
KNOWS
{ Node[1]|'name' = 'Tobias' : String
'age' = 25 : long
}

Query

```
start n=(2, 1) where n.name =~ /Tob.*/ return n
```

The node named Tobias.

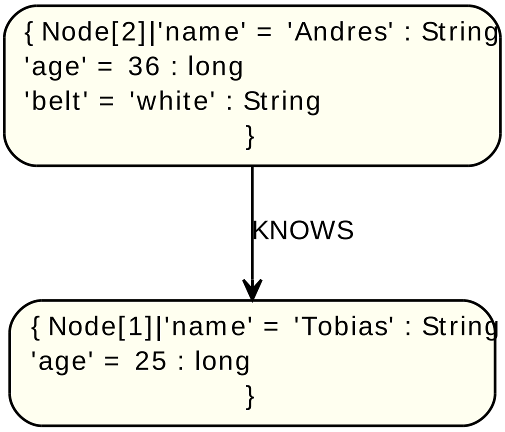
Result

```
+---------------------------------+
| n                               |
+---------------------------------+
| Node[1]{name->"Tobias",age->25} |
+---------------------------------+
1 rows, 0 ms
```

4.4.4. Filtering on relationship type

You can put the exact relationship type in the MATCH pattern, but sometimes you want to be able to do more advanced filtering on the type. You can use the special property TYPE to compare the type with something else. In this example, the query does a regular expression comparison with the name of the relationship type.

Graph


{ Node[2]|'name' = 'Andres' : String
'age' = 36 : long
'belt' = 'white' : String
}
KNOWS
{ Node[1]|'name' = 'Tobias' : String
'age' = 25 : long
}

Query

```
start n=(2) match (n)-[r]->() where type(r) =~ /K.*/ return r
```

The relationship that has a type whose name starts with K.

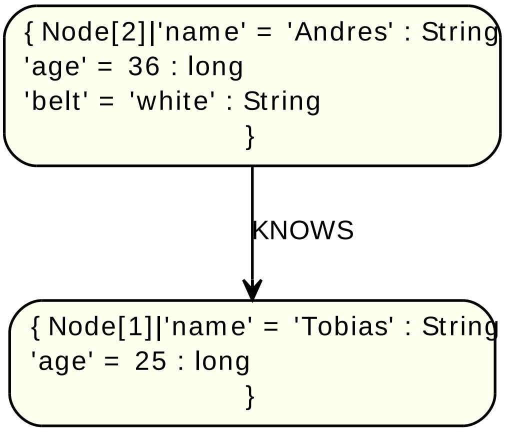
Result

```
+--------------+
| r            |
+--------------+
| :KNOWS[0] {} |
+--------------+
1 rows, 1 ms
```

4.4.5. Property exists

To only include nodes/relationships that have a property, just write out the identifier and the property you expect it to have.

Graph


{ Node[2]|'name' = 'Andres' : String
'age' = 36 : long
'belt' = 'white' : String
}
KNOWS
{ Node[1]|'name' = 'Tobias' : String
'age' = 25 : long
}

Query

```
start n=(2, 1) where n.belt return n
```

The node named Andres.

Result

```
+--------------------------------------------+
| n                                          |
+-----------------------------------------------+
| Node[2]{name->"Andres",age->36,belt->"white"} |
+-----------------------------------------------+
1 rows, 1 ms
```

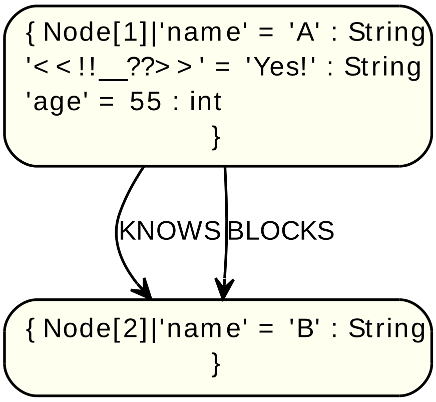
4.5. Return

In the return part of your query, you define which parts of the pattern you are interested in. It can be nodes, relationships, or properties on these.

4.5.1. Return nodes

To return a node, list it in the return statemenet.

Graph


{ Node[1]|'name' = 'A' : String
'<<!!__??>>' = 'Yes!' : String
'age' = 55 : int
}
KNOWS
BLOCKS
{ Node[2]|'name' = 'B' : String
}

Query

```
start n=(2) return n
```

The node.

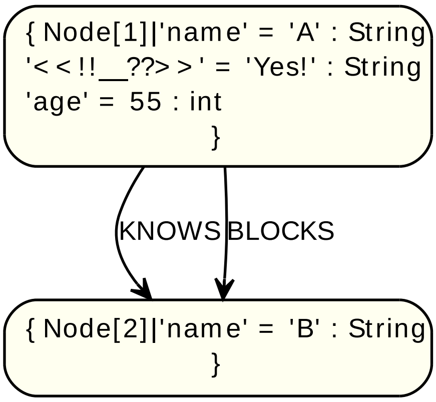
Result

```
+--------------------+
| n                  |
+--------------------+
| Node[2]{name->"B"} |
+--------------------+
1 rows, 0 ms
```

4.5.2. Return relationships

To return a relationship, just include it in the return list.

Graph


{ Node[1]|'name' = 'A' : String
'<<!!__??>>' = 'Yes!' : String
'age' = 55 : int
}
KNOWS
BLOCKS
{ Node[2]|'name' = 'B' : String
}

Query

```
start n=(1) match (n)-[r:KNOWS]->(c) return r
```

The relationship.

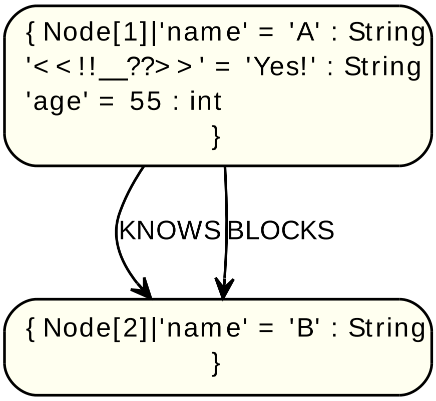
Result

```
+--------------+
| r            |
+--------------+
| :KNOWS[0] {} |
+--------------+
1 rows, 1 ms
```

4.5.3. Return property

To return a property, use the dot separator, like this:

Graph


{ Node[1]|'name' = 'A' : String
'<<!!__??>>' = 'Yes!' : String
'age' = 55 : int
}
KNOWS
BLOCKS
{ Node[2]|'name' = 'B' : String
}

Query

```
start n=(1) return n.name
```

The the value of the property *name*.

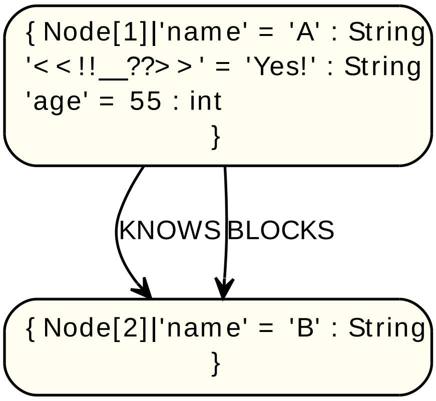
Result

```
+--------+
| n.name |
+--------+
| A      |
+--------+
1 rows, 0 ms
```

4.5.4. Identifier with uncommon characters

To introduce a placeholder that is made up of characters that are outside of the english alphabet, you can use the ` to enclose the identifier, like this:

Graph


{ Node[1]|'name' = 'A' : String
'<<!!__??>>' = 'Yes!' : String
'age' = 55 : int
}
KNOWS
BLOCKS
{ Node[2]|'name' = 'B' : String
}

Query

```
start `This isn't a common identifier`=(1)
return `This isn't a common identifier`.`<<!!__??>>`
```

The node indexed with name "A" is returned

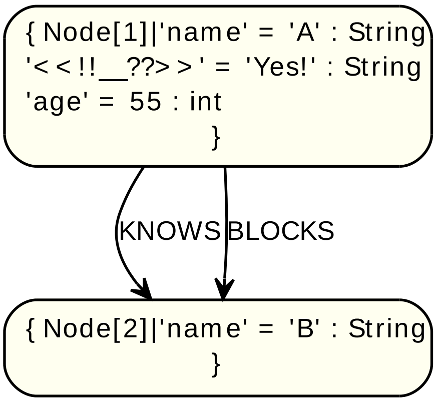
Result

```
+-------------------------------------------+
| This isn't a common identifier.<<!!__??>> |
+-------------------------------------------+
| Yes!                                      |
+-------------------------------------------+
1 rows, 1 ms
```

4.5.5. Optional properties

If a property might or might not be there, you can select it optionally by adding a questionmark to the identifier, like this:

Graph


{ Node[1]|'name' = 'A' : String
'<<!!__??>>' = 'Yes!' : String
'age' = 55 : int
}
KNOWS
BLOCKS
{ Node[2]|'name' = 'B' : String
}

Query

```
start n=(1, 2) return n.age?
```

The age when the node has that property, or null if the property is not there.

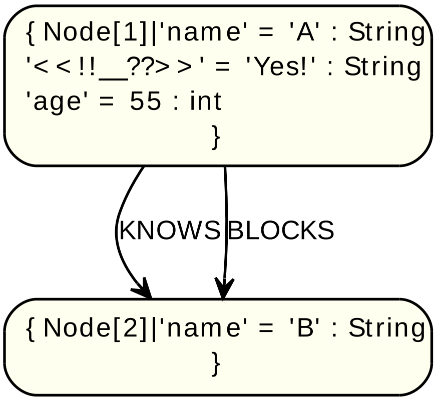
Result

```
+--------+
| n.age  |
+--------+
| 55     |
| <null> |
+--------+
2 rows, 0 ms
```

4.5.6. Unique results

DISTINCT retrieves only unique rows depending on the columns that have been selected to output.

Graph


{ Node[1]|'name' = 'A' : String
'<<!!__??>>' = 'Yes!' : String
'age' = 55 : int
}
KNOWS
BLOCKS
{ Node[2]|'name' = 'B' : String
}

Query

```
start a=(1) match (a)-->(b) return distinct b
```

The node named B, but only once.

Result

```
+--------------------+
| b                  |
+--------------------+
| Node[2]{name->"B"} |
+--------------------+
1 rows, 1 ms
```

4.6. Aggregation

To calculate aggregated data, Cypher offers aggregation, much like SQL's `GROUP BY`. If any aggregation functions are found in the `RETURN` statement, all the columns without aggregating functions are used as the grouping key.

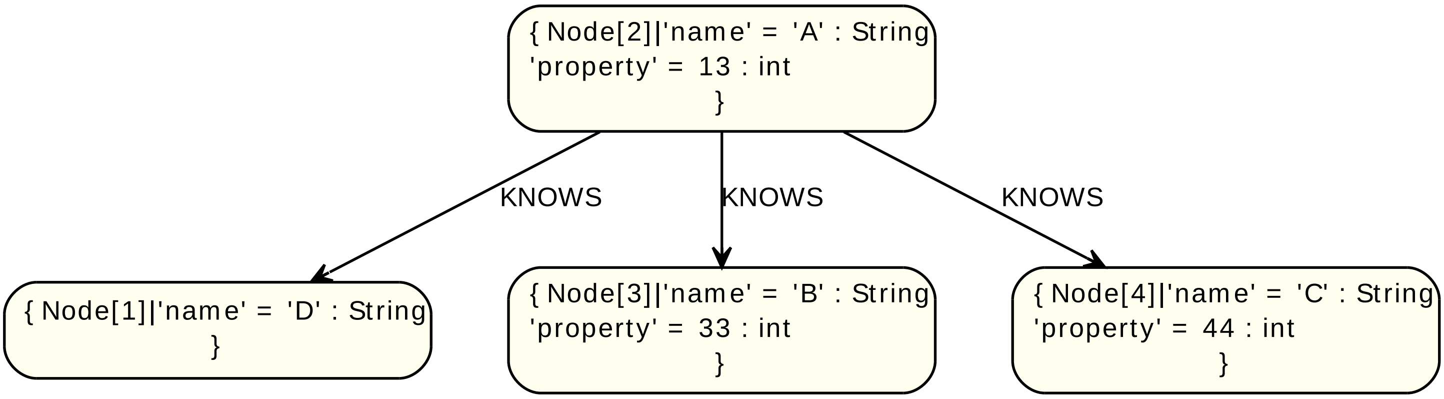
4.6.1. COUNT

`COUNT` is used to count the number of rows. `COUNT` can be used in two forms - `COUNT(*)` which just counts the number of matching rows, and `COUNT(<identifier>)`, which counts the number of non-null values in `<identifier>`.

4.6.2. Count nodes

To count the number of nodes, for example the number of nodes connected to one node, you can use `count(*)`.

Graph


{ Node[2]|'name' = 'A' : String
'property' = 13 : int
}
KNOWS
KNOWS
KNOWS
{ Node[1]|'name' = 'D' : String
}
{ Node[3]|'name' = 'B' : String
'property' = 33 : int
}
{ Node[4]|'name' = 'C' : String
'property' = 44 : int
}

Query

```
start n=(2) match (n)-->(x) return n, count(*)
```

The start node and the count of related nodes.

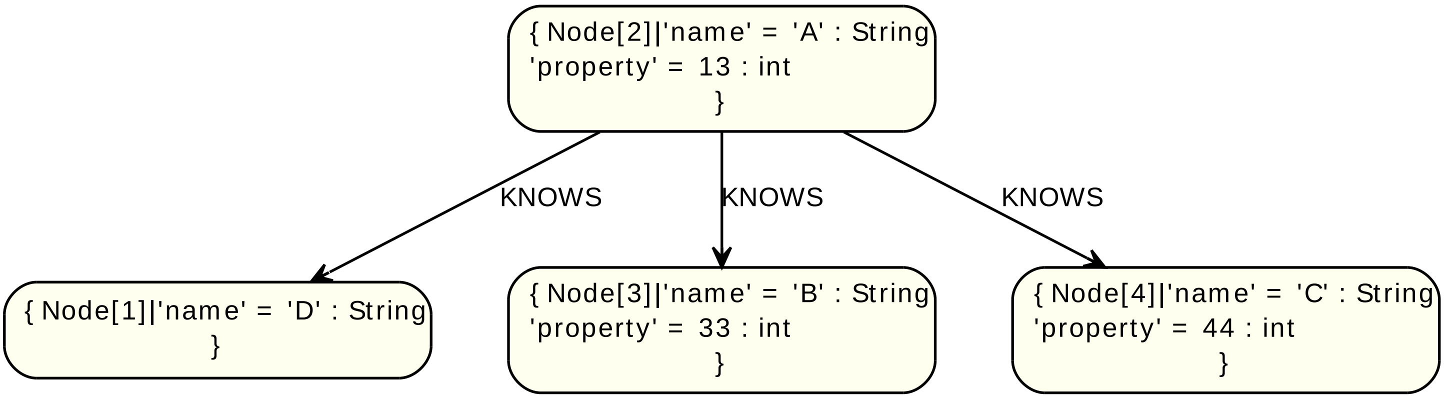
Result

```
+-------------------------------------------+
| n                             | count(*) |
+-------------------------------------------+
| Node[2]{name->"A",property->13} | 3        |
+-------------------------------------------+
1 rows, 2 ms
```

4.6.3. Count entities

Instead of counting the number of results with count(*), it might be more expressive to include the name of the identifier you care about.

Graph


{ Node[2]|'name' = 'A' : String
'property' = 13 : int
}
KNOWS
KNOWS
KNOWS
{ Node[1]|'name' = 'D' : String
}
{ Node[3]|'name' = 'B' : String
'property' = 33 : int
}
{ Node[4]|'name' = 'C' : String
'property' = 44 : int
}

Query

```
start n=(2) match (n)-->(x) return count(x)
```

The number of connected nodes from the start node.

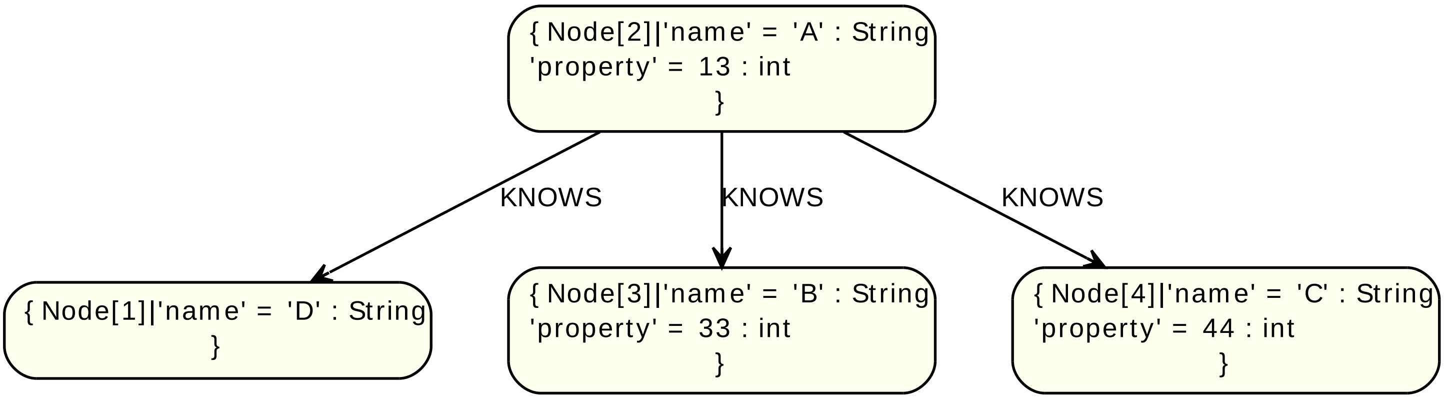
Result

```
+----------+
| count(x) |
+----------+
| 3        |
+----------+
1 rows, 2 ms
```

4.6.4. Count non null values

You can count the non-null values by using `count(<identifier>)`.

Graph


{ Node[2]|'name' = 'A' : String
'property' = 13 : int
}
KNOWS
KNOWS
KNOWS
{ Node[1]|'name' = 'D' : String
}
{ Node[3]|'name' = 'B' : String
'property' = 33 : int
}
{ Node[4]|'name' = 'C' : String
'property' = 44 : int
}

Query

```
start n=(2,3,4,1) return count(n.property?)
```

The count of related nodes.

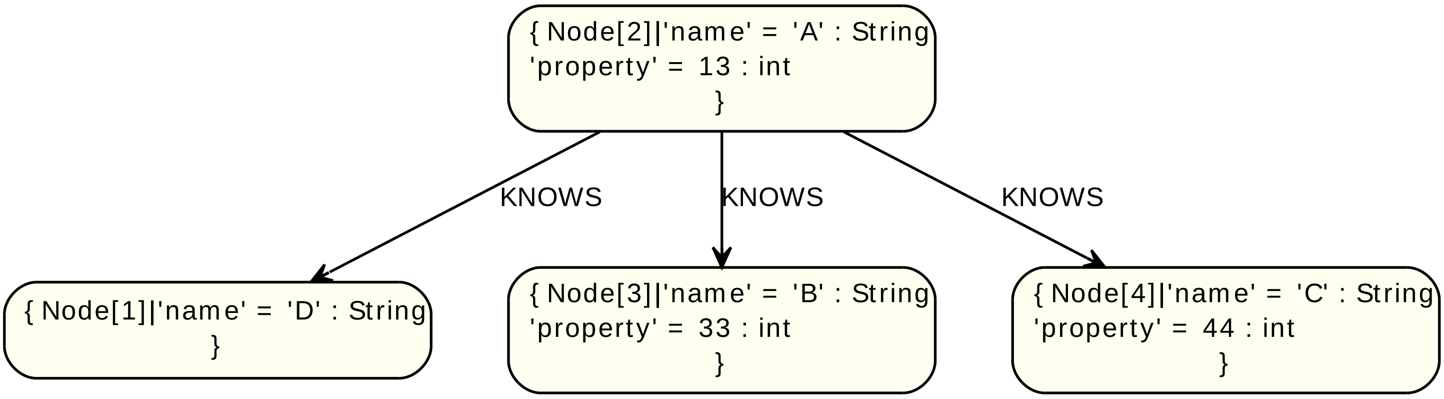
Result

```
+-------------------+
| count(n.property) |
+-------------------+
| 3                 |
+-------------------+
1 rows, 1 ms
```

4.6.5. SUM

The `SUM` aggregation function simply sums all the numeric values it encounters. Null values are silently dropped. This is an example of how you can use `SUM`.

Graph


{ Node[2]|'name' = 'A' : String
'property' = 13 : int
}
KNOWS
KNOWS
KNOWS
{ Node[1]|'name' = 'D' : String
}
{ Node[3]|'name' = 'B' : String
'property' = 33 : int
}
{ Node[4]|'name' = 'C' : String
'property' = 44 : int
}

Query

```
start n=(2,3,4) return sum(n.property)
```

The sum of all the values in the property *property*.

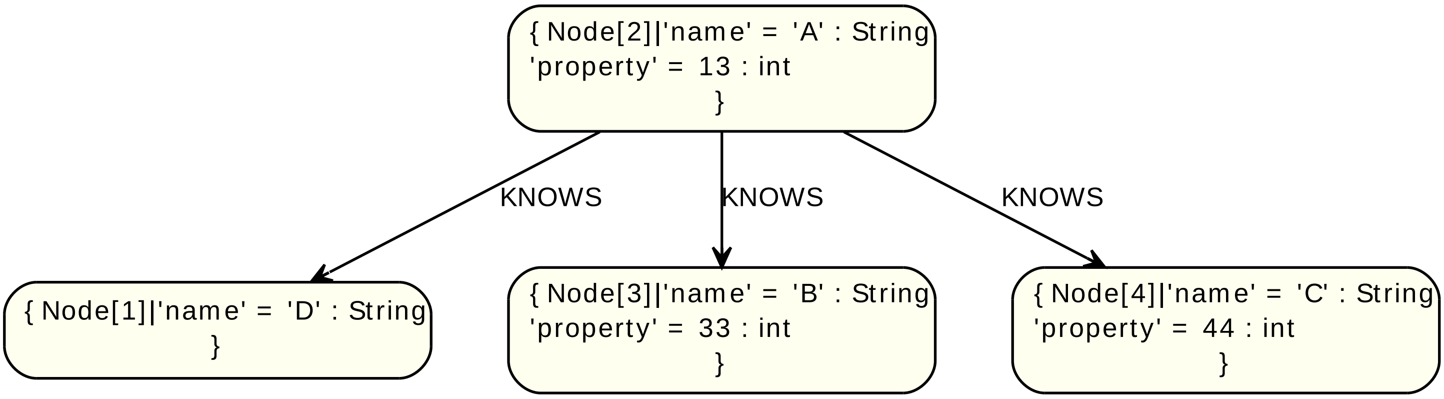
Result

```
+-----------------+
| sum(n.property) |
+-----------------+
| 90              |
+-----------------+
1 rows, 0 ms
```

4.6.6. AVG

`AVG` calculates the average of a numeric column.

Graph


{ Node[2]|'name' = 'A' : String
'property' = 13 : int
}
KNOWS
KNOWS
KNOWS
{ Node[1]|'name' = 'D' : String
}
{ Node[3]|'name' = 'B' : String
'property' = 33 : int
}
{ Node[4]|'name' = 'C' : String
'property' = 44 : int
}

Query

```
start n=(2,3,4) return avg(n.property)
```

The average of all the values in the property *property*.

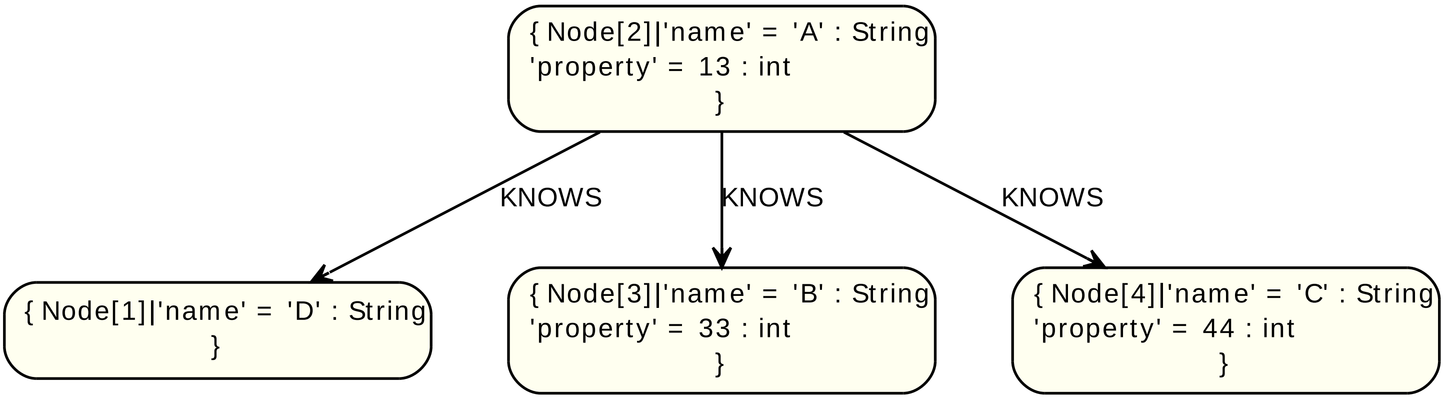
Result

```
+-----------------+
| avg(n.property) |
+-----------------+
| 30.0            |
+-----------------+
1 rows, 1 ms
```

4.6.7. MAX

`MAX` find the largets value in a numeric column.

Graph


{ Node[2]|'name' = 'A' : String
'property' = 13 : int
}
KNOWS
KNOWS
KNOWS
{ Node[1]|'name' = 'D' : String
}
{ Node[3]|'name' = 'B' : String
'property' = 33 : int
}
{ Node[4]|'name' = 'C' : String
'property' = 44 : int
}

Query

```
start n=(2,3,4) return max(n.property)
```

The largest of all the values in the property *property*.

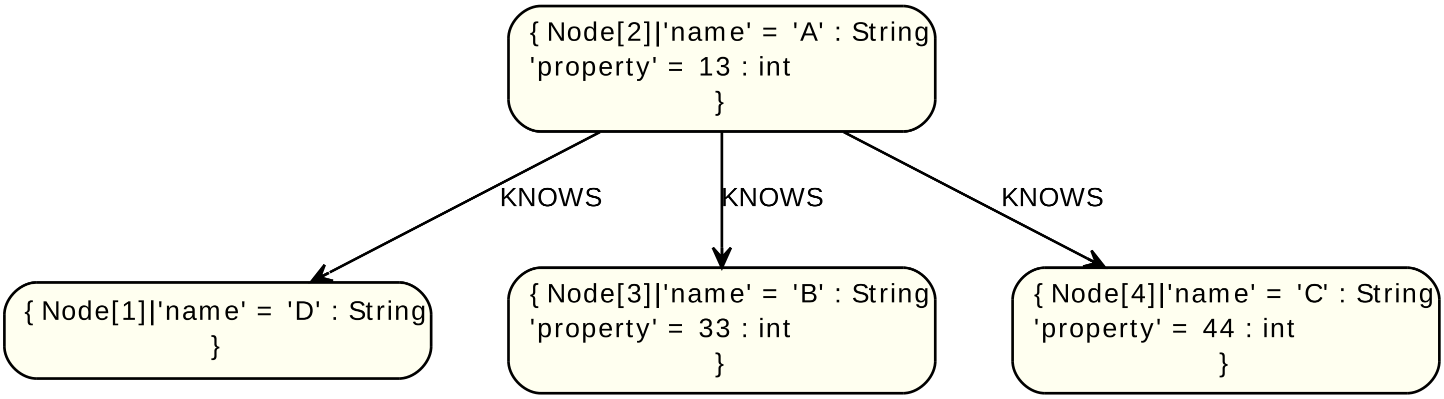
Result

```
+-----------------+
| max(n.property) |
+-----------------+
| 44              |
+-----------------+
1 rows, 1 ms
```

4.6.8. MIN

`MIN` takes a numeric property as input, and returns the smallest value in that column.

Graph


{ Node[2]|'name' = 'A' : String
'property' = 13 : int
}
KNOWS
KNOWS
KNOWS
{ Node[1]|'name' = 'D' : String
}
{ Node[3]|'name' = 'B' : String
'property' = 33 : int
}
{ Node[4]|'name' = 'C' : String
'property' = 44 : int
}

Query

```
start n=(2,3,4) return min(n.property)
```

The smallest of all the values in the property *property*.

Result

```
+-----------------+
| min(n.property) |
+-----------------+
| 13              |
+-----------------+
1 rows, 0 ms
```

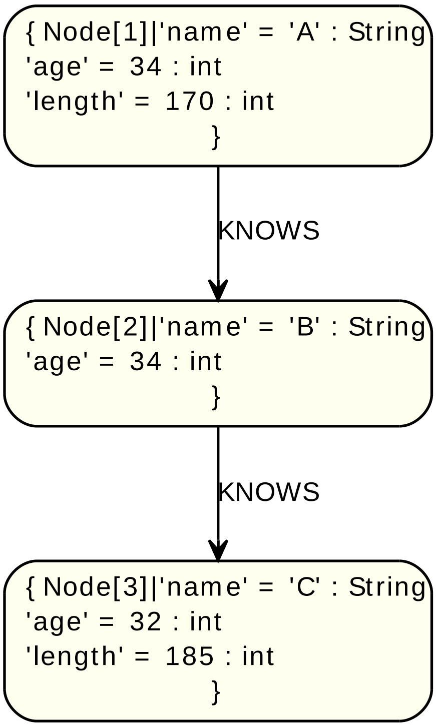
4.7. Order by

To sort the output, use the Order by clause. Note that you can not sort on nodes or relationships, just on properties on these.

4.7.1. Order nodes by property

ORDER BY is used to sort the output

Graph



Query

```
start n=(3,1,2) return n order by n.name
```

The nodes, sorted by their name.

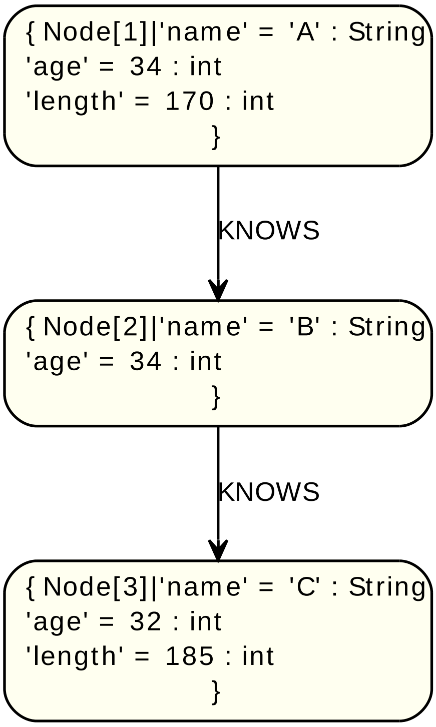
Result

```
+---------------------------------------+
| n                                     |
+---------------------------------------+
| Node[1]{name->"A",age->34,length->170} |
| Node[2]{name->"B",age->34}            |
| Node[3]{name->"C",age->32,length->185} |
+---------------------------------------+
3 rows, 1 ms
```

4.7.2. Order nodes by multiple properties

You can order by multiple properties by stating each identifier in the ORDER BY statement. Cypher will sort the result by the first identifier listed, and for equals values, go to the next property in the order by, and so on.

Graph



Query

```
start n=(3,1,2) return n order by n.age, n.name
```

The nodes, sorted first by their age, and then by their name.

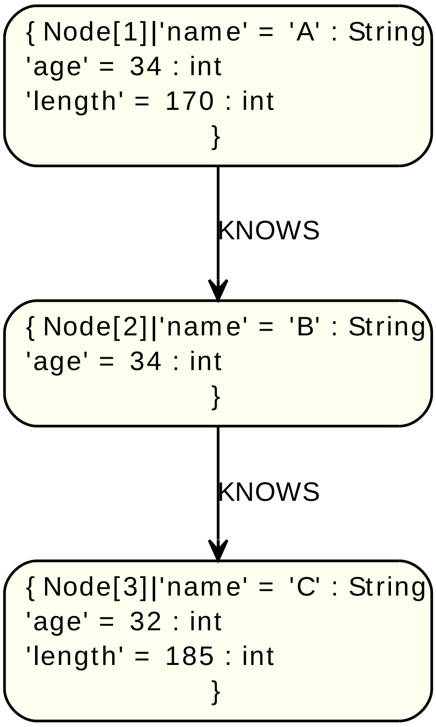
Result

```
+-----------------------------------------+
| n                                       |
+-----------------------------------------+
| Node[3]{name->"C",age->32,length->185} |
| Node[1]{name->"A",age->34,length->170} |
| Node[2]{name->"B",age->34}             |
+-----------------------------------------+
3 rows, 1 ms
```

4.7.3. Order nodes in descending order

By adding DESC[ENDING] after the identifier to sort on, the sort will be done in reverse order.

Graph



Query

```
start n=(3,1,2) return n order by n.name DESC
```

The nodes, sorted by their name reversely.

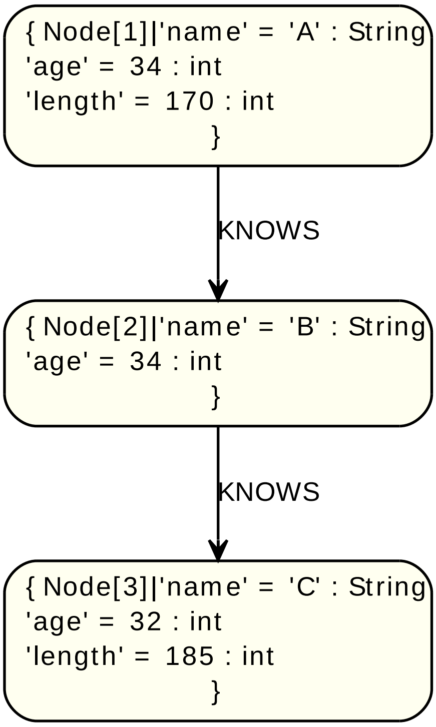
Result

```
+---------------------------------------+
| n                                     |
+---------------------------------------+
| Node[3]{name->"C",age->32,length->185} |
| Node[2]{name->"B",age->34}            |
| Node[1]{name->"A",age->34,length->170} |
+---------------------------------------+
3 rows, 0 ms
```

4.7.4. Ordering null

When sorting the result set, null will always come at the end of the result set for ascending sorting, and first when doing descending sort.

Graph



Query

```
start n=(3,1,2) return n.length?, n order by n.length?
```

The nodes sorted by the length property, with a node without that property last.

Result

```
+---------------------------------------------------+
| n.length | n                                      |
+---------------------------------------------------+
| 170      | Node[1]{name->"A",age->34,length->170} |
| 185      | Node[3]{name->"C",age->32,length->185} |
| <null>   | Node[2]{name->"B",age->34}             |
+---------------------------------------------------+
3 rows, 1 ms
```

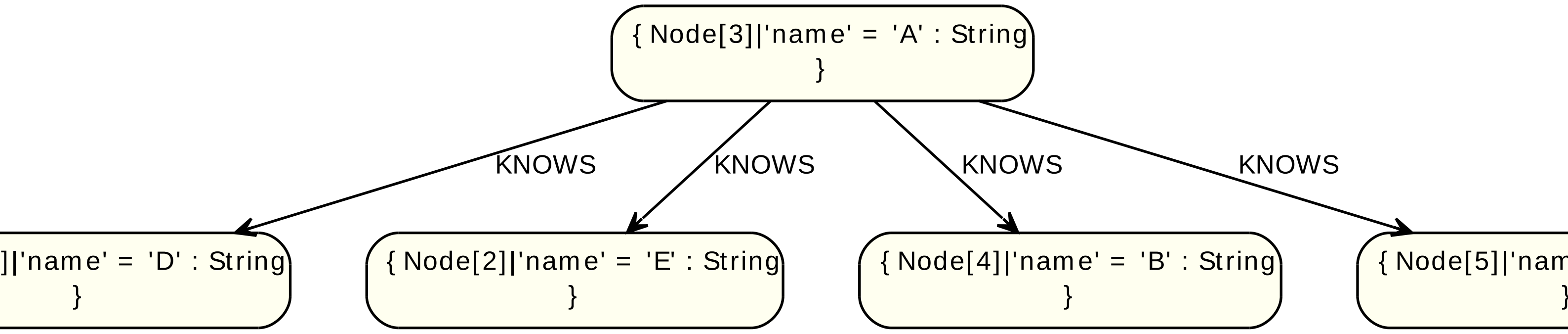
4.8. Skip

Skip enables the return of only subsets of the total result. By using skip, the result set will trimmed from the top. Please note that no guarantees are made on the order of the result unless the query species the order by clause.

4.8.1. Skip first three

To return a subset of the result, starting from third result, use this syntax:

Graph


{ Node[3]|'name' = 'A' : String }
KNOWS
KNOWS
KNOWS
KNOWS
]|'name' = 'D' : String }
{ Node[2]|'name' = 'E' : String }
{ Node[4]|'name' = 'B' : String }
{ Node[5]|'nam

Query

```
start n=(3, 4, 5, 1, 2) return n order by n.name skip 3
```

The first three nodes are skipped, and only the last two are returned.

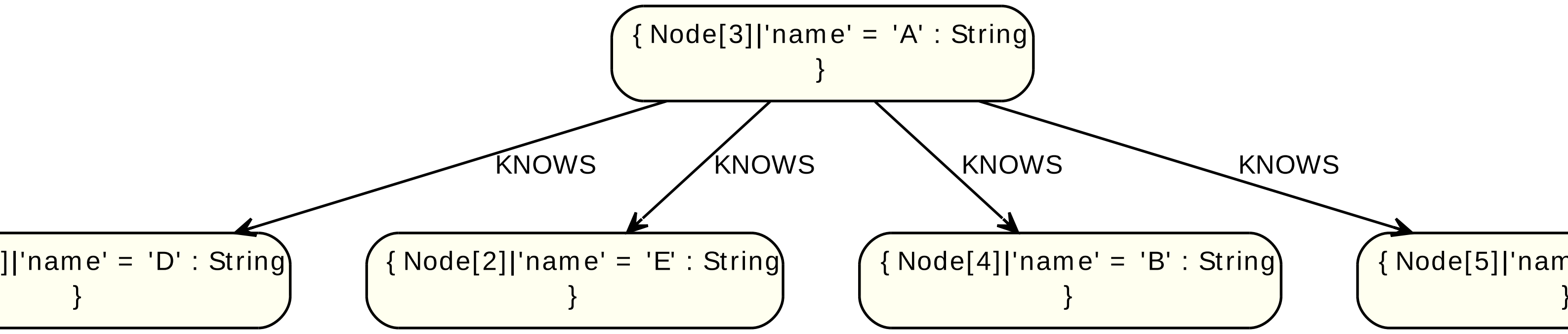
Result

```
+--------------------+
| n                  |
+--------------------+
| Node[1]{name->"D"} |
| Node[2]{name->"E"} |
+--------------------+
2 rows, 1 ms
```

4.8.2. Return middle two

To return a subset of the result, starting from somewhere in the middle, use this syntax:

Graph


{ Node[3]|'name' = 'A' : String }
KNOWS
KNOWS
KNOWS
KNOWS
]|'name' = 'D' : String }
{ Node[2]|'name' = 'E' : String }
{ Node[4]|'name' = 'B' : String }
{ Node[5]|'nam

Query

```
start n=(3, 4, 5, 1, 2) return n order by n.name skip 1 limit 2
```

Two nodes from the middle are returned

Result

```
+--------------------+
| n                  |
+--------------------+
| Node[4]{name->"B"} |
| Node[5]{name->"C"} |
+--------------------+
2 rows, 1 ms
```

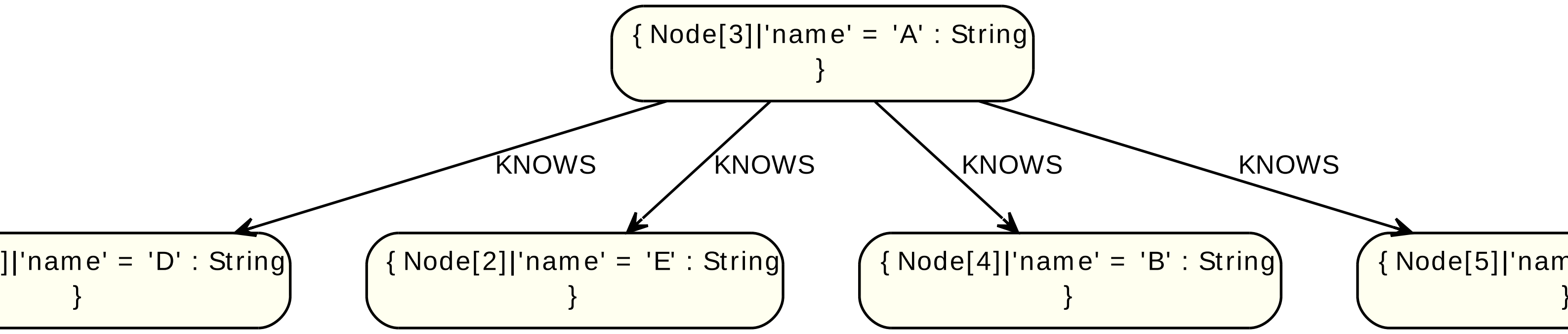
4.9. Limit

Limit enables the return of only subsets of the total result.

4.9.1. Return first part

To return a subset of the result, starting from the top, use this syntax:

Graph


{ Node[3]|'name' = 'A' : String }
KNOWS
KNOWS
KNOWS
KNOWS
]|'name' = 'D' : String }
{ Node[2]|'name' = 'E' : String }
{ Node[4]|'name' = 'B' : String }
{ Node[5]|'nam

Query

```
start n=(3, 4, 5, 1, 2) return n limit 3
```

The top three items are returned

Result

```
+--------------------+
| n                  |
+--------------------+
| Node[3]{name->"A"} |
| Node[4]{name->"B"} |
| Node[5]{name->"C"} |
+--------------------+
3 rows, 1 ms
```

4.10. Functions in Cypher

Here is an list of the functions in Cypher, seperated into three different sections: Predicates and Aggregated functions

4.10.1. Predicates

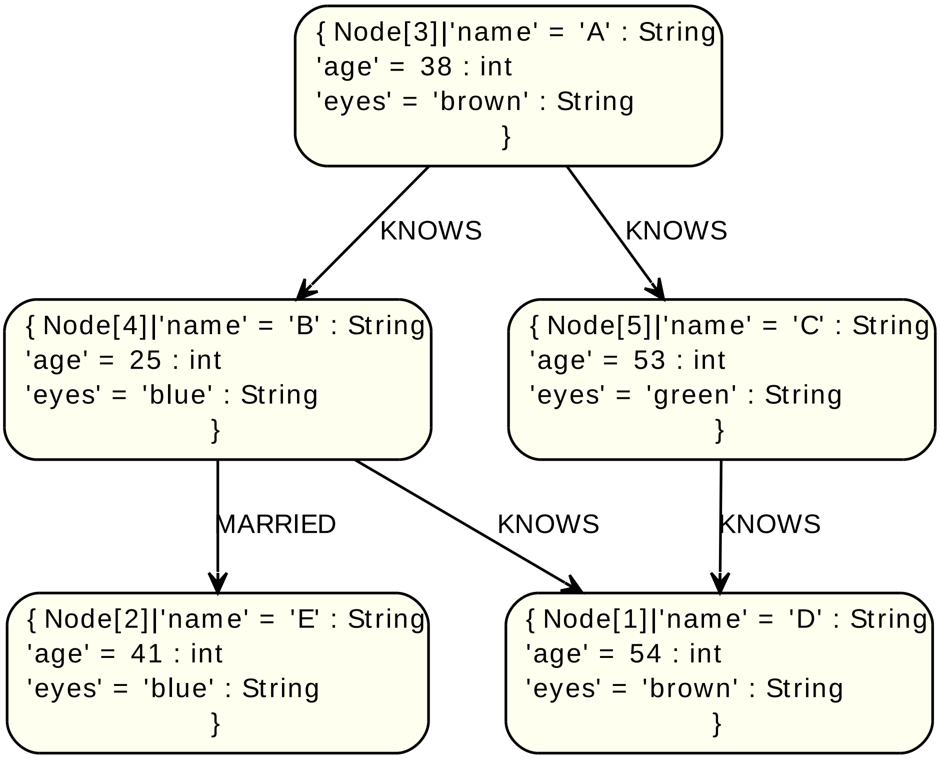
Predicates are boolean functions that return true or false for a given set of input. They are most commonly used to filter out subgraphs in the WHERE part of a query.

4.10.2. ALL

Tests the predicate closure to see if all items in the iterable match.

Syntax: ALL(iterable, [symbol #] predicate-closure) Arguments: iterable: An array property, or an iterable symbol, or an iterable function. symbol: The closure will have a symbol introduced in it’s context. Here you decide which symbol to use. If you leave the symbol out, the default symbol _ (underscore) will be used. predicate-closure: A predicate that is tested against all items in iterable

Graph


{ Node[3]|'name' = 'A' : String
'age' = 38 : int
'eyes' = 'brown' : String
}
KNOWS
KNOWS
{ Node[4]|'name' = 'B' : String
'age' = 25 : int
'eyes' = 'blue' : String
}
{ Node[5]|'name' = 'C' : String
'age' = 53 : int
'eyes' = 'green' : String
}
MARRIED
KNOWS
KNOWS
{ Node[2]|'name' = 'E' : String
'age' = 41 : int
'eyes' = 'blue' : String
}
{ Node[1]|'name' = 'D' : String
'age' = 54 : int
'eyes' = 'brown' : String
}

Query

```
start a=(3), b=(1) match p=a-[^1..3]->b where all(nodes(p), x => x.age > 30) return p
```

All nodes in the path.

Result

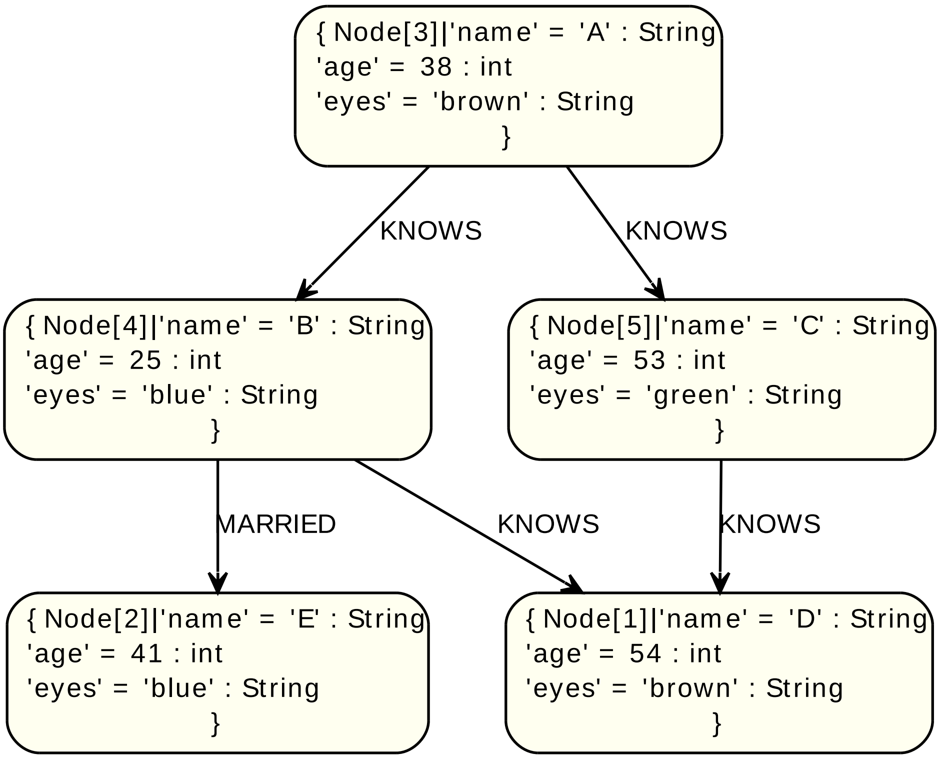
```
+-------------------------------------------------------------------+
| p                                                                 |
+-------------------------------------------------------------------+
| Path(Node[3], Relationship[1], Node[5], Relationship[3], Node[1]) |
+-------------------------------------------------------------------+
1 rows, 3 ms
```

4.10.3. ANY

Tests the predicate closure to see if at least one item in the iterable match.

Syntax: ANY(iterable, [symbol #] predicate-closure) Arguments: iterable: An array property, or an iterable symbol, or an iterable function. symbol: The closure will have a symbol introduced in it’s context. Here you decide which symbol to use. If you leave the symbol out, the default symbol _ (underscore) will be used. predicate-closure: A predicate that is tested against all items in iterable

Graph


{ Node[3]|'name' = 'A' : String
'age' = 38 : int
'eyes' = 'brown' : String
}
KNOWS
KNOWS
{ Node[4]|'name' = 'B' : String
'age' = 25 : int
'eyes' = 'blue' : String
}
{ Node[5]|'name' = 'C' : String
'age' = 53 : int
'eyes' = 'green' : String
}
MARRIED
KNOWS
KNOWS
{ Node[2]|'name' = 'E' : String
'age' = 41 : int
'eyes' = 'blue' : String
}
{ Node[1]|'name' = 'D' : String
'age' = 54 : int
'eyes' = 'brown' : String
}

Query

```
start a=(3) match p=a-[^1..3]->b where any(nodes(p), x => x.eyes = "blue") return p
```

All nodes in the path.

Result

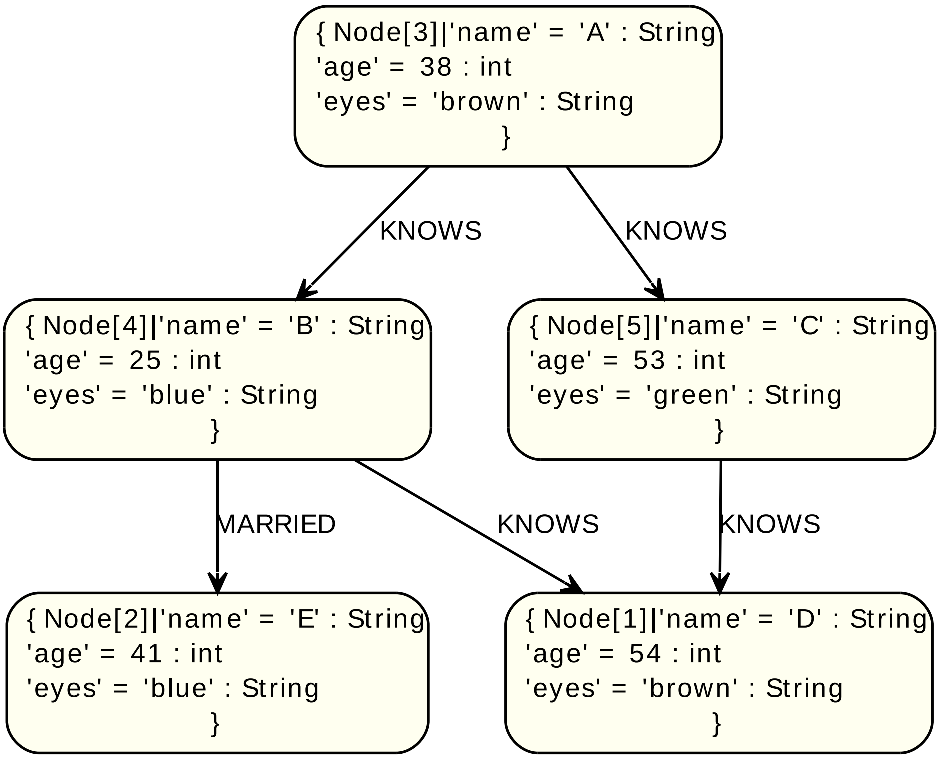
```
+-------------------------------------------------------------------+
| p                                                                 |
+-------------------------------------------------------------------+
| Path(Node[3], Relationship[0], Node[4])                           |
| Path(Node[3], Relationship[0], Node[4], Relationship[2], Node[1]) |
| Path(Node[3], Relationship[0], Node[4], Relationship[4], Node[2]) |
+-------------------------------------------------------------------+
3 rows, 5 ms
```

4.10.4. NONE

Tests the predicate closure to see if no items in the iterable match. If even one matches, the function returns false.

Syntax: NONE(iterable, [symbol #] predicate-closure) Arguments: iterable: An array property, or an iterable symbol, or an iterable function. symbol: The closure will have a symbol introduced in it’s context. Here you decide which symbol to use. If you leave the symbol out, the default symbol _ (underscore) will be used. predicate-closure: A predicate that is tested against all items in iterable

Graph


{ Node[3]|'name' = 'A' : String
'age' = 38 : int
'eyes' = 'brown' : String
}
KNOWS
KNOWS
{ Node[4]|'name' = 'B' : String
'age' = 25 : int
'eyes' = 'blue' : String
}
{ Node[5]|'name' = 'C' : String
'age' = 53 : int
'eyes' = 'green' : String
}
MARRIED
KNOWS
KNOWS
{ Node[2]|'name' = 'E' : String
'age' = 41 : int
'eyes' = 'blue' : String
}
{ Node[1]|'name' = 'D' : String
'age' = 54 : int
'eyes' = 'brown' : String
}

Query

```
start n=(3) match p=n-[^1..3]->b where NONE(nodes(p), x => x.age = 25) return p
```

All nodes in the path.

Result

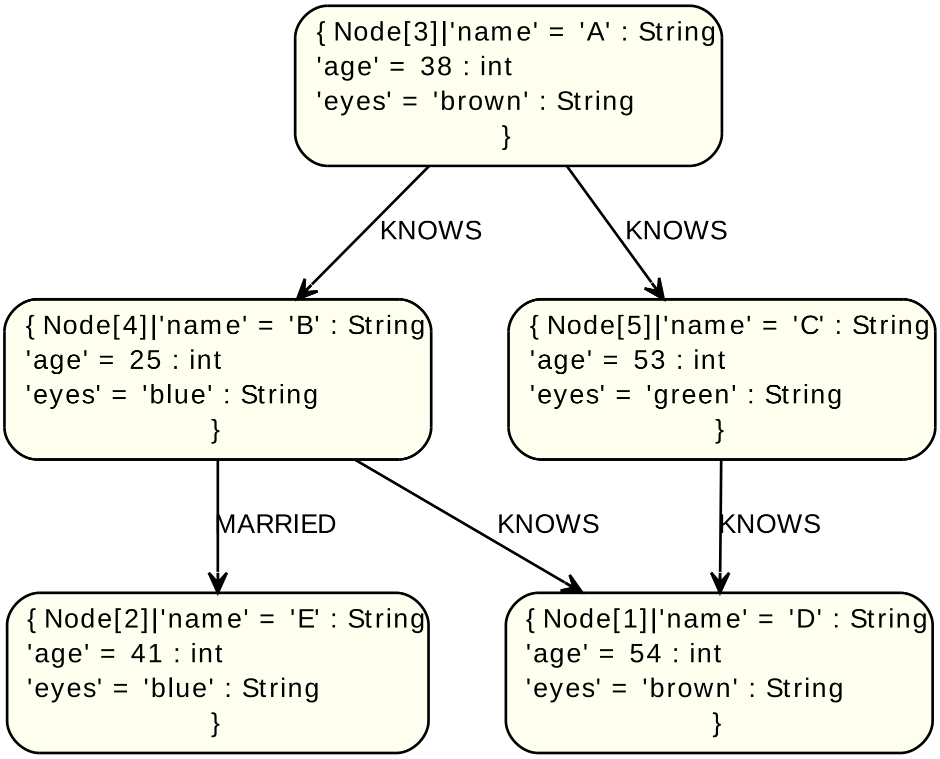
```
+------------------------------------------------------------------+
| p                                                                |
+------------------------------------------------------------------+
| Path(Node[3], Relationship[1], Node[5])                          |
| Path(Node[3], Relationship[1], Node[5], Relationship[3], Node[1]) |
+------------------------------------------------------------------+
2 rows, 5 ms
```

4.10.5. SINGLE

Returns true if the closure predicate matches exactly one of the items in the iterable.

Syntax: SINGLE(iterable, [symbol #] predicate-closure) Arguments: iterable: An array property, or an iterable symbol, or an iterable function. symbol: The closure will have a symbol introduced in it’s context. Here you decide which symbol to use. If you leave the symbol out, the default symbol _ (underscore) will be used. predicate-closure: A predicate that is tested against all items in iterable

Graph


{ Node[3]|'name' = 'A' : String
'age' = 38 : int
'eyes' = 'brown' : String
}
KNOWS
KNOWS
{ Node[4]|'name' = 'B' : String
'age' = 25 : int
'eyes' = 'blue' : String
}
{ Node[5]|'name' = 'C' : String
'age' = 53 : int
'eyes' = 'green' : String
}
MARRIED
KNOWS
KNOWS
{ Node[2]|'name' = 'E' : String
'age' = 41 : int
'eyes' = 'blue' : String
}
{ Node[1]|'name' = 'D' : String
'age' = 54 : int
'eyes' = 'brown' : String
}

Query

```
start n=(3) match p=n-->b where SINGLE(nodes(p), _.eyes = "blue") return p
```

All nodes in the path.

Result

```
+----------------------------------------+
| p                                      |
+----------------------------------------+
| Path(Node[3], Relationship[0], Node[4]) |
+----------------------------------------+
1 rows, 2 ms
```

4.10.6. Scalar functions

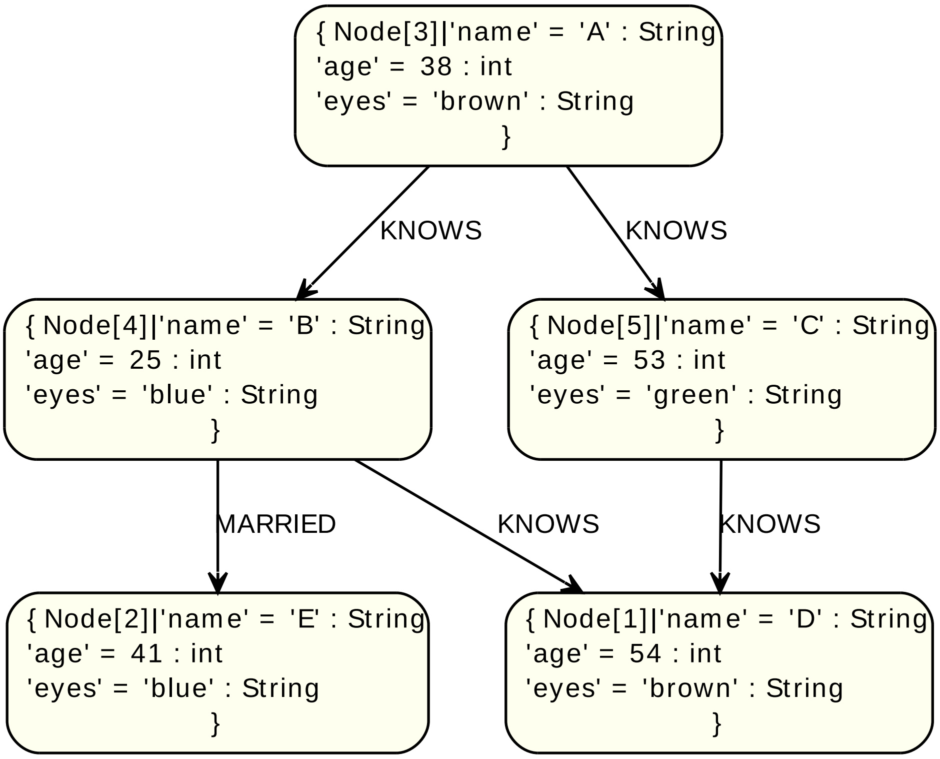
Scalar functions return a single value.

4.10.7. LENGTH

To return or filter on the length of a path, use the special property LENGTH

Syntax: LENGTH(iterable) Arguments: iterable: An iterable, value or function call

Graph


{ Node[3]|'name' = 'A' : String
'age' = 38 : int
'eyes' = 'brown' : String
}
KNOWS
KNOWS
{ Node[4]|'name' = 'B' : String
'age' = 25 : int
'eyes' = 'blue' : String
}
{ Node[5]|'name' = 'C' : String
'age' = 53 : int
'eyes' = 'green' : String
}
MARRIED
KNOWS
KNOWS
{ Node[2]|'name' = 'E' : String
'age' = 41 : int
'eyes' = 'blue' : String
}
{ Node[1]|'name' = 'D' : String
'age' = 54 : int
'eyes' = 'brown' : String
}

Query

```
start a=(3) match p=a-->b-->c return length(p)
```

The length of the path p.

Result

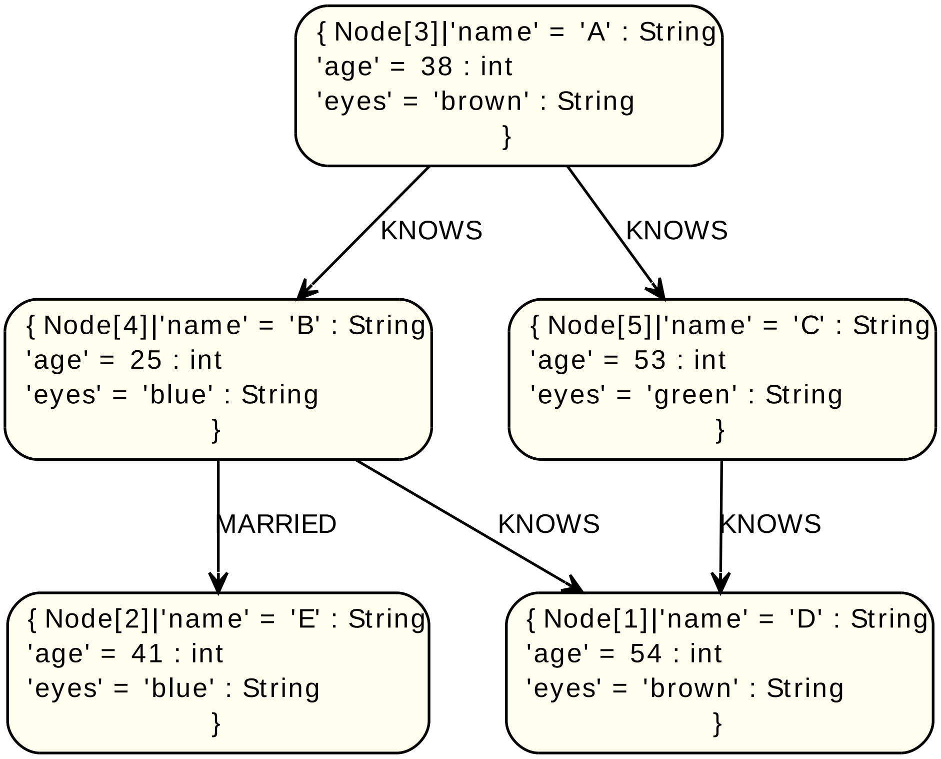
```
+-----------+
| LENGTH(p) |
+-----------+
| 5         |
| 5         |
| 5         |
+-----------+
3 rows, 4 ms
```

4.10.8. TYPE

Returns a string representation of the relationship type.

Syntax: TYPE(relationship) Arguments: relationship: A relationship

Graph


{ Node[3]|'name' = 'A' : String
'age' = 38 : int
'eyes' = 'brown' : String
}
KNOWS
KNOWS
{ Node[4]|'name' = 'B' : String
'age' = 25 : int
'eyes' = 'blue' : String
}
{ Node[5]|'name' = 'C' : String
'age' = 53 : int
'eyes' = 'green' : String
}
MARRIED
KNOWS
KNOWS
{ Node[2]|'name' = 'E' : String
'age' = 41 : int
'eyes' = 'blue' : String
}
{ Node[1]|'name' = 'D' : String
'age' = 54 : int
'eyes' = 'brown' : String
}

Query

```
start n=(3) match (n)-[r]->() return type(r)
```

The relationship type of r.

Result

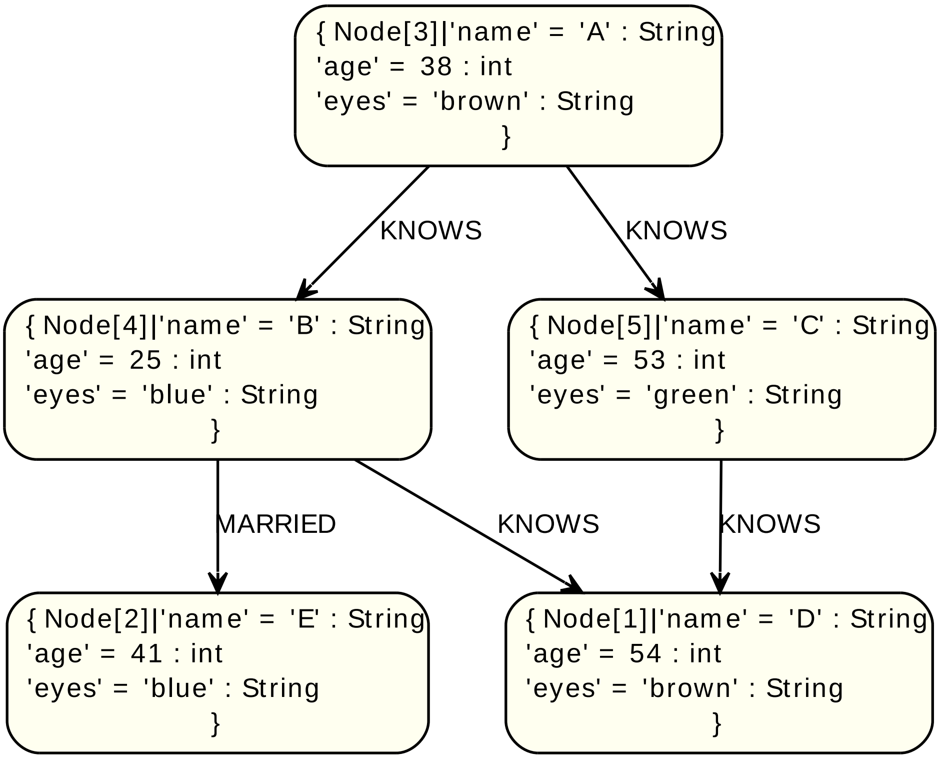
```
+---------+
| TYPE(r) |
+---------+
| KNOWS   |
| KNOWS   |
+---------+
2 rows, 1 ms
```

4.10.9. ID

Returns the id of the relationship or node

Syntax: ID(property-container) Arguments: property-container: A node or a relationship

Graph


{ Node[3]|'name' = 'A' : String
'age' = 38 : int
'eyes' = 'brown' : String
}
KNOWS
KNOWS
{ Node[4]|'name' = 'B' : String
'age' = 25 : int
'eyes' = 'blue' : String
}
{ Node[5]|'name' = 'C' : String
'age' = 53 : int
'eyes' = 'green' : String
}
MARRIED
KNOWS
KNOWS
{ Node[2]|'name' = 'E' : String
'age' = 41 : int
'eyes' = 'blue' : String
}
{ Node[1]|'name' = 'D' : String
'age' = 54 : int
'eyes' = 'brown' : String
}

Query

```
start a=(3, 4, 5) return ID(a)
```

The node id for three nodes.

Result

```
+-------+
| ID(a) |
+-------+
| 3     |
| 4     |
| 5     |
+-------+
3 rows, 0 ms
```

4.10.10. Iterable functions

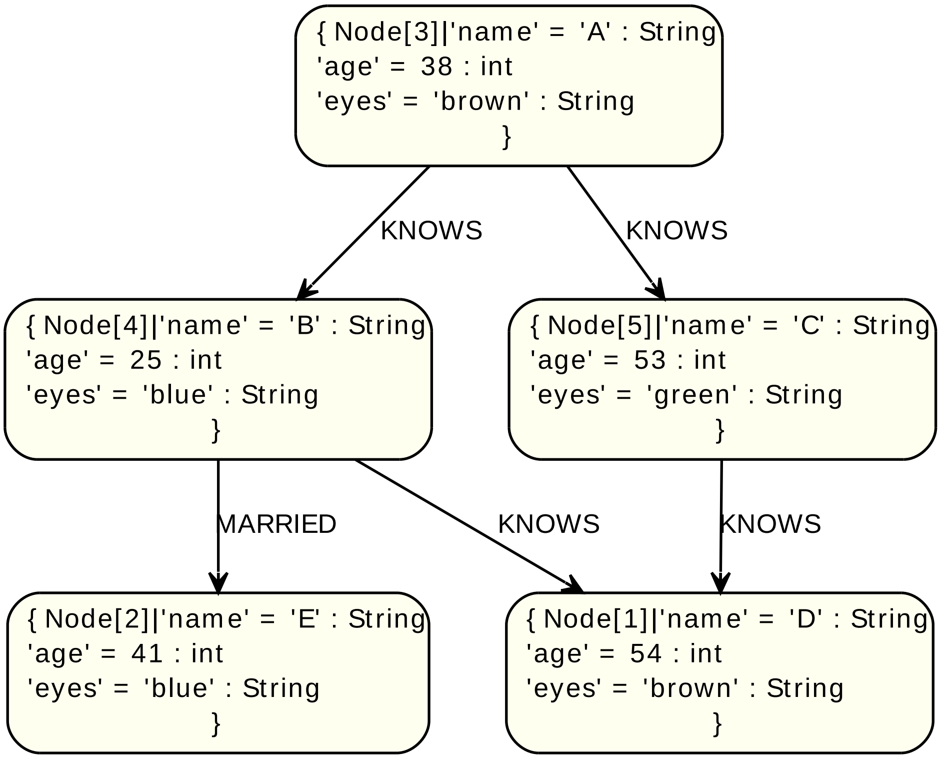
Iterable functions return an iterable of things - nodes in a path, and so on.

4.10.11. NODES

Returns all nodes in a path

Syntax: NODES(path) Arguments: path: A path

Graph


{ Node[3]|'name' = 'A' : String
'age' = 38 : int
'eyes' = 'brown' : String
}
KNOWS
KNOWS
{ Node[4]|'name' = 'B' : String
'age' = 25 : int
'eyes' = 'blue' : String
}
{ Node[5]|'name' = 'C' : String
'age' = 53 : int
'eyes' = 'green' : String
}
MARRIED
KNOWS
KNOWS
{ Node[2]|'name' = 'E' : String
'age' = 41 : int
'eyes' = 'blue' : String
}
{ Node[1]|'name' = 'D' : String
'age' = 54 : int
'eyes' = 'brown' : String
}

Query

```
start a=(3), c=(2) match p=a-->b-->c return NODES(p)
```

All the nodes in the path p.

Result

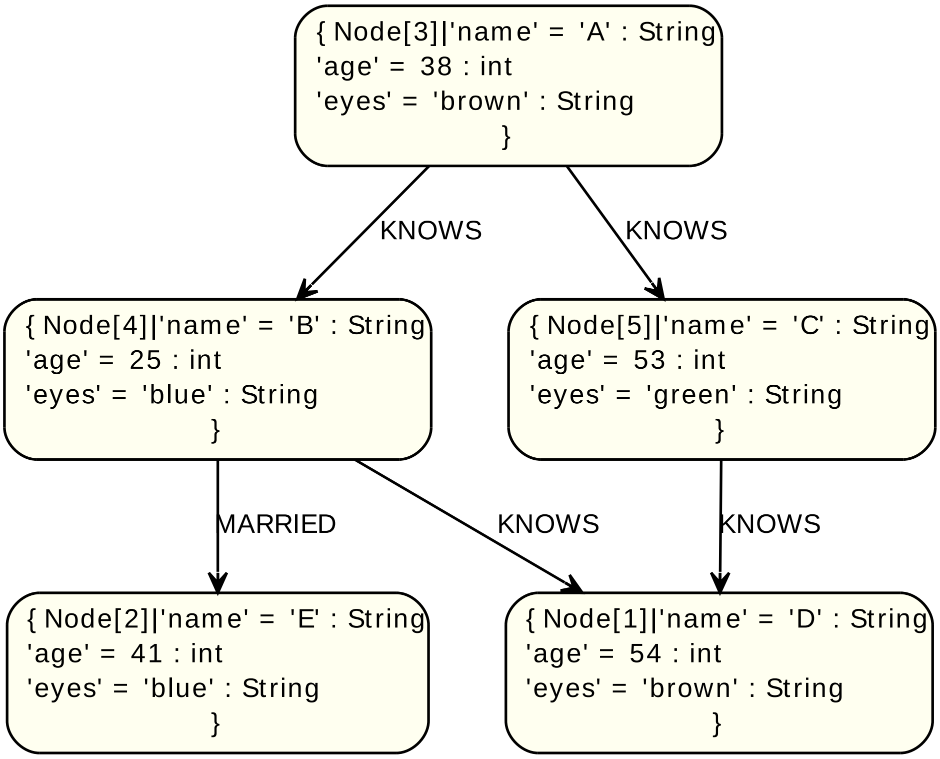
```
+--------------------------------+
| NODES(p)                       |
+--------------------------------+
| List(Node[3], Node[4], Node[2]) |
+--------------------------------+
1 rows, 2 ms
```

4.10.12. RELATIONSHIPS

Returns all relationships in a path

Syntax: RELATIONSHIPS(path) Arguments: path: A path

Graph


{ Node[3]|'name' = 'A' : String
'age' = 38 : int
'eyes' = 'brown' : String
}
KNOWS
KNOWS
{ Node[4]|'name' = 'B' : String
'age' = 25 : int
'eyes' = 'blue' : String
}
{ Node[5]|'name' = 'C' : String
'age' = 53 : int
'eyes' = 'green' : String
}
MARRIED
KNOWS
KNOWS
{ Node[2]|'name' = 'E' : String
'age' = 41 : int
'eyes' = 'blue' : String
}
{ Node[1]|'name' = 'D' : String
'age' = 54 : int
'eyes' = 'brown' : String
}

Query

```
start a=(3), c=(2) match p=a-->b-->c return RELATIONSHIPS(p)
```

All the nodes in the path p.

Result

```
+---------------------------------------+
| RELATIONSHIPS(p)                      |
+---------------------------------------+
| List(Relationship[0], Relationship[4]) |
+---------------------------------------+
1 rows, 3 ms
```

Chapter 5. Neo4j Server

5.1. Server Installation

Neo4j can be installed as a server, running either as a headless application or system service.

1. Download the latest release from http://neo4j.org/download
    - select the appropriate version for your platform
2. Extract the contents of the archive
    - refer to the top-level extracted directory as `NEO4J-HOME`
3. Use the scripts in the `bin` directory
    - for Linux/MacOS, run `$NEO4J_HOME/bin/neo4j start`
    - for Windows, double-click on `%NEO4J_HOME%\bin\Neo4j.bat`
4. Refer to the packaged information in the `doc` directory for details

5.1.1. As a Windows service

With administrative rights, Neo4j can be installed as a Windows service.

1. Click Start –> All Programs –> Accessories
2. Right click Command Prompt –> Run as Administrator
3. Provide authorization and/or the Administrator password
4. Navigate to `%NEO4J_HOME%`
5. Run `bin\Neo4j.bat install`

To uninstall, run `bin\Neo4j.bat remove` as Administrator.

To query the status of the service, run `bin\Neo4j.bat query`

To start/stop the service from the command prompt, run `bin\Neo4j.bat +action+`

5.1.2. Linux Service

Neo4j can participate in the normal system startup and shutdown process. The following procedure should work on most popular Linux distributions:

1. `cd $NEO4J_HOME`
2. `sudo ./bin/neo4j install`
   if asked, enter your password to gain super-user privileges
3. `service neo4j-service status`
   should indicate that the server is not running
4. `service neo4j-service start`
   will start the server

   During installation you will be given the option to select the user Neo4j will run as. You will be asked to supply a username (defaulting to `neo4j`) and if that user is not present on the system it will be created as a system account and the `$NEO4J_HOME/data` directory will be `chown`'ed to that user.

   You are encouraged to create a dedicated user for running the service and for that reason it is suggested that you unpack the distribution package under `/opt` or your site specific optional packages directory.

   Finally, note that if you chose to create a new user account, on uninstall you will be prompted to remove it from the system.

5.1.3. Mac OSX Service

Neo4j can be installed as a Mac launchd job:

1. `cd $NEO4J_HOME`
2. `sudo ./bin/neo4j install`
   if asked, enter your password to gain super-user privileges
3. `launchctl load ~/Library/LaunchAgents/wrapper.neo4j-server.plist`
   needed to tell launchd about the "job"
4. `launchctl list | grep neo`
   should reveal the launchd "wrapper.neo4j-server" job for running the Neo4j Server
5. `launchctl start wrapper.neo4j-server`
   to start the Neo4j Server under launchd control
6. `./bin/neo4j status`
   should indicate that the server is running

5.1.4. Multiple Server instances on one machine

Neo4j can be set up to run as several instances on one machine, providing for instance several databases for development. To configure, install two instances of the Neo4j Server in two different directories. Before running the Windows install or startup, change in conf/neo4j-wrapper.conf

```
# Name of the service for the first instance
wrapper.name=neo4j_1
```

and for the second instance

```
# Name of the service for the second instance
wrapper.name=neo4j_2
```

in order not to get name clashes installing and starting the instances as services.

Also, the port numbers for the web administration and the servers should be changed to non-clashing values in conf/neo4j-server.properties:

Server 1 (port 7474):

```
org.neo4j.server.webserver.port=7474

org.neo4j.server.webadmin.data.uri=http://localhost:7474/db/data/

org.neo4j.server.webadmin.management.uri=http://localhost:7474/db/manage/
```

Server 2 (port 7475):

```
org.neo4j.server.webserver.port=7475

org.neo4j.server.webadmin.data.uri=http://localhost:7475/db/data/

org.neo4j.server.webadmin.management.uri=http://localhost:7475/db/manage/
```

5.2. Server Configuration

Quick info

- The server’s primary configuration file is found under *conf/neo4j-server.properties*
- The *conf/log4j.properties* file contains the default server logging configuration
- Low-level performance tuning parameters are found in *conf/neo4j.properties*
- Configuraion of the deamonizing wrapper are found in *conf/neo4j-wrapper.properties*

5.2.1. Important server configurations parameters

The main configuration file for the server can be found at *conf/neo4j-server.properties*. This file contains several important settings, and although the defaults are sensible administrators might choose to make changes (especially to the port settings).

Set the location on disk of the database directory like this:

```
org.neo4j.server.database.location=data/graph.db
```

Note

On Windows systems, absolute locations including drive letters need to read *"c:/data/db"*.

Specify the HTTP server port supporting data, administrative, and UI access:

```
org.neo4j.server.webserver.port=7474
```

Set the location of the round-robin database directory which gathers metrics on the running server instance:

```
org.neo4j.server.webadmin.rrdb.location=data/graph.db/../rrd
```

Set the URI path for the REST data API through which the database is accessed. For non-local access, use the full URI of your server e.g. http://example.org:7575/database. For local access use a relative URI, e.g. /db/data

```
org.neo4j.server.webadmin.data.uri=/db/data/
```

Setting the management URI for the administration API that the Webadmin tool uses. For non-local access, use the full URI of your server e.g. http://example.org:7575/database/management. For local access use a relative URI, e.g. /db/manage

```
org.neo4j.server.webadmin.management.uri=/db/manage
```

If you plan to connect to the Webadmin from other than localhost, put in the external hostname of your server instead of localhost, e.g. http://my.host:7474/db/manage. Force the server to use IPv4 network addresses, in *conf/neo4j-wrapper.conf* under the section *Java Additional Parameters* add a new paramter:

```
wrapper.java.additional.3=-Djava.net.preferIPv4Stack=true
```

Low-level performance tuning parameters can be explicitly set by referring to the following property:

```
org.neo4j.server.db.tuning.properties=neo4j.properties
```

If this property isn’t set, the server will look for a file called *neo4j.properties* in the same directory as the *neo4j-server.properties* file.

If this property isn't set, and there is no *neo4j.properties* file in the default configuration directory, then the server will log a warning. Subsequently at runtime the database engine will attempt tune itself based on the prevailing conditions.

5.2.2. Neo4j Database performance configuration

The fine-tuning of the low-level Neo4j graph database engine is specified in a separate properties file, *conf/neo4j.properties*.

The graph database engine has a range of performance tuning options which are enumerated in Section 5.6, "Server Performance Tuning". Note that other factors than Neo4j tuning should be considered when performance tuning a server, including general server load, memory and file contention, and even garbage collection penalties on the JVM, though such considerations are beyond the scope of this configuration document.

5.2.3. Logging configuration

The logging framework in use by the Neo4j server is `java.util.logging` `<http://download.oracle.com/javase/6/docs/technotes/guides/logging/overview.html>` and is configured in *conf/logging.properties*.

By default it is setup to print `INFO` level messages both on screen and in a rolling file in *data/log*. Most deployments will choose to use their own configuration here to meet local standards. During development, much useful information can be found in the logs so some form of logging to disk is well worth keeping. On the other hand, if you want to completely silence the console output, set:

```
java.util.logging.ConsoleHandler.level=OFF
```

By default log files are rotated at approximately 10Mb and named consecutively neo4j.<id>.<rotation sequence #>.log To change the naming scheme, rotation frequency and backlog size modify

```
java.util.logging.FileHandler.pattern
java.util.logging.FileHandler.limit
java.util.logging.FileHandler.count
```

respectively to your needs. Details are available at the Javadoc for `java.util.logging.FileHandler` `<http://download.oracle.com/javase/6/docs/api/java/util/logging/FileHandler.html>`.

Apart from log statements originating from the Neo4j server, other libraries report their messages through various frameworks.

Zookeeper is hardwired to use the log4j logging framework. The bundled *conf/log4j.properties* applies for this use only and uses a rolling appender and outputs logs by default to the *data/log* directory.

5.2.4. Other configuration options

Enabling logging from the garbage collector

To get garbage collection logging output you have to pass the corresponding option to the server JVM executable by setting in *conf/neo4j-wrapper.conf* the value

```
wrapper.java.additional.3=-Xloggc:data/log/neo4j-gc.log
```

This line is already present and needs uncommenting. Note also that logging is not directed to console ; You will find the logging statements in *data/log/ne4j-gc.log* or whatever directory you set at the option.

5.3. 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-wrapper.properties* file.

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

```
# Java Additional Parameters
wrapper.java.additional.1=-Dorg.neo4j.server.properties=conf/neo4j-server.properties
wrapper.java.additional.2=-Dlog4j.configuration=file:conf/log4j.properties
wrapper.java.additional.3=-agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=5005 -Xdebug-Xnoagent-
```

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.

5.4. Starting the Neo4j server in high availability mode

Note

The High Availability features are only available in the Neo4j Enterprise Edition.

To run the Neo4j server in high availability mode there are two things you need to do. You have to configure the server to start up the database in high availability mode and you have to configure the Neo4j database for operating in high availability mode.

Instructing the server to start the database in high availability mode is as easy as setting the `org.neo4j.server.database.mode` property in the server properties file *(conf/neo-server.properties)* to `ha`. The default value for this parameter is `single`, which will start the database in standalone mode without participating in a cluster, still giving you Online Backup.

Configuring the Neo4j database for operating in high availability mode requires specifying a few properties in *conf/neo4j.properties*. First you need to specify `ha.machine_id`, this is a positive integer id that uniquely identifies this server in the cluster.

Example: `ha.machine_id = 1`

Then you have to specify `ha.zoo_keeper_servers`, this is a comma separated list of hosts and ports for communicating with each member of the Neo4j Coordinator cluster.

For example: `ha.zoo_keeper_servers = neo4j-manager-01:2180,neo4j-manager-02:2180,neo4j-manager-03:2180`.

You can also, optionally, configure the `ha.cluster_name`. This is the name of the cluster this instance is supposed to join. Accepted characters are alphabetical, numerical, dot, dash, and underscore. This configuration is useful if you have multiple Neo4j HA clusters managed by the same Coordinator cluster.

Example: `ha.cluster_name = my_neo4j_ha_cluster`

5.4.1. Starting a Neo4j Coordinator

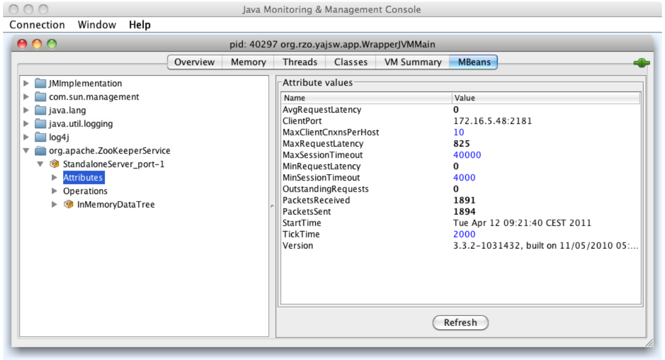
A Neo4j Coordinator cluster provides the Neo4j HA Data cluster with reliable coordination of lifecycle activities, like electing the master. Neo4j Server includes everything needed for running a Neo4j Coordinator.

Configuration of a Coordinator is specified in these files:

- *conf/coord.cfg* - coordinator operational settings
- *data/coordinator/myid* - unqiue identification of the coordinator

Once a Neo4j Coordinator instance has been configured, you can use the `bin/neo4j-coordinator` command to start the Neo4j Coordinator server on all desired servers with the same configuration, just changing the *data/coordinator/myid* to unique numbers. You can check that the coordinator is up by running `jconsole`, attaching to the JVM and check for `org.apache.zookeeper` MBeans.

Figure 5.1. Neo4j Coordinator MBeans View



Name	Value
AvgRequestLatency	0
ClientPort	172.16.5.48:2181
MaxClientCnxnsPerHost	10
MaxRequestLatency	825
MaxSessionTimeout	40000
MinRequestLatency	0
MinSessionTimeout	4000
OutstandingRequests	0
PacketsReceived	1891
PacketsSent	1894
StartTime	Tue Apr 12 09:21:40 CEST 2011
TickTime	2000
Version	3.3.2-1031432, built on 11/05/2010 05:...

5.4.2. Starting the Neo4j Server

Once the desired neo4j Coordinators are up and running, you are ready to start your Neo4j HA instance using `bin/neo4j start`. The details of the HA logs are available in the *messages.log* of the graph database data directory, normally *data/graph.db/mesages.log*. You should see an entry like this one:

```
Tue Apr 12 09:25:58 CEST 2011: MasterServer communication server started and bound to 6361
Tue Apr 12 09:25:58 CEST 2011: Started as master
Tue Apr 12 09:25:58 CEST 2011: master-rebound set to 1
```

5.5. Using the server with an embedded database

Even if you are using the Neo4j Java API directly, for instance via `EmbeddedGraphDatabase` or `HighlyAvailableGraphDatabase`, you can still use the features the server provides.

The Neo4j server exposes a class called WrappingNeoServerBootstrapper <http://components.neo4j.org/neo4j-server/1.4.2/apidocs/org/neo4j/server/WrappingNeoServerBootstrapper.html> , which is capable of starting a Neo4j server in the same process as your application. It uses an AbstractGraphDatabase <http://components.neo4j.org/neo4j-kernel/1.4.2/apidocs/org/neo4j/kernel/AbstractGraphDatabase.html> instance that you provide.

This gives your application, among other things, the REST API, statistics gathering and the web administration interface that comes with the server.

Usage example.

```
WrappingNeoServerBootstrapper srv = new WrappingNeoServerBootstrapper(
        myDb );

srv.start();

// Server is now running in background threads

srv.stop();
```

5.5.1. Providing custom configuration

You can modify the server settings programmatically and, within reason, the same settings are available to you here as those outlined in Section 5.2, “Server Configuration”.

The settings that are not available (or rather, that are ignored) are those that concern the underlying database, such as database location and database configuration path.

Custom configuration example.

```
EmbeddedServerConfigurator config = new EmbeddedServerConfigurator(
        myDb );
config.configuration().setProperty(
        Configurator.WEBSERVER_PORT_PROPERTY_KEY, 7575 );

WrappingNeoServerBootstrapper srv = new WrappingNeoServerBootstrapper(
        myDb, config );

srv.start();
```

5.5.2. Installing

To run the server all the libraries you need are in the *system/lib/* directory of the download package <http://neo4j.org/download/>. For further instructions, see Section 12.1, “Include Neo4j in your project”. The only difference to the embedded setup is that *system/lib/* should be added as well, not only the *lib/* directory.

For users of dependency management, an example for Maven follows.

Maven pom.xml snippet.

```
<dependencies>
  <dependency>
    <groupId>org.neo4j.app</groupId>
    <artifactId>neo4j-server</artifactId>
    <version>${neo4j-version}</version>
  </dependency>
```

```
</dependencies>
<repositories>
  <repository>
    <id>neo4j-release-repository</id>
    <name>Neo4j Maven 2 release repository</name>
    <url>http://m2.neo4j.org/releases</url>
    <releases>
      <enabled>true</enabled>
    </releases>
    <snapshots>
      <enabled>false</enabled>
    </snapshots>
  </repository>
</repositories>
```

Where `${neo4j-version}` is the intended version.

5.6. Server Performance Tuning

At the heart of the Neo4j server is a regular Neo4j storage engine instance. That engine can be tuned in the same way as the other embedded configurations, using the same file format. The only difference is that the server must be told where to find the fine-tuning configuration.

Quick info

- The neo4j.properties file is a standard configuration file that databases load in order to tune their memory use and caching strategies.
- See Section 2.1, “Caches in Neo4j” for more information.

5.6.1. Specifying Neo4j tuning properties

The `conf/neo4j-server.properties` file in the server distribution, is the main configuration file for the server. In this file we can specify a second properties file that contains the database tuning settings (that is, the `neo4j.properties` file). This is done by setting a single property to point to a valid `neo4j.properties` file:

```
org.neo4j.server.db.tuning.properties={neo4j.properties file}
```

On restarting the server the tuning enhancements specified in the `neo4j.properties` file will be loaded and configured into the underlying database engine.

5.6.2. Specifying JVM tuning properties

Tuning the standalone server is achieved by editing the `neo4j-wrapper.conf` file in the `conf` directory of `NEO4J_HOME`.

Edit the following properties:

Table 5.1. neo4j-wrapper.conf JVM tuning properties

Property Name	Meaning
`wrapper.java.initmemory`	initial heap size (in MB)
`wrapper.java.maxmemory`	maximum heap size (in MB)
`wrapper.java.additional.N`	additional literal JVM parameter, where N is a number for each

For more information on the tuning properties, see Section 2.2, “JVM Settings”.

Chapter 6. REST API

The Neo4j REST API is designed with discoverability in mind, so that you can start with a `GET` on the Section 6.1, “Service root” and from there discover URIs to perform other requests. The examples below uses URIs in the examples; they are subject to change in the future, so for future-proofness *discover URIs where possible*, instead of relying on the current layout. The default representation is json <http://www.json.org/>, both for responses and for data sent with `POST`/`PUT` requests.

Below follows a listing of ways to interact with the REST API. You can also see a (at runtime) generated description of the API be pointing your browser to the (exact URI may vary) http://localhost:7474/db/data/application.wadl

To interact with the JSON interface you must explicitly set the request header `Accept:application/json` for those requests that responds with data. You should also set the header `Content-Type:application/json` if your request sends data, for example when you’re creating a relationship. The examples include the relevant request and response headers.

6.1. Service root

6.1.1. Get service root

The service root is your starting point to discover the REST API.

Example request

- **GET** `http://0.0.0.0:7474/db/data/`
- **Accept:** `application/json`

Example response

- **200:** `OK`
- **Content-Type:** `application/json`

```
{
  "relationship_index" : "http://0.0.0.0:7474/db/data/index/relationship",
  "node" : "http://0.0.0.0:7474/db/data/node",
  "relationship_types" : "http://0.0.0.0:7474/db/data/relationship/types",
  "batch" : "http://0.0.0.0:7474/db/data/batch",
  "extensions_info" : "http://0.0.0.0:7474/db/data/ext",
  "node_index" : "http://0.0.0.0:7474/db/data/index/node",
  "reference_node" : "http://0.0.0.0:7474/db/data/node/0",
  "extensions" : {
  }
}
```

6.2. Nodes

6.2.1. Create Node

Example request

- **`POST`** `http://0.0.0.0:7474/db/data/node`
- **`Accept:`** `application/json`

Example response

- **`201:`** `Created`
- **`Content-Type:`** `application/json`
- **`Location:`** `http://0.0.0.0:7474/db/data/node/1`

```
{
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/1/relationships/out",
  "data" : {
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/1/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/1/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/1/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/1",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/1/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/1/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/1/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/1/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/1/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/1/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/1/relationships/in/{-list|&|types}"
}
```

6.2.2. Create Node with properties

Example request

- **`POST`** `http://0.0.0.0:7474/db/data/node`
- **`Accept:`** `application/json`
- **`Content-Type:`** `application/json`

```
{"foo" : "bar"}
```

Example response

- **`201:`** `Created`
- **`Content-Length:`** `1072`
- **`Content-Type:`** `application/json`
- **`Location:`** `http://0.0.0.0:7474/db/data/node/2`

```
{
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/2/relationships/out",
  "data" : {
    "foo" : "bar"
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/2/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/2/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/2/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/2",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/2/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/2/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/2/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/2/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/2/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/2/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/2/relationships/in/{-list|&|types}"
}
```

6.2.3. Get node

Note that the response contains URI/templates for the available operations for getting properties and relationships.

Example request

- **GET** `http://0.0.0.0:7474/db/data/node/1`
- **Accept:** `application/json`

Example response

- **200:** `OK`
- **Content-Type:** `application/json`

```
{
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/1/relationships/out",
  "data" : {
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/1/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/1/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/1/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/1",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/1/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/1/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/1/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/1/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/1/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/1/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/1/relationships/in/{-list|&|types}"
}
```

6.2.4. Get non-existent node

Example request

- **GET** `http://0.0.0.0:7474/db/data/node/600000`
- **Accept:** `application/json`

Example response

- **404:** `Not Found`
- **Content-Type:** `application/json`

```
{
  "message" : "Cannot find node with id [600000] in database.",
  "exception" : "org.neo4j.server.rest.web.NodeNotFoundException: Cannot find node with id [600000] in databas
  "stacktrace" : [ "org.neo4j.server.rest.web.DatabaseActions.node(DatabaseActions.java:101)", "org.neo4j.serv
}
```

6.2.5. Delete node

Example request

- **DELETE** `http://0.0.0.0:7474/db/data/node/9`
- **Accept:** `application/json`

Example response

- **204:** `No Content`

6.2.6. Nodes with relationships can not be deleted

The relationships on a node has to be deleted before the node can be deleted.

Example request

- **DELETE** `http://0.0.0.0:7474/db/data/node/10`
- **Accept:** `application/json`

Example response

- **409:** `Conflict`
- **Content-Type:** `application/json`

```
{
  "message" : "The node with id 10 cannot be deleted. Check that the node is orphaned before deletion.",
  "exception" : "org.neo4j.server.rest.web.OperationFailureException: The node with id 10 cannot be deleted. C
  "stacktrace" : [ "org.neo4j.server.rest.web.DatabaseActions.deleteNode(DatabaseActions.java:226)", "org.neo4
}
```

6.3. Relationships

The general pattern to get relationships is:

```
GET http://localhost:7474/db/data/node/123/relationships/{dir}/{-list|&|types}
```

Where `dir` is one of `all`, `in`, `out` and `types` is an ampersand-separated list of types. See the examples below for more information.

6.3.1. Create relationship

Example request

- **POST** `http://0.0.0.0:7474/db/data/node/3/relationships`
- **Accept:** `application/json`
- **Content-Type:** `application/json`

```
{"to" : "http://0.0.0.0:7474/db/data/node/4", "type" : "LOVES"}
```

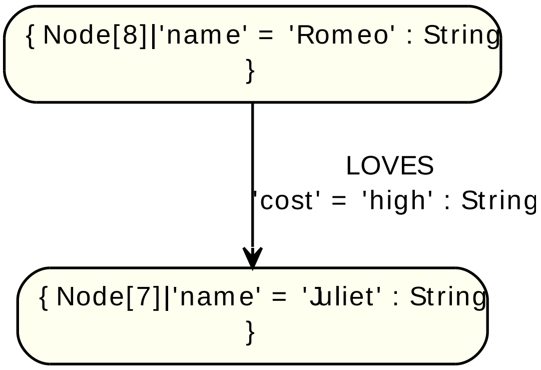
Example response

- **201:** `Created`
- **Content-Type:** `application/json`
- **Location:** `http://0.0.0.0:7474/db/data/relationship/2`

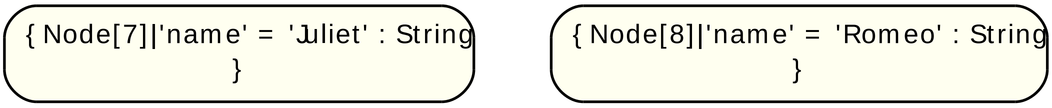
```
{
  "start" : "http://0.0.0.0:7474/db/data/node/3",
  "data" : {
  },
  "self" : "http://0.0.0.0:7474/db/data/relationship/2",
  "property" : "http://0.0.0.0:7474/db/data/relationship/2/properties/{key}",
  "properties" : "http://0.0.0.0:7474/db/data/relationship/2/properties",
  "type" : "LOVES",
  "extensions" : {
  },
  "end" : "http://0.0.0.0:7474/db/data/node/4"
}
```

6.3.2. Delete relationship

Starting Graph:


{ Node[8]|'name' = 'Romeo' : String
}
LOVES
'cost' = 'high' : String
{ Node[7]|'name' = 'Juliet' : String
}

Final Graph:


{ Node[7]|'name' = 'Juliet' : String }
{ Node[8]|'name' = 'Romeo' : String }

Example request

- **DELETE** `http://localhost:7474/db/data/relationship/3`
- **Accept:** `application/json`

Example response

- **204:** `No Content`

6.3.3. Get all relationships

Example request

- **GET** `http://0.0.0.0:7474/db/data/node/6/relationships/all`
- **Accept:** `application/json`

Example response

- **200:** `OK`
- **Content-Type:** `application/json`

```
[ {
  "start" : "http://0.0.0.0:7474/db/data/node/6",
  "data" : {
  },
  "self" : "http://0.0.0.0:7474/db/data/relationship/3",
  "property" : "http://0.0.0.0:7474/db/data/relationship/3/properties/{key}",
  "properties" : "http://0.0.0.0:7474/db/data/relationship/3/properties",
  "type" : "LIKES",
  "extensions" : {
  },
  "end" : "http://0.0.0.0:7474/db/data/node/7"
}, {
  "start" : "http://0.0.0.0:7474/db/data/node/8",
  "data" : {
  },
  "self" : "http://0.0.0.0:7474/db/data/relationship/4",
  "property" : "http://0.0.0.0:7474/db/data/relationship/4/properties/{key}",
  "properties" : "http://0.0.0.0:7474/db/data/relationship/4/properties",
  "type" : "LIKES",
  "extensions" : {
  },
  "end" : "http://0.0.0.0:7474/db/data/node/6"
}, {
  "start" : "http://0.0.0.0:7474/db/data/node/6",
  "data" : {
  },
  "self" : "http://0.0.0.0:7474/db/data/relationship/5",
  "property" : "http://0.0.0.0:7474/db/data/relationship/5/properties/{key}",
  "properties" : "http://0.0.0.0:7474/db/data/relationship/5/properties",
  "type" : "HATES",
  "extensions" : {
  },
  "end" : "http://0.0.0.0:7474/db/data/node/9"
} ]
```

6.3.4. Get incoming relationships

Example request

- **GET** `http://0.0.0.0:7474/db/data/node/11/relationships/in`
- **Accept:** `application/json`

Example response

- **200:** `OK`
- **Content-Type:** `application/json`

```
[ {
  "start" : "http://0.0.0.0:7474/db/data/node/13",
  "data" : {
  },
  "self" : "http://0.0.0.0:7474/db/data/relationship/7",
  "property" : "http://0.0.0.0:7474/db/data/relationship/7/properties/{key}",
  "properties" : "http://0.0.0.0:7474/db/data/relationship/7/properties",
  "type" : "LIKES",
  "extensions" : {
  },
  "end" : "http://0.0.0.0:7474/db/data/node/11"
} ]
```

6.3.5. Get outgoing relationships

Example request

- **GET** `http://0.0.0.0:7474/db/data/node/16/relationships/out`
- **Accept:** `application/json`

Example response

- **200:** `OK`
- **Content-Type:** `application/json`

```
[ {
  "start" : "http://0.0.0.0:7474/db/data/node/16",
  "data" : {
  },
  "self" : "http://0.0.0.0:7474/db/data/relationship/9",
  "property" : "http://0.0.0.0:7474/db/data/relationship/9/properties/{key}",
  "properties" : "http://0.0.0.0:7474/db/data/relationship/9/properties",
  "type" : "LIKES",
  "extensions" : {
  },
  "end" : "http://0.0.0.0:7474/db/data/node/17"
}, {
  "start" : "http://0.0.0.0:7474/db/data/node/16",
  "data" : {
  },
  "self" : "http://0.0.0.0:7474/db/data/relationship/11",
  "property" : "http://0.0.0.0:7474/db/data/relationship/11/properties/{key}",
  "properties" : "http://0.0.0.0:7474/db/data/relationship/11/properties",
  "type" : "HATES",
  "extensions" : {
  },
  "end" : "http://0.0.0.0:7474/db/data/node/19"
} ]
```

6.3.6. Get typed relationships

Note that the "&" needs to be escaped for example when using cURL <http://curl.haxx.se/> from the terminal.

Example request

- **GET** `http://0.0.0.0:7474/db/data/node/21/relationships/all/LIKES&HATES`
- **Accept:** `application/json`

Example response

- **200:** `OK`
- **Content-Type:** `application/json`

```
[ {
  "start" : "http://0.0.0.0:7474/db/data/node/21",
  "data" : {
  },
  "self" : "http://0.0.0.0:7474/db/data/relationship/12",
  "property" : "http://0.0.0.0:7474/db/data/relationship/12/properties/{key}",
  "properties" : "http://0.0.0.0:7474/db/data/relationship/12/properties",
  "type" : "LIKES",
  "extensions" : {
  },
  "end" : "http://0.0.0.0:7474/db/data/node/22"
}, {
  "start" : "http://0.0.0.0:7474/db/data/node/23",
  "data" : {
  },
  "self" : "http://0.0.0.0:7474/db/data/relationship/13",
  "property" : "http://0.0.0.0:7474/db/data/relationship/13/properties/{key}",
  "properties" : "http://0.0.0.0:7474/db/data/relationship/13/properties",
  "type" : "LIKES",
  "extensions" : {
  },
  "end" : "http://0.0.0.0:7474/db/data/node/21"
}, {
  "start" : "http://0.0.0.0:7474/db/data/node/21",
  "data" : {
  },
  "self" : "http://0.0.0.0:7474/db/data/relationship/14",
  "property" : "http://0.0.0.0:7474/db/data/relationship/14/properties/{key}",
  "properties" : "http://0.0.0.0:7474/db/data/relationship/14/properties",
  "type" : "HATES",
  "extensions" : {
  },
  "end" : "http://0.0.0.0:7474/db/data/node/24"
} ]
```

6.3.7. Get relationships on a node without relationships

Example request

- **GET** `http://0.0.0.0:7474/db/data/node/40/relationships/all`
- **Accept:** `application/json`

Example response

- **200:** `OK`

- **Content-Type:** application/json

```
[ ]
```

6.4. Relationship types

6.4.1. Get relationship types

Example request

- **`GET`** `http://0.0.0.0:7474/db/data/relationship/types`
- **`Accept:`** `application/json`

Example response

- **`200:`** `OK`
- **`Content-Type:`** `application/json`

```
["foo","bar"]
```

6.5. Node properties

6.5.1. Set property on node

Example request

- **PUT** `http://0.0.0.0:7474/db/data/node/5/properties/foo`
- **Accept:** `application/json`
- **Content-Type:** `application/json`

```
"bar"
```

Example response

- **204:** `No Content`

6.5.2. Update node properties

Example request

- **PUT** `http://0.0.0.0:7474/db/data/node/1/properties`
- **Accept:** `application/json`
- **Content-Type:** `application/json`

```
{
  "jim" : "tobias"
}
```

Example response

- **204:** `No Content`

6.5.3. Get properties for node

Example request

- **GET** `http://0.0.0.0:7474/db/data/node/5/properties`
- **Accept:** `application/json`

Example response

- **200:** `OK`
- **Content-Type:** `application/json`

```
{
  "foo" : "bar"
}
```

6.5.4. Get properties for node (empty result)

If there are no properties, there will be an HTTP 204 response.

Example request

- **GET** `http://0.0.0.0:7474/db/data/node/1/properties`
- **Accept:** `application/json`

Example response

- **204:** `No Content`

6.5.5. Property values can not be null

This example shows the response you get when trying to set a property to null.

Example request

- **POST** `http://0.0.0.0:7474/db/data/node`
- **Accept:** `application/json`
- **Content-Type:** `application/json`

```
{"foo":null}
```

Example response

- **400:** `Bad Request`
- **Content-Type:** `application/json`

```
{
  "message" : "Could not set property \"foo\", unsupported type: null",
  "exception" : "org.neo4j.server.rest.web.PropertyValueException: Could not set property \"foo\", unsupported
  "stacktrace" : [ "org.neo4j.server.rest.web.DatabaseActions.set(DatabaseActions.java:133)", "org.neo4j.serve
}
```

6.5.6. Property values can not be nested

Nesting properties is not supported. You could for example store the nested json as a string instead.

Example request

- **POST** `http://0.0.0.0:7474/db/data/node`
- **Accept:** `application/json`
- **Content-Type:** `application/json`

```
{"foo" : {"bar" : "baz"}}
```

Example response

- **400:** `Bad Request`
- **Content-Type:** `application/json`

```
{
  "message" : "Could not set property \"foo\", unsupported type: {bar=baz}",
  "exception" : "org.neo4j.server.rest.web.PropertyValueException: Could not set property \"foo\", unsupported
  "stacktrace" : [ "org.neo4j.server.rest.web.DatabaseActions.set(DatabaseActions.java:133)", "org.neo4j.serve
}
```

6.5.7. Delete all properties from node

Example request

- **DELETE** `http://0.0.0.0:7474/db/data/node/2/properties`
- **Accept:** `application/json`

Example response

- **204:** `No Content`

6.6. Relationship properties

6.6.1. Update relationship properties

Example request

- **`PUT`** `http://0.0.0.0:7474/db/data/relationship/0/properties`
- **`Accept:`** `application/json`
- **`Content-Type:`** `application/json`

```
{
  "jim" : "tobias"
}
```

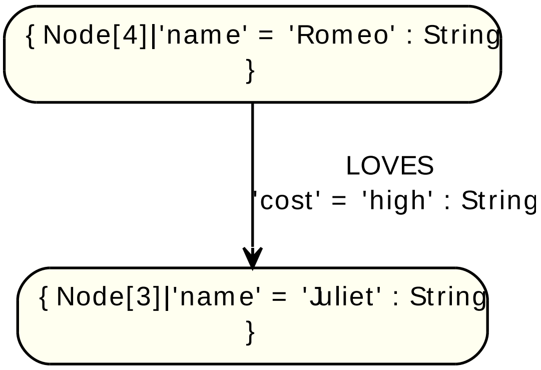
Example response

- **`204:`** `No Content`

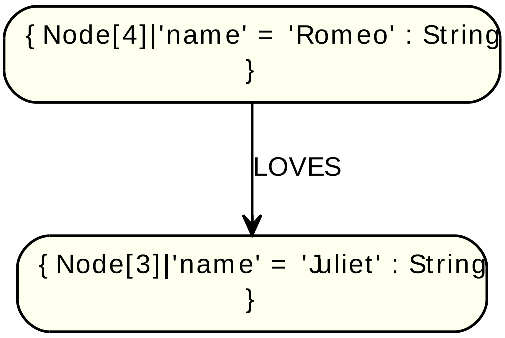
inlcude::remove-properties-from-a-relationship.txt[]

6.6.2. Remove property from a relationship

Starting Graph:


{ Node[4]|'name' = 'Romeo' : String }
LOVES
'cost' = 'high' : String
{ Node[3]|'name' = 'Juliet' : String }

Final Graph:


{ Node[4]|'name' = 'Romeo' : String }
LOVES
{ Node[3]|'name' = 'Juliet' : String }

Example request

- **`DELETE`** `http://localhost:7474/db/data/relationship/1/properties/cost`
- **`Accept:`** `application/json`

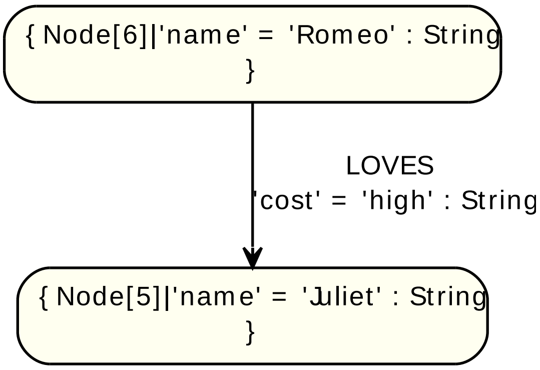
Example response

- **204:** `No Content`

6.6.3. Remove non-existent property from a relationship

Documentation not available

Final Graph:


{ Node[6]|'name' = 'Romeo' : String }
LOVES
'cost' = 'high' : String
{ Node[5]|'name' = 'Juliet' : String }

Example request

- **DELETE** `http://localhost:7474/db/data/relationship/2/properties/non-existent`
- **Accept:** `application/json`

Example response

- **404:** `Not Found`
- **Content-Type:** `application/json`

```
{
  "message" : "Relationship[2] does not have a property \"non-existent\"",
  "exception" : "org.neo4j.server.rest.web.NoSuchPropertyException: Relationship[2] does not have a property \
  "stacktrace" : [ "org.neo4j.server.rest.web.DatabaseActions.removeRelationshipProperty(DatabaseActions.java:
}
```

6.6.4. Remove properties from a non-existing relationship

Final Graph:

Example request

- **DELETE** `http://localhost:7474/db/data/relationship/1234/properties`
- **Accept:** `application/json`

Example response

- **404:** `Not Found`
- **Content-Type:** `application/json`

```
{
  "exception" : "org.neo4j.server.rest.web.RelationshipNotFoundException",
  "stacktrace" : [ "org.neo4j.server.rest.web.DatabaseActions.relationship(DatabaseActions.java:115)", "org.ne
}
```

6.6.5. Remove property from a non-existing relationship

Final Graph:

Example request

- **DELETE** `http://localhost:7474/db/data/relationship/1234/properties/cost`
- **Accept:** `application/json`

Example response

- **404:** `Not Found`
- **Content-Type:** `application/json`

```
{
  "exception" : "org.neo4j.server.rest.web.RelationshipNotFoundException",
  "stacktrace" : [ "org.neo4j.server.rest.web.DatabaseActions.relationship(DatabaseActions.java:115)", "org.ne
}
```

6.7. Indexes

An index can contain either nodes or relationships.

Note
To create an index with default configuration, simply start using it by adding nodes/relationships to it. It will then be automatically created for you.

What default configuration means depends on how you have configured your database. If you haven't changed any indexing configuration, it means the indexes will be using a Lucene-based backend.

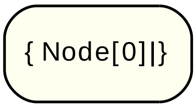
All the examples below show you how to do operations on node indexes, but all of them are just as applicable to relationship indexes. Simple change the "node" part of the URL to "relationship".

If you want to customize the index settings, see Section 6.7.2, "Create node index with configuration".

6.7.1. Create node index

Note
Instead of creating the index this way, you can simply start to use it, and it will be created automatically.

Final Graph:



Example request

- **POST** `http://0.0.0.0:7474/db/data/index/node/`
- **Accept:** `application/json`
- **Content-Type:** `application/json`

```
{
  "name" : "favorites"
}
```

Example response

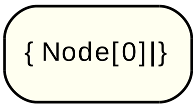
- **201:** `Created`
- **Content-Type:** `application/json`
- **Location:** `http://0.0.0.0:7474/db/data/index/node/favorites/`

```
{
  "template" : "http://0.0.0.0:7474/db/data/index/node/favorites/{key}/{value}"
}
```

6.7.2. Create node index with configuration

This request is only necessary if you want to customize the index settings. If you are happy with the defaults, you can just start indexing nodes/relationships, as non-existent indexes will automatically be created as you do. See Section 7.10, "Configuration and fulltext indexes" for more information on index configuration.

Final Graph:



Example request

- **POST** `http://0.0.0.0:7474/db/data/index/node/`
- **Accept:** `application/json`
- **Content-Type:** `application/json`

```
{"name":"fulltext", "config":{"type":"fulltext","provider":"lucene"}}
```

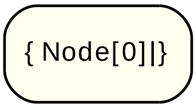
Example response

- **201:** `Created`
- **Content-Type:** `application/json`
- **Location:** `http://0.0.0.0:7474/db/data/index/node/fulltext/`

```
{
  "template" : "http://0.0.0.0:7474/db/data/index/node/fulltext/{key}/{value}",
  "provider" : "lucene",
  "type" : "fulltext"
}
```

6.7.3. Delete node index

Final Graph:



Example request

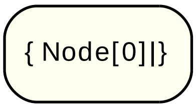
- **DELETE** `http://0.0.0.0:7474/db/data/index/node/kvnode`
- **Accept:** `application/json`

Example response

- **204:** `No Content`

6.7.4. List node indexes

Final Graph:



Example request

- **GET** `http://0.0.0.0:7474/db/data/index/node/`

- **Accept:** `application/json`

Example response

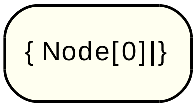
- **200:** `OK`
- **Content-Type:** `application/json`

```
{
  "favorites" : {
    "template" : "http://0.0.0.0:7474/db/data/index/node/favorites/{key}/{value}",
    "provider" : "lucene",
    "type" : "exact"
  }
}
```

6.7.5. List node indexes (empty result)

This is an example covering the case where no node index exists.

Final Graph:



Example request

- **GET** `http://0.0.0.0:7474/db/data/index/node/`
- **Accept:** `application/json`

Example response

- **204:** `No Content`

6.7.6. Add node to index

Associates a node with the given key/value pair in the given index.

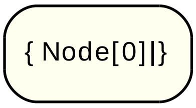
Note

Spaces in the URI have to be escaped.

Caution

This does **not** overwrite previous entries. If you index the same key/value/item combination twice, two index entries are created. To do update-type operations, you need to delete the old entry before adding a new one.

Final Graph:



Example request

- **POST** `http://0.0.0.0:7474/db/data/index/node/favorites/key/the%20value`

- **Accept:** `application/json`
- **Content-Type:** `application/json`

```
"http://0.0.0.0:7474/db/data/node/0"
```

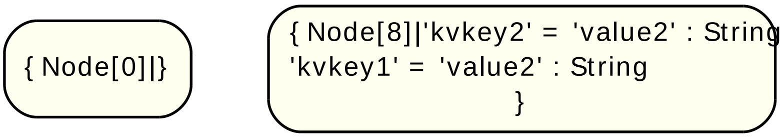
Example response

- **201:** `Created`
- **Content-Type:** `application/json`
- **Location:** `http://0.0.0.0:7474/db/data/index/node/favorites/key/the%20value/0`

```
{
  "indexed" : "http://0.0.0.0:7474/db/data/index/node/favorites/key/the%20value/0",
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/0/relationships/out",
  "data" : {
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/0/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/0/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/0/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/0",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/0/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/0/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/0/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/0/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/0/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/0/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/0/relationships/in/{-list|&|types}"
}
```

6.7.7. Remove all entries with a given node from an index

Final Graph:


{ Node[0]|}
{ Node[8]|'kvkey2' = 'value2' : String
'kvkey1' = 'value2' : String
}

Example request

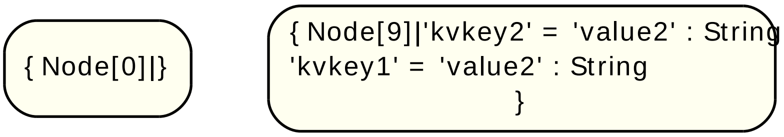
- **DELETE** `http://0.0.0.0:7474/db/data/index/node/kvnode/8`
- **Accept:** `application/json`

Example response

- **204:** `No Content`

6.7.8. Remove all entries with a given node and key from an index

Final Graph:


{ Node[0]|}
{ Node[9]|'kvkey2' = 'value2' : String
'kvkey1' = 'value2' : String
}

Example request

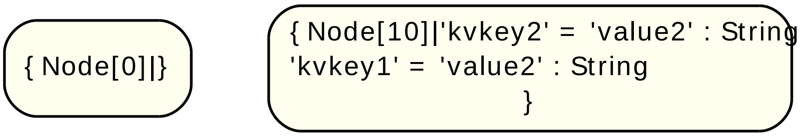
- **`DELETE`** `http://0.0.0.0:7474/db/data/index/node/kvnode/kvkey2/9`
- **`Accept:`** `application/json`

Example response

- **`204:`** `No Content`

6.7.9. Remove all entries with a given node, key and value from an index

Final Graph:


{ Node[0]|}
{ Node[10]|'kvkey2' = 'value2' : String
'kvkey1' = 'value2' : String
}

Example request

- **`DELETE`** `http://0.0.0.0:7474/db/data/index/node/kvnode/kvkey1/value1/10`
- **`Accept:`** `application/json`

Example response

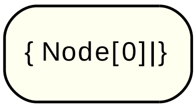
- **`204:`** `No Content`

6.7.10. Find node by exact match

Note

Spaces in the URI have to be escaped.

Final Graph:



Example request

- **`GET`** `http://0.0.0.0:7474/db/data/index/node/favorites/key/the%20value`
- **`Accept:`** `application/json`

Example response

- **`200:`** `OK`
- **`Content-Type:`** `application/json`

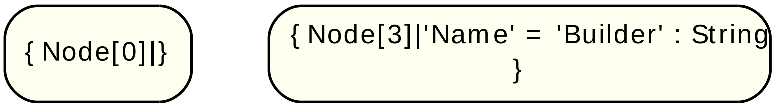
```
[ {
  "indexed" : "http://0.0.0.0:7474/db/data/index/node/favorites/key/the%20value/0",
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/0/relationships/out",
  "data" : {
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/0/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/0/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/0/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/0",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/0/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/0/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/0/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/0/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/0/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/0/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/0/relationships/in/{-list|&|types}"
} ]
```

6.7.11. Find node by query

The query language used here depends on what type of index you are querying. The default index type is Lucene, in which case you should use the Lucene query language here. Below and Example of a fuzzy search over multiple keys.

See: http://lucene.apache.org/java/3_1_0/queryparsersyntax.html

Final Graph:



Example request

- **GET** `http://0.0.0.0:7474/db/data/index/node/bobTheIndex?query=Name:Build~0.1%20AND%20Gender:Male`
- **Accept:** `application/json`

Example response

- **200:** `OK`
- **Content-Type:** `application/json`

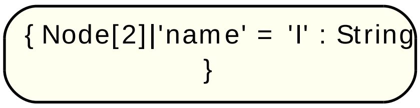
```
[ {
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/3/relationships/out",
  "data" : {
    "Name" : "Builder"
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/3/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/3/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/3/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/3",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/3/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/3/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/3/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/3/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/3/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/3/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/3/relationships/in/{-list|&|types}"
} ]
```

6.8. Auto-Indexes

6.8.1. Find node by exact match from an automatic index

Automatic index nodes can be found via exact lookups with normal Index REST syntax.

Final Graph:



Example request

- **GET** `http://localhost:7474/db/data/index/auto/node/name/I`
- **Accept:** `application/json`

Example response

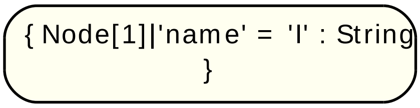
- **200:** `OK`
- **Content-Type:** `application/json`

```
[ {
  "outgoing_relationships" : "http://localhost:7474/db/data/node/2/relationships/out",
  "data" : {
    "name" : "I"
  },
  "traverse" : "http://localhost:7474/db/data/node/2/traverse/{returnType}",
  "all_typed_relationships" : "http://localhost:7474/db/data/node/2/relationships/all/{-list|&|types}",
  "property" : "http://localhost:7474/db/data/node/2/properties/{key}",
  "self" : "http://localhost:7474/db/data/node/2",
  "outgoing_typed_relationships" : "http://localhost:7474/db/data/node/2/relationships/out/{-list|&|types}",
  "properties" : "http://localhost:7474/db/data/node/2/properties",
  "incoming_relationships" : "http://localhost:7474/db/data/node/2/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://localhost:7474/db/data/node/2/relationships",
  "paged_traverse" : "http://localhost:7474/db/data/node/2/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://localhost:7474/db/data/node/2/relationships/all",
  "incoming_typed_relationships" : "http://localhost:7474/db/data/node/2/relationships/in/{-list|&|types}"
} ]
```

6.8.2. Find node by query from an automatic index

See Find node by query for the actual query syntax.

Final Graph:



Example request

- **GET** `http://localhost:7474/db/data/index/auto/node/?query=name:I`
- **Accept:** `application/json`

Example response

- **200:** OK
- **Content-Type:** application/json

```
[ {
  "outgoing_relationships" : "http://localhost:7474/db/data/node/1/relationships/out",
  "data" : {
    "name" : "I"
  },
  "traverse" : "http://localhost:7474/db/data/node/1/traverse/{returnType}",
  "all_typed_relationships" : "http://localhost:7474/db/data/node/1/relationships/all/{-list|&|types}",
  "property" : "http://localhost:7474/db/data/node/1/properties/{key}",
  "self" : "http://localhost:7474/db/data/node/1",
  "outgoing_typed_relationships" : "http://localhost:7474/db/data/node/1/relationships/out/{-list|&|types}",
  "properties" : "http://localhost:7474/db/data/node/1/properties",
  "incoming_relationships" : "http://localhost:7474/db/data/node/1/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://localhost:7474/db/data/node/1/relationships",
  "paged_traverse" : "http://localhost:7474/db/data/node/1/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://localhost:7474/db/data/node/1/relationships/all",
  "incoming_typed_relationships" : "http://localhost:7474/db/data/node/1/relationships/in/{-list|&|types}"
} ]
```

6.9. Configurable Auto-Indexing

Out of the box auto-indexing supports exact matches since they are created with the default configuration (http://docs.neo4j.org/chunked/snapshot/indexing-create.html) the first time you access them. However it is possible to intervene in the lifecycle of the server before any auto indexes are created to change their configuration.

This approach **cannot** be used on databases that already have auto-indexes established. To change the auto-index configuration existing indexes would have to be deleted first, so be careful!

Caution

This technique works, but it is not particularly pleasant. Future versions of Neo4j may remove this loophole in favour of a better structured feature for managing auto-indexing configurations.

Auto-indexing must be enabled through configuration before we can create or configure them. Firstly ensure that you've added some config like this into your server's `neo4j.properties` file:

```
node_auto_indexing=true
relationship_auto_indexing=true
node_keys_indexable=name,phone
relationship_keys_indexable=since
```

The `node_auto_indexing` and `relationship_auto_indexing` turn auto-indexing on for nodes and relationships respectively. The `node_keys_indexable` key allows you to specify a comma-separated list of node property keys to be indexed. The `relationship_keys_indexable` does the same for relationship property keys.

Next start the server as usual by invoking the start script as described in Section 5.1, "Server Installation".

Next we have to pre-empt the creation of an auto-index, by telling the server to create an apparently manual index which has the same name as the node (or relationship) auto-index. For example, in this case we'll create a node auto index whose name is `node_auto_index`, like so:

6.9.1. Create an auto index for nodes with specific configuration

Example request

- **`POST`** `http://0.0.0.0:7474/db/data/index/node/`
- **`Accept:`** `application/json`
- **`Content-Type:`** `application/json`

```
{"name":"node_auto_index", "config":{"type":"fulltext","provider":"lucene"}}
```

Example response

- **`201:`** `Created`
- **`Content-Type:`** `application/json`
- **`Location:`** `http://0.0.0.0:7474/db/data/index/node/node_auto_index/`

```
{
  "template" : "http://0.0.0.0:7474/db/data/index/node/node_auto_index/{key}/{value}",
  "provider" : "lucene",
  "type" : "fulltext"
}
```

If you require configured auto-indexes for relationships, the approach is similar:

6.9.2. Create an auto index for relationships with specific configuration

Example request

- **`POST`** `http://0.0.0.0:7474/db/data/index/relationship/`
- **`Accept:`** `application/json`
- **`Content-Type:`** `application/json`

```
{"name":"relationship_auto_index", "config":{"type":"fulltext","provider":"lucene"}}
```

Example response

- **`201:`** `Created`
- **`Content-Type:`** `application/json`
- **`Location:`** `http://0.0.0.0:7474/db/data/index/relationship/relationship_auto_index/`

```
{
  "template" : "http://0.0.0.0:7474/db/data/index/relationship/relationship_auto_index/{key}/{value}",
  "provider" : "lucene",
  "type" : "fulltext"
}
```

In case you're curious how this works, on the server side it triggers the creation of an index which happens to have the same name as the auto index that the database would create for itself. Now when we interact with the database, the index thinks the index is already is created so the state machine skips over that step and just gets on with normal day-to-day auto-indexing.

Caution

You have to do this early in your server lifecycle, before any normal auto indexes are created.

6.10. Traversals

Traversals are performed from a start node. The traversal is controlled by the URI and the body sent with the request.

returnType

The kind of objects in the response is determined by *traverse/{returnType}* in the URL. `returnType` can have one of these values:

- `node`
- `relationship`
- `path` - contains full representations of start and end node, the rest are URIs
- `fullpath` - contains full representations of all nodes and relationships

To decide how the graph should be traversed you can use these parameters in the request body:

order

Decides in which order to visit nodes. Possible values:

- `breadth_first` - see Breadth-first search <http://en.wikipedia.org/wiki/Breadth-first_search>
- `depth_first` - see Depth-first search <http://en.wikipedia.org/wiki/Depth-first_search>

relationships

Decides which relationship types and directions should be followed. The direction can be one of:

- `all`
- `in`
- `out`

uniqueness

Decides how uniqueness should be calculated. Possible values:

- `node_global`
- `none`
- `relationship_global`
- `node_path`
- `relationship_path`

prune_evaluator

Decides whether the traverser should continue down that path or if it should be pruned so that the traverser won't continue down that path. You can write your own prune evaluator as (see Section 6.10.1, "Traversal using a return filter" or use the `built-in` `none` prune evaluator.

return_filter

Decides whether the current position should be included in the result. You can provide your own code for this (see Section 6.10.1, "Traversal using a return filter"), or use one of the built-in filters:

- `all`
- `all_but_start_node`

max_depth

Is a short-hand way of specifying a prune evaluator which prunes after a certain depth. If not specified a max depth of 1 is used and if a `prune_evaluator` is specified instead of a `max_depth`, no max depth limit is set.

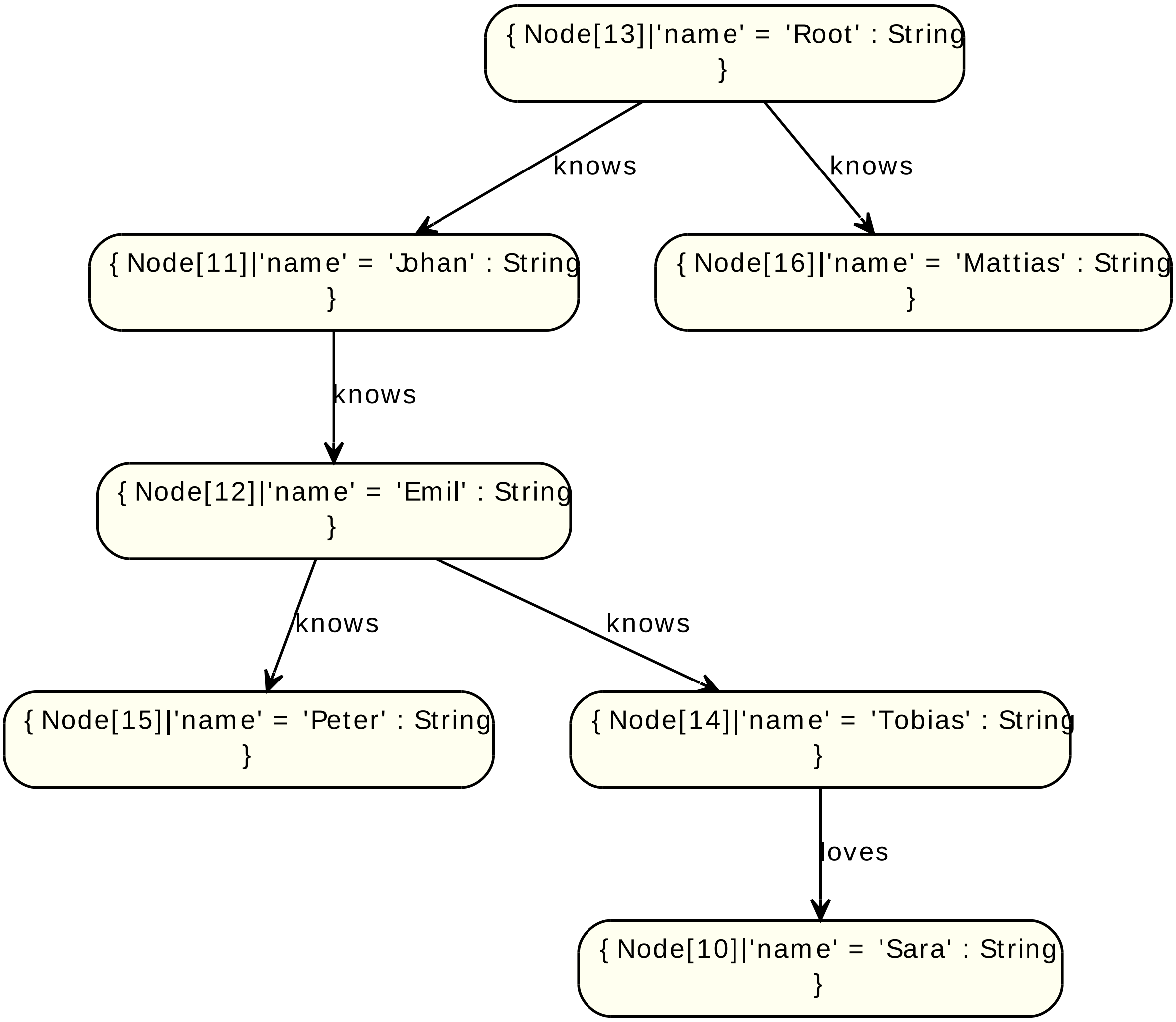
The `position` object in the body of the `return_filter` and `prune_evaluator` is a `Path` `<http://components.neo4j.org/neo4j/1.4.2/apidocs/org/neo4j/graphdb/Path.html>` object representing the path from the start node to the current traversal position.

Out of the box, the REST API supports JavaScript code in filters/evaluators. See the examples for the exact syntax of the request.

6.10.1. Traversal using a return filter

In this example, the `none` prune evaluator is used and a return filter is supplied in order to return all names containing "t". The result is to be returned as nodes and the max depth is set to 3.

Final Graph:


{ Node[13]|'name' = 'Root' : String }
knows
knows
{ Node[11]|'name' = 'Johan' : String }
{ Node[16]|'name' = 'Mattias' : String }
knows
{ Node[12]|'name' = 'Emil' : String }
knows
knows
{ Node[15]|'name' = 'Peter' : String }
{ Node[14]|'name' = 'Tobias' : String }
loves
{ Node[10]|'name' = 'Sara' : String }

Example request

- **POST** `http://localhost:7474/db/data/node/13/traverse/node`
- **Accept:** `application/json`
- **Content-Type:** `application/json`

```
{
  "order" : "breadth_first",
  "return_filter" : {
    "body" : "position.endNode().getProperty('name').toLowerCase().contains('t')",
    "language" : "javascript"
  },
  "prune_evaluator" : {
    "body" : "position.length() > 10",
    "language" : "javascript"
  },
  "uniqueness" : "node_global",
  "relationships" : [ {
    "direction" : "all",
    "type" : "knows"
  }, {
    "direction" : "all",
    "type" : "loves"
  } ],
  "max_depth" : 3
}
```

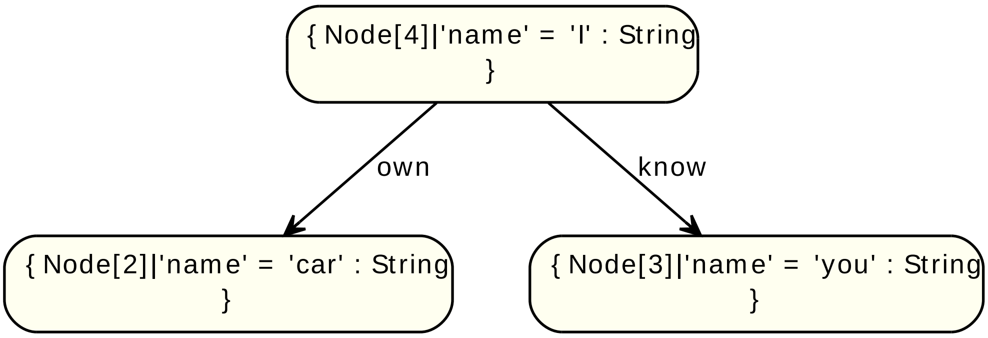
Example response

- **200:** OK
- **Content-Type:** application/json

```
[ {
  "outgoing_relationships" : "http://localhost:7474/db/data/node/13/relationships/out",
  "data" : {
    "name" : "Root"
  },
  "traverse" : "http://localhost:7474/db/data/node/13/traverse/{returnType}",
  "all_typed_relationships" : "http://localhost:7474/db/data/node/13/relationships/all/{-list|&|types}",
  "property" : "http://localhost:7474/db/data/node/13/properties/{key}",
  "self" : "http://localhost:7474/db/data/node/13",
  "outgoing_typed_relationships" : "http://localhost:7474/db/data/node/13/relationships/out/{-list|&|types}",
  "properties" : "http://localhost:7474/db/data/node/13/properties",
  "incoming_relationships" : "http://localhost:7474/db/data/node/13/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://localhost:7474/db/data/node/13/relationships",
  "paged_traverse" : "http://localhost:7474/db/data/node/13/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://localhost:7474/db/data/node/13/relationships/all",
  "incoming_typed_relationships" : "http://localhost:7474/db/data/node/13/relationships/in/{-list|&|types}"
}, {
  "outgoing_relationships" : "http://localhost:7474/db/data/node/16/relationships/out",
  "data" : {
    "name" : "Mattias"
  },
  "traverse" : "http://localhost:7474/db/data/node/16/traverse/{returnType}",
  "all_typed_relationships" : "http://localhost:7474/db/data/node/16/relationships/all/{-list|&|types}",
  "property" : "http://localhost:7474/db/data/node/16/properties/{key}",
  "self" : "http://localhost:7474/db/data/node/16",
  "outgoing_typed_relationships" : "http://localhost:7474/db/data/node/16/relationships/out/{-list|&|types}",
  "properties" : "http://localhost:7474/db/data/node/16/properties",
  "incoming_relationships" : "http://localhost:7474/db/data/node/16/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://localhost:7474/db/data/node/16/relationships",
  "paged_traverse" : "http://localhost:7474/db/data/node/16/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://localhost:7474/db/data/node/16/relationships/all",
  "incoming_typed_relationships" : "http://localhost:7474/db/data/node/16/relationships/in/{-list|&|types}"
}, {
  "outgoing_relationships" : "http://localhost:7474/db/data/node/15/relationships/out",
  "data" : {
    "name" : "Peter"
  },
  "traverse" : "http://localhost:7474/db/data/node/15/traverse/{returnType}",
  "all_typed_relationships" : "http://localhost:7474/db/data/node/15/relationships/all/{-list|&|types}",
  "property" : "http://localhost:7474/db/data/node/15/properties/{key}",
  "self" : "http://localhost:7474/db/data/node/15",
  "outgoing_typed_relationships" : "http://localhost:7474/db/data/node/15/relationships/out/{-list|&|types}",
  "properties" : "http://localhost:7474/db/data/node/15/properties",
  "incoming_relationships" : "http://localhost:7474/db/data/node/15/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://localhost:7474/db/data/node/15/relationships",
  "paged_traverse" : "http://localhost:7474/db/data/node/15/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://localhost:7474/db/data/node/15/relationships/all",
  "incoming_typed_relationships" : "http://localhost:7474/db/data/node/15/relationships/in/{-list|&|types}"
}, {
  "outgoing_relationships" : "http://localhost:7474/db/data/node/14/relationships/out",
  "data" : {
    "name" : "Tobias"
  },
  "traverse" : "http://localhost:7474/db/data/node/14/traverse/{returnType}",
  "all_typed_relationships" : "http://localhost:7474/db/data/node/14/relationships/all/{-list|&|types}",
  "property" : "http://localhost:7474/db/data/node/14/properties/{key}",
  "self" : "http://localhost:7474/db/data/node/14",
  "outgoing_typed_relationships" : "http://localhost:7474/db/data/node/14/relationships/out/{-list|&|types}",
  "properties" : "http://localhost:7474/db/data/node/14/properties",
  "incoming_relationships" : "http://localhost:7474/db/data/node/14/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://localhost:7474/db/data/node/14/relationships",
  "paged_traverse" : "http://localhost:7474/db/data/node/14/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://localhost:7474/db/data/node/14/relationships/all",
  "incoming_typed_relationships" : "http://localhost:7474/db/data/node/14/relationships/in/{-list|&|types}"
} ]
```

6.10.2. Return relationships from a traversal

Final Graph:


{ Node[4]|'name' = 'I' : String }
own
know
{ Node[2]|'name' = 'car' : String }
{ Node[3]|'name' = 'you' : String }

Example request

- **POST** `http://localhost:7474/db/data/node/4/traverse/relationship`
- **Accept:** `application/json`
- **Content-Type:** `application/json`

```
{"order":"breadth_first","uniqueness":"none","return_filter":{"language":"builtin","name":"all"}}
```

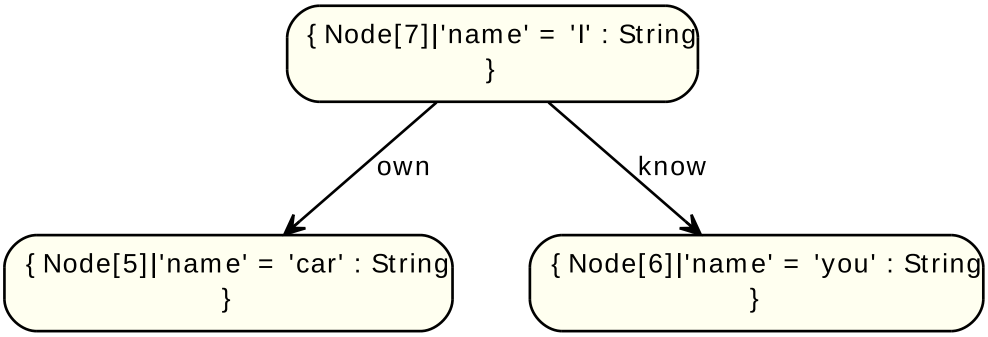
Example response

- **200:** `OK`
- **Content-Type:** `application/json`

```
[ {
  "start" : "http://localhost:7474/db/data/node/4",
  "data" : {
  },
  "self" : "http://localhost:7474/db/data/relationship/0",
  "property" : "http://localhost:7474/db/data/relationship/0/properties/{key}",
  "properties" : "http://localhost:7474/db/data/relationship/0/properties",
  "type" : "know",
  "extensions" : {
  },
  "end" : "http://localhost:7474/db/data/node/3"
}, {
  "start" : "http://localhost:7474/db/data/node/4",
  "data" : {
  },
  "self" : "http://localhost:7474/db/data/relationship/1",
  "property" : "http://localhost:7474/db/data/relationship/1/properties/{key}",
  "properties" : "http://localhost:7474/db/data/relationship/1/properties",
  "type" : "own",
  "extensions" : {
  },
  "end" : "http://localhost:7474/db/data/node/2"
} ]
```

6.10.3. Return paths from a traversal

Final Graph:


{ Node[7]|'name' = 'I' : String }
own
know
{ Node[5]|'name' = 'car' : String }
{ Node[6]|'name' = 'you' : String }

Example request

- **POST** `http://localhost:7474/db/data/node/7/traverse/path`
- **Accept:** `application/json`
- **Content-Type:** `application/json`

```
{"order":"breadth_first","uniqueness":"none","return_filter":{"language":"builtin","name":"all"}}
```

Example response

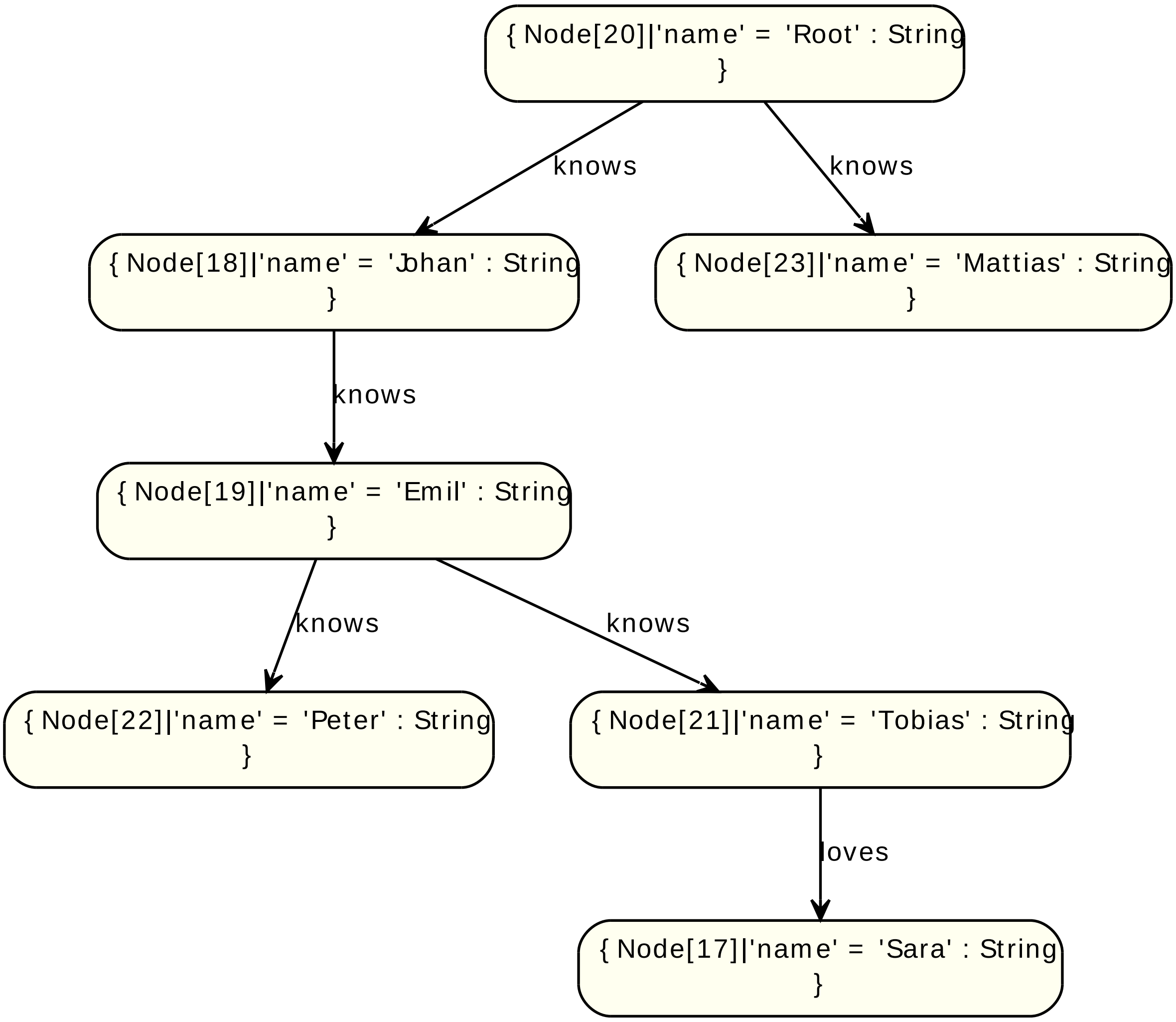
- **200:** `OK`
- **Content-Type:** `application/json`

```
[ {
  "start" : "http://localhost:7474/db/data/node/7",
  "nodes" : [ "http://localhost:7474/db/data/node/7" ],
  "length" : 0,
  "relationships" : [ ],
  "end" : "http://localhost:7474/db/data/node/7"
}, {
  "start" : "http://localhost:7474/db/data/node/7",
  "nodes" : [ "http://localhost:7474/db/data/node/7", "http://localhost:7474/db/data/node/6" ],
  "length" : 1,
  "relationships" : [ "http://localhost:7474/db/data/relationship/2" ],
  "end" : "http://localhost:7474/db/data/node/6"
}, {
  "start" : "http://localhost:7474/db/data/node/7",
  "nodes" : [ "http://localhost:7474/db/data/node/7", "http://localhost:7474/db/data/node/5" ],
  "length" : 1,
  "relationships" : [ "http://localhost:7474/db/data/relationship/3" ],
  "end" : "http://localhost:7474/db/data/node/5"
} ]
```

6.10.4. Traversal returning nodes below a certain depth

Here, all nodes at a traversal depth below 3 are returned.

Final Graph:


{ Node[20]|'name' = 'Root' : String }
knows
knows
{ Node[18]|'name' = 'Johan' : String }
{ Node[23]|'name' = 'Mattias' : String }
knows
{ Node[19]|'name' = 'Emil' : String }
knows
knows
{ Node[22]|'name' = 'Peter' : String }
{ Node[21]|'name' = 'Tobias' : String }
loves
{ Node[17]|'name' = 'Sara' : String }

Example request

- **`POST`** `http://localhost:7474/db/data/node/20/traverse/node`
- **`Accept:`** `application/json`

- **Content-Type:** `application/json`

```
{
  "return_filter" : {
    "body" : "position.length()<3;",
    "language" : "javascript"
  },
  "prune_evaluator" : {
    "name" : "none",
    "language" : "builtin"
  }
}
```

Example response

- **200:** `OK`
- **Content-Type:** `application/json`

```
[ {
  "outgoing_relationships" : "http://localhost:7474/db/data/node/20/relationships/out",
  "data" : {
    "name" : "Root"
  },
  "traverse" : "http://localhost:7474/db/data/node/20/traverse/{returnType}",
  "all_typed_relationships" : "http://localhost:7474/db/data/node/20/relationships/all/{-list|&|types}",
  "property" : "http://localhost:7474/db/data/node/20/properties/{key}",
  "self" : "http://localhost:7474/db/data/node/20",
  "outgoing_typed_relationships" : "http://localhost:7474/db/data/node/20/relationships/out/{-list|&|types}",
  "properties" : "http://localhost:7474/db/data/node/20/properties",
  "incoming_relationships" : "http://localhost:7474/db/data/node/20/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://localhost:7474/db/data/node/20/relationships",
  "paged_traverse" : "http://localhost:7474/db/data/node/20/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://localhost:7474/db/data/node/20/relationships/all",
  "incoming_typed_relationships" : "http://localhost:7474/db/data/node/20/relationships/in/{-list|&|types}"
}, {
  "outgoing_relationships" : "http://localhost:7474/db/data/node/23/relationships/out",
  "data" : {
    "name" : "Mattias"
  },
  "traverse" : "http://localhost:7474/db/data/node/23/traverse/{returnType}",
  "all_typed_relationships" : "http://localhost:7474/db/data/node/23/relationships/all/{-list|&|types}",
  "property" : "http://localhost:7474/db/data/node/23/properties/{key}",
  "self" : "http://localhost:7474/db/data/node/23",
  "outgoing_typed_relationships" : "http://localhost:7474/db/data/node/23/relationships/out/{-list|&|types}",
  "properties" : "http://localhost:7474/db/data/node/23/properties",
  "incoming_relationships" : "http://localhost:7474/db/data/node/23/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://localhost:7474/db/data/node/23/relationships",
  "paged_traverse" : "http://localhost:7474/db/data/node/23/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://localhost:7474/db/data/node/23/relationships/all",
  "incoming_typed_relationships" : "http://localhost:7474/db/data/node/23/relationships/in/{-list|&|types}"
}, {
  "outgoing_relationships" : "http://localhost:7474/db/data/node/18/relationships/out",
  "data" : {
    "name" : "Johan"
  },
  "traverse" : "http://localhost:7474/db/data/node/18/traverse/{returnType}",
  "all_typed_relationships" : "http://localhost:7474/db/data/node/18/relationships/all/{-list|&|types}",
  "property" : "http://localhost:7474/db/data/node/18/properties/{key}",
  "self" : "http://localhost:7474/db/data/node/18",
  "outgoing_typed_relationships" : "http://localhost:7474/db/data/node/18/relationships/out/{-list|&|types}",
  "properties" : "http://localhost:7474/db/data/node/18/properties",
  "incoming_relationships" : "http://localhost:7474/db/data/node/18/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://localhost:7474/db/data/node/18/relationships",
  "paged_traverse" : "http://localhost:7474/db/data/node/18/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://localhost:7474/db/data/node/18/relationships/all",
  "incoming_typed_relationships" : "http://localhost:7474/db/data/node/18/relationships/in/{-list|&|types}"
}, {
  "outgoing_relationships" : "http://localhost:7474/db/data/node/19/relationships/out",
  "data" : {
    "name" : "Emil"
  },
  "traverse" : "http://localhost:7474/db/data/node/19/traverse/{returnType}",
  "all_typed_relationships" : "http://localhost:7474/db/data/node/19/relationships/all/{-list|&|types}",
  "property" : "http://localhost:7474/db/data/node/19/properties/{key}",
  "self" : "http://localhost:7474/db/data/node/19",
  "outgoing_typed_relationships" : "http://localhost:7474/db/data/node/19/relationships/out/{-list|&|types}",
  "properties" : "http://localhost:7474/db/data/node/19/properties",
  "incoming_relationships" : "http://localhost:7474/db/data/node/19/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://localhost:7474/db/data/node/19/relationships",
  "paged_traverse" : "http://localhost:7474/db/data/node/19/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://localhost:7474/db/data/node/19/relationships/all",
  "incoming_typed_relationships" : "http://localhost:7474/db/data/node/19/relationships/in/{-list|&|types}"
} ]
```

6.10.5. Creating a paged traverser

Paged traversers are created by `POST`-ing a traversal description to the link identified by the `paged_traverser` key in a node representation. When creating a paged traverser, the same options apply as for a regular traverser, meaning that `node`, `path`, or `fullpath`, can be targeted.

Example request

- **POST** `http://0.0.0.0:7474/db/data/node/34/paged/traverse/node`
- **Accept:** `application/json`
- **Content-Type:** `application/json`

```
{
  "prune_evaluator":{
    "language":"builtin",
    "name":"none"
  },
  "return_filter":{
    "language":"javascript",
    "body":"position.endNode().getProperty('name').contains('1');"
  },
  "order":"depth_first",
  "relationships":{
    "type":"NEXT",
    "direction":"out"
  }
}
```

Example response

- **201:** `Created`
- **Content-Type:** `application/json`

- **Location:** `http://0.0.0.0:7474/db/data/node/34/paged/traverse/node/1351f8afe6a54321be4d38a4e0815695`

```
[ {
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/35/relationships/out",
  "data" : {
    "name" : "1"
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/35/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/35/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/35/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/35",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/35/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/35/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/35/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/35/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/35/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/35/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/35/relationships/in/{-list|&|types}"
}, {
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/44/relationships/out",
  "data" : {
    "name" : "10"
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/44/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/44/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/44/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/44",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/44/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/44/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/44/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/44/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/44/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/44/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/44/relationships/in/{-list|&|types}"
}, {
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/45/relationships/out",
  "data" : {
    "name" : "11"
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/45/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/45/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/45/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/45",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/45/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/45/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/45/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/45/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/45/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/45/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/45/relationships/in/{-list|&|types}"
}, {
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/46/relationships/out",
  "data" : {
    "name" : "12"
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/46/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/46/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/46/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/46",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/46/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/46/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/46/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/46/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/46/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/46/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/46/relationships/in/{-list|&|types}"
}, {
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/47/relationships/out",
  "data" : {
    "name" : "13"
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/47/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/47/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/47/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/47",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/47/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/47/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/47/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/47/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/47/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/47/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/47/relationships/in/{-list|&|types}"
}, {
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/48/relationships/out",
  "data" : {
    "name" : "14"
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/48/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/48/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/48/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/48",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/48/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/48/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/48/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/48/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/48/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/48/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/48/relationships/in/{-list|&|types}"
}, {
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/49/relationships/out",
  "data" : {
    "name" : "15"
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/49/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/49/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/49/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/49",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/49/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/49/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/49/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/49/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/49/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/49/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/49/relationships/in/{-list|&|types}"
```

```
}, {
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/50/relationships/out",
  "data" : {
    "name" : "16"
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/50/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/50/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/50/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/50",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/50/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/50/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/50/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/50/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/50/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/50/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/50/relationships/in/{-list|&|types}"
}, {
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/51/relationships/out",
  "data" : {
    "name" : "17"
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/51/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/51/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/51/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/51",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/51/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/51/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/51/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/51/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/51/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/51/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/51/relationships/in/{-list|&|types}"
}, {
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/52/relationships/out",
  "data" : {
    "name" : "18"
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/52/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/52/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/52/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/52",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/52/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/52/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/52/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/52/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/52/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/52/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/52/relationships/in/{-list|&|types}"
}, {
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/53/relationships/out",
  "data" : {
    "name" : "19"
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/53/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/53/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/53/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/53",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/53/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/53/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/53/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/53/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/53/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/53/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/53/relationships/in/{-list|&|types}"
}, {
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/55/relationships/out",
  "data" : {
    "name" : "21"
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/55/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/55/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/55/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/55",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/55/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/55/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/55/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/55/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/55/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/55/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/55/relationships/in/{-list|&|types}"
}, {
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/65/relationships/out",
  "data" : {
    "name" : "31"
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/65/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/65/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/65/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/65",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/65/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/65/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/65/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/65/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/65/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/65/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/65/relationships/in/{-list|&|types}"
} ]
```

6.10.6. Paging through the results of a paged traverser

Paged traversers hold state on the server, and allow clients to page through the results of a traversal. To progress to the next page of traversal results, the client issues a HTTP GET request on the paged traversal URI which causes the traversal to fill the next page (or partially fill it if insufficient results are available).

Note that if a traverser expires through inactivity it will cause a 404 response on the next `GET` request. Traversers' leases are renewed on every successful access for the same amount of time as originally specified.

When the paged traverser reaches the end of its results, the client can expect a 404 response as the traverser is disposed by the server.

Example request

- **`GET`** `http://0.0.0.0:7474/db/data/node/67/paged/traverse/node/0b7d634201e047c2b9321fe90702c3cb`

- **Accept:** `application/json`

Example response

- **200:** `OK`
- **Content-Type:** `application/json`

```
[ {
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/398/relationships/out",
  "data" : {
    "name" : "331"
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/398/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/398/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/398/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/398",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/398/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/398/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/398/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/398/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/398/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/398/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/398/relationships/in/{-list|&|types}"
}, {
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/408/relationships/out",
  "data" : {
    "name" : "341"
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/408/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/408/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/408/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/408",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/408/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/408/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/408/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/408/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/408/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/408/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/408/relationships/in/{-list|&|types}"
}, {
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/418/relationships/out",
  "data" : {
    "name" : "351"
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/418/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/418/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/418/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/418",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/418/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/418/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/418/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/418/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/418/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/418/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/418/relationships/in/{-list|&|types}"
}, {
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/428/relationships/out",
  "data" : {
    "name" : "361"
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/428/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/428/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/428/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/428",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/428/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/428/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/428/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/428/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/428/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/428/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/428/relationships/in/{-list|&|types}"
}, {
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/438/relationships/out",
  "data" : {
    "name" : "371"
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/438/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/438/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/438/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/438",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/438/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/438/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/438/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/438/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/438/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/438/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/438/relationships/in/{-list|&|types}"
}, {
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/448/relationships/out",
  "data" : {
    "name" : "381"
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/448/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/448/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/448/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/448",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/448/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/448/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/448/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/448/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/448/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/448/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/448/relationships/in/{-list|&|types}"
}, {
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/458/relationships/out",
  "data" : {
    "name" : "391"
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/458/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/458/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/458/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/458",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/458/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/458/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/458/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/458/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/458/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/458/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/458/relationships/in/{-list|&|types}"
}, {
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/468/relationships/out",
  "data" : {
    "name" : "401"
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/468/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/468/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/468/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/468",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/468/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/468/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/468/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/468/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/468/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/468/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/468/relationships/in/{-list|&|types}"
}, {
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/477/relationships/out",
  "data" : {
    "name" : "410"
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/477/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/477/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/477/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/477",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/477/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/477/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/477/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/477/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/477/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/477/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/477/relationships/in/{-list|&|types}"
}, {
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/478/relationships/out",
  "data" : {
    "name" : "411"
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/478/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/478/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/478/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/478",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/478/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/478/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/478/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/478/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/478/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/478/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/478/relationships/in/{-list|&|types}"
}, {
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/479/relationships/out",
  "data" : {
    "name" : "412"
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/479/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/479/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/479/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/479",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/479/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/479/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/479/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/479/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/479/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/479/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/479/relationships/in/{-list|&|types}"
}, {
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/480/relationships/out",
  "data" : {
    "name" : "413"
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/480/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/480/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/480/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/480",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/480/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/480/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/480/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/480/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/480/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/480/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/480/relationships/in/{-list|&|types}"
}, {
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/481/relationships/out",
  "data" : {
    "name" : "414"
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/481/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/481/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/481/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/481",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/481/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/481/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/481/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/481/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/481/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/481/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/481/relationships/in/{-list|&|types}"
}, {
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/482/relationships/out",
  "data" : {
    "name" : "415"
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/482/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/482/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/482/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/482",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/482/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/482/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/482/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/482/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/482/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/482/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/482/relationships/in/{-list|&|types}"
```

```
}, {
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/483/relationships/out",
  "data" : {
    "name" : "416"
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/483/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/483/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/483/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/483",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/483/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/483/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/483/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/483/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/483/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/483/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/483/relationships/in/{-list|&|types}"
}, {
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/484/relationships/out",
  "data" : {
    "name" : "417"
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/484/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/484/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/484/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/484",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/484/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/484/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/484/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/484/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/484/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/484/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/484/relationships/in/{-list|&|types}"
}, {
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/485/relationships/out",
  "data" : {
    "name" : "418"
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/485/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/485/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/485/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/485",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/485/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/485/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/485/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/485/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/485/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/485/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/485/relationships/in/{-list|&|types}"
}, {
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/486/relationships/out",
  "data" : {
    "name" : "419"
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/486/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/486/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/486/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/486",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/486/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/486/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/486/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/486/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/486/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/486/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/486/relationships/in/{-list|&|types}"
}, {
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/488/relationships/out",
  "data" : {
    "name" : "421"
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/488/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/488/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/488/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/488",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/488/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/488/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/488/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/488/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/488/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/488/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/488/relationships/in/{-list|&|types}"
}, {
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/498/relationships/out",
  "data" : {
    "name" : "431"
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/498/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/498/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/498/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/498",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/498/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/498/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/498/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/498/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/498/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/498/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/498/relationships/in/{-list|&|types}"
}, {
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/508/relationships/out",
  "data" : {
    "name" : "441"
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/508/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/508/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/508/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/508",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/508/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/508/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/508/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/508/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/508/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/508/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/508/relationships/in/{-list|&|types}"
} ]
```

6.10.7. Paged traverser page size

The default page size is 50 items, but depending on the application larger or smaller pages sizes might be appropriate. This can be set by adding a `pageSize` query parameter.

Example request

- **POST** `http://0.0.0.0:7474/db/data/node/544/paged/traverse/node?pageSize=1`
- **Accept:** `application/json`
- **Content-Type:** `application/json`

```
{
  "prune_evaluator":{
    "language":"builtin",
    "name":"none"
  },
  "return_filter":{
    "language":"javascript",
    "body":"position.endNode().getProperty('name').contains('1');"
  },
  "order":"depth_first",
  "relationships":{
    "type":"NEXT",
    "direction":"out"
  }
}
```

Example response

- **201:** `Created`
- **Content-Type:** `application/json`
- **Location:** `http://0.0.0.0:7474/db/data/node/544/paged/traverse/node/46260d71bdfe45878f2d2992ba659596`

```
[ {
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/545/relationships/out",
  "data" : {
    "name" : "1"
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/545/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/545/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/545/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/545",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/545/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/545/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/545/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/545/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/545/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/545/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/545/relationships/in/{-list|&|types}"
} ]
```

6.10.8. Paged traverser timeout

The default timeout for a paged traverser is 60 seconds, but depending on the application larger or smaller timeouts might be appropriate. This can be set by adding a `leaseTime` query parameter with the number of seconds the paged traverser should last.

Example request

- **POST** `http://0.0.0.0:7474/db/data/node/577/paged/traverse/node?leaseTime=10`
- **Accept:** `application/json`
- **Content-Type:** `application/json`

```
{
  "prune_evaluator":{
    "language":"builtin",
    "name":"none"
  },
  "return_filter":{
    "language":"javascript",
    "body":"position.endNode().getProperty('name').contains('1');"
  },
  "order":"depth_first",
  "relationships":{
    "type":"NEXT",
    "direction":"out"
  }
}
```

Example response

- **201:** `Created`
- **Content-Type:** `application/json`
- **Location:** `http://0.0.0.0:7474/db/data/node/577/paged/traverse/node/0c7969f15c234f9f9dc1fd3a27d6ea9e`

```
[ {
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/578/relationships/out",
  "data" : {
    "name" : "1"
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/578/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/578/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/578/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/578",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/578/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/578/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/578/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/578/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/578/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/578/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/578/relationships/in/{-list|&|types}"
}, {
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/587/relationships/out",
  "data" : {
    "name" : "10"
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/587/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/587/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/587/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/587",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/587/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/587/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/587/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/587/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/587/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/587/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/587/relationships/in/{-list|&|types}"
}, {
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/588/relationships/out",
  "data" : {
    "name" : "11"
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/588/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/588/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/588/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/588",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/588/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/588/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/588/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/588/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/588/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/588/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/588/relationships/in/{-list|&|types}"
}, {
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/589/relationships/out",
  "data" : {
    "name" : "12"
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/589/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/589/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/589/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/589",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/589/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/589/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/589/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/589/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/589/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/589/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/589/relationships/in/{-list|&|types}"
}, {
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/590/relationships/out",
  "data" : {
    "name" : "13"
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/590/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/590/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/590/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/590",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/590/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/590/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/590/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/590/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/590/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/590/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/590/relationships/in/{-list|&|types}"
}, {
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/591/relationships/out",
  "data" : {
    "name" : "14"
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/591/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/591/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/591/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/591",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/591/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/591/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/591/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/591/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/591/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/591/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/591/relationships/in/{-list|&|types}"
}, {
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/592/relationships/out",
  "data" : {
    "name" : "15"
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/592/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/592/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/592/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/592",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/592/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/592/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/592/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/592/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/592/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/592/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/592/relationships/in/{-list|&|types}"
}, {
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/593/relationships/out",
  "data" : {
    "name" : "16"
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/593/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/593/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/593/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/593",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/593/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/593/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/593/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/593/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/593/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/593/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/593/relationships/in/{-list|&|types}"
}, {
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/594/relationships/out",
  "data" : {
    "name" : "17"
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/594/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/594/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/594/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/594",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/594/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/594/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/594/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/594/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/594/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/594/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/594/relationships/in/{-list|&|types}"
}, {
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/595/relationships/out",
  "data" : {
    "name" : "18"
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/595/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/595/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/595/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/595",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/595/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/595/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/595/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/595/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/595/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/595/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/595/relationships/in/{-list|&|types}"
}, {
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/596/relationships/out",
  "data" : {
    "name" : "19"
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/596/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/596/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/596/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/596",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/596/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/596/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/596/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/596/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/596/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/596/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/596/relationships/in/{-list|&|types}"
}, {
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/598/relationships/out",
  "data" : {
    "name" : "21"
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/598/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/598/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/598/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/598",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/598/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/598/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/598/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/598/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/598/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/598/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/598/relationships/in/{-list|&|types}"
}, {
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/608/relationships/out",
  "data" : {
    "name" : "31"
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/608/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/608/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/608/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/608",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/608/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/608/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/608/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/608/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/608/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/608/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/608/relationships/in/{-list|&|types}"
} ]
```

6.11. Built-in Graph Algorithms

Neo4j comes with a number of built-in graph algorithms. They are performed from a start node. The traversal is controlled by the URI and the body sent with the request.

algorithm

The algorithm to choose. If not set, default is `shortestPath`. `algorithm` can have one of these values:

- `shortestPath`
- `allSimplePaths`
- `allPaths`
- `dijkstra` (optional with `cost_property` and `default_cost` parameters)

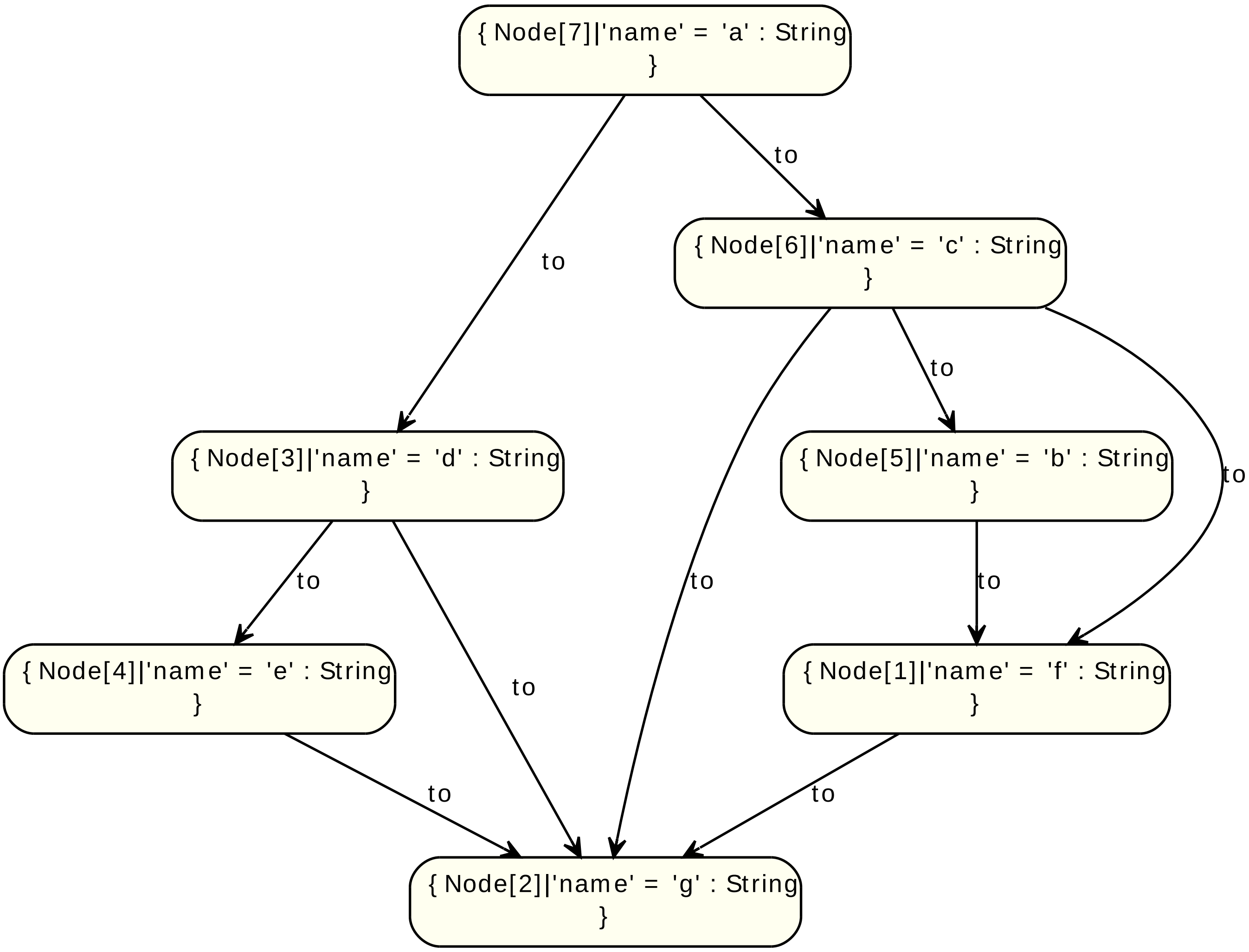
max_depth

The maximum depth as an integer for the algorithms like ShortestPath, where applicable. Default is `1`.

6.11.1. Find all shortest paths

The `shortestPath` algorithm can find multiple paths between the same nodes, like in this example.

Final Graph:


{ Node[7]|'name' = 'a' : String }
{ Node[6]|'name' = 'c' : String }
{ Node[3]|'name' = 'd' : String }
{ Node[5]|'name' = 'b' : String }
{ Node[4]|'name' = 'e' : String }
{ Node[1]|'name' = 'f' : String }
{ Node[2]|'name' = 'g' : String }
to

Example request

- **POST** `http://localhost:7474/db/data/node/7/paths`
- **Accept:** `application/json`
- **Content-Type:** `application/json`

```
{
  "to":"http://localhost:7474/db/data/node/2",
   "max_depth":3,
   "relationships":{
    "type":"to",
     "direction":"out"
  },
   "algorithm":"shortestPath"
}
```

Example response

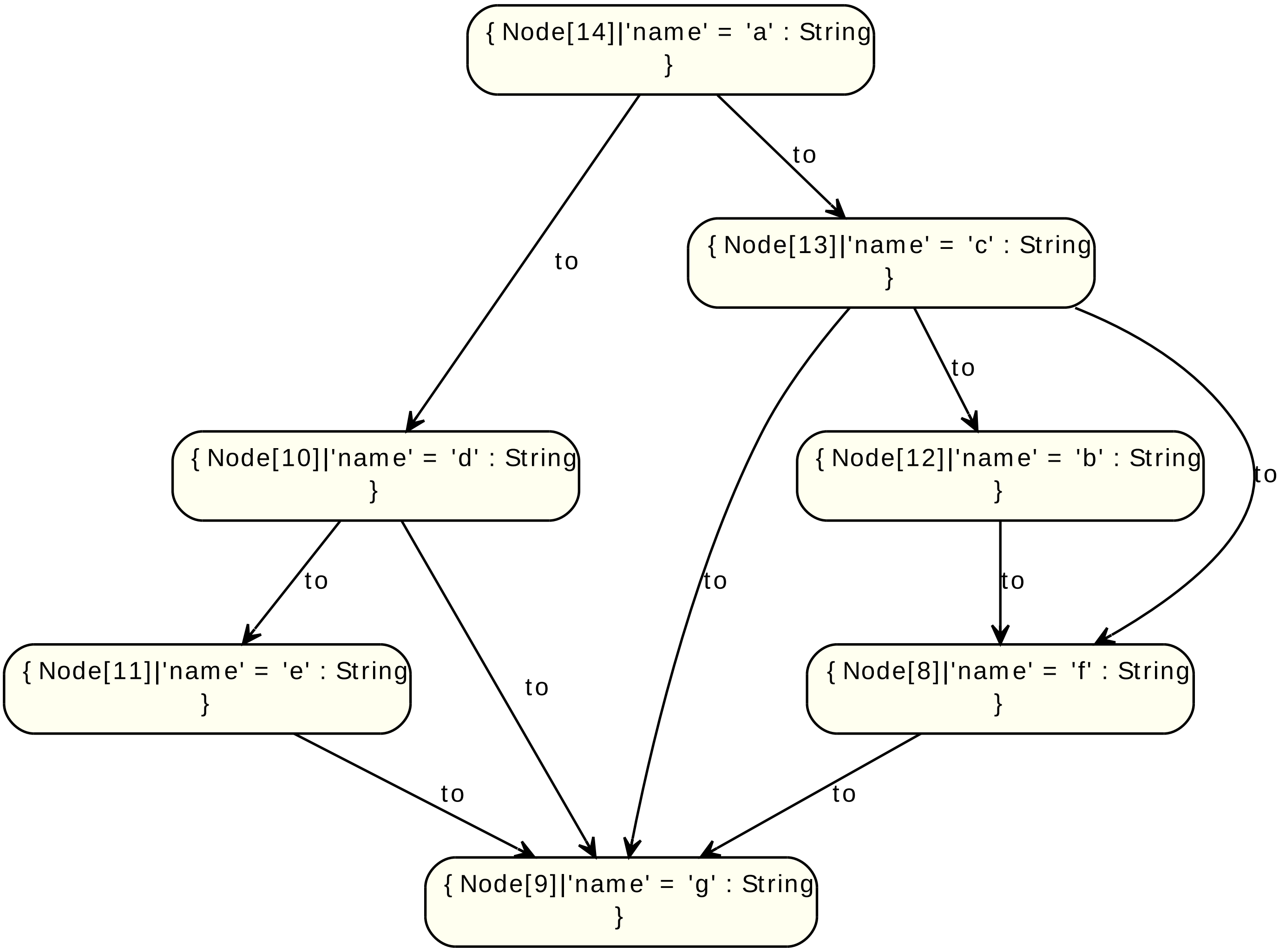
- **200:** `OK`
- **Content-Type:** `application/json`

```
[ {
  "start" : "http://localhost:7474/db/data/node/7",
  "nodes" : [ "http://localhost:7474/db/data/node/7", "http://localhost:7474/db/data/node/3", "http://localhos
  "length" : 2,
  "relationships" : [ "http://localhost:7474/db/data/relationship/1", "http://localhost:7474/db/data/relations
  "end" : "http://localhost:7474/db/data/node/2"
}, {
  "start" : "http://localhost:7474/db/data/node/7",
  "nodes" : [ "http://localhost:7474/db/data/node/7", "http://localhost:7474/db/data/node/6", "http://localhos
  "length" : 2,
  "relationships" : [ "http://localhost:7474/db/data/relationship/0", "http://localhost:7474/db/data/relations
  "end" : "http://localhost:7474/db/data/node/2"
} ]
```

6.11.2. Find one of the shortest paths between nodes

If no path algorithm is specified, a `ShortestPath` algorithm with a max depth of 1 will be chosen. In this example, the `max_depth` is set to `3` in order to find the shortest path between 3 linked nodes.

Final Graph:


{ Node[14]|'name' = 'a' : String }
{ Node[13]|'name' = 'c' : String }
{ Node[10]|'name' = 'd' : String }
{ Node[12]|'name' = 'b' : String }
{ Node[11]|'name' = 'e' : String }
{ Node[8]|'name' = 'f' : String }
{ Node[9]|'name' = 'g' : String }
to

Example request

- **POST** `http://localhost:7474/db/data/node/14/path`
- **Accept:** `application/json`
- **Content-Type:** `application/json`

```
{
  "to":"http://localhost:7474/db/data/node/9",
   "max_depth":3,
   "relationships":{
    "type":"to",
     "direction":"out"
  },
   "algorithm":"shortestPath"
}
```

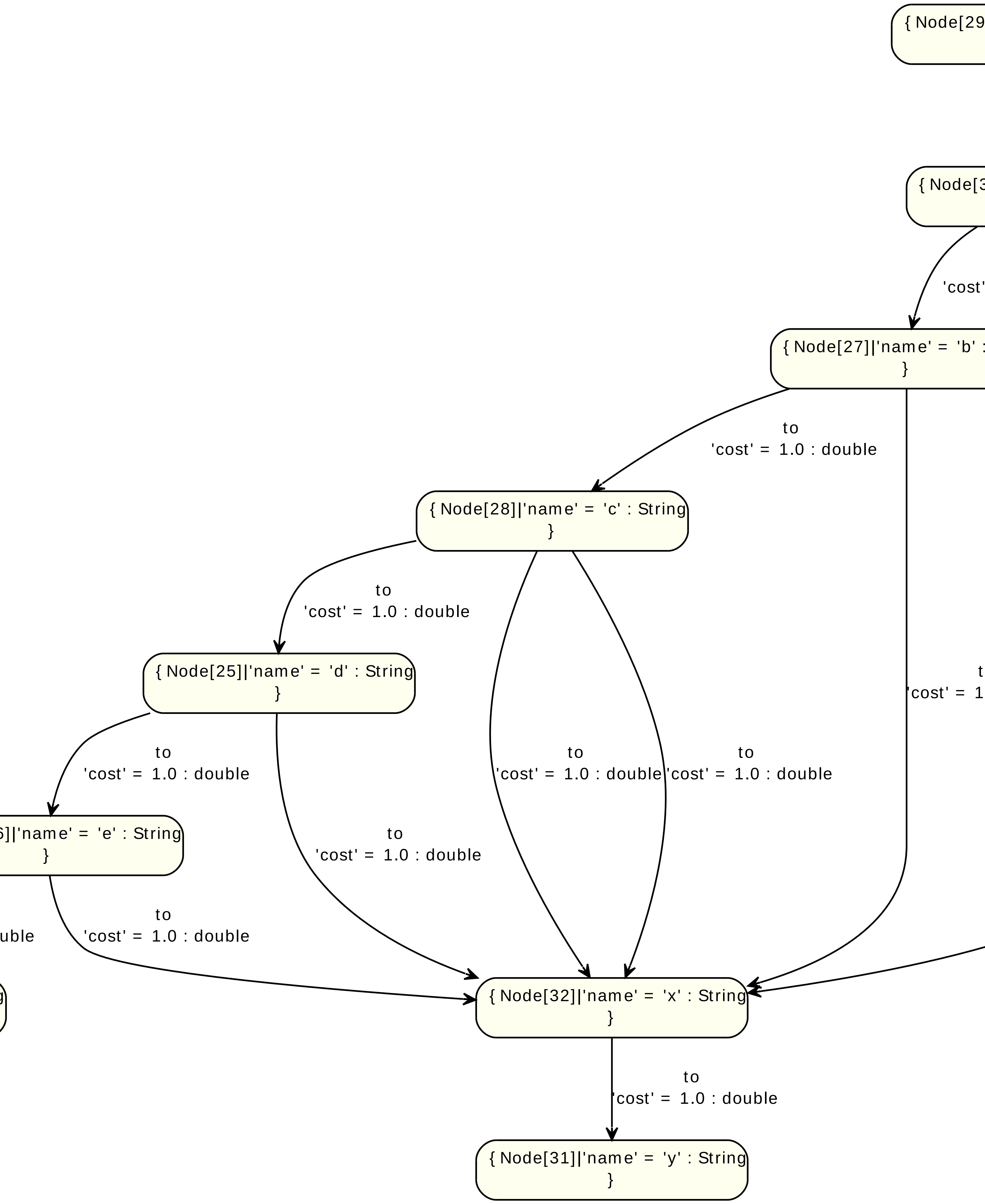
Example response

- **200:** OK
- **Content-Type:** `application/json`

```
{
  "start" : "http://localhost:7474/db/data/node/14",
  "nodes" : [ "http://localhost:7474/db/data/node/14", "http://localhost:7474/db/data/node/10", "http://localh
  "length" : 2,
  "relationships" : [ "http://localhost:7474/db/data/relationship/11", "http://localhost:7474/db/data/relation
  "end" : "http://localhost:7474/db/data/node/9"
}
```

6.11.3. Execute a Dijkstra algorithm with similar weights on relationships

Final Graph:


{ Node[29
{ Node[3
'cost'
{ Node[27]|'name' = 'b' :
}
to
'cost' = 1.0 : double
{ Node[28]|'name' = 'c' : String
}
to
'cost' = 1.0 : double
{ Node[25]|'name' = 'd' : String
}
t
'cost' = 1
to
'cost' = 1.0 : double
to
'cost' = 1.0 : double
to
'cost' = 1.0 : double
6]|'name' = 'e' : String
}
to
'cost' = 1.0 : double
to
'cost' = 1.0 : double
uble
{ Node[32]|'name' = 'x' : String
}
to
'cost' = 1.0 : double
{ Node[31]|'name' = 'y' : String
}

Example request

- **POST** `http://localhost:7474/db/data/node/29/path`
- **Accept:** `application/json`
- **Content-Type:** `application/json`

```
{
  "to":"http://localhost:7474/db/data/node/32",
   "cost_property":"cost",
   "relationships":{
    "type":"to",
     "direction":"out"
  },
   "algorithm":"dijkstra"
}
```

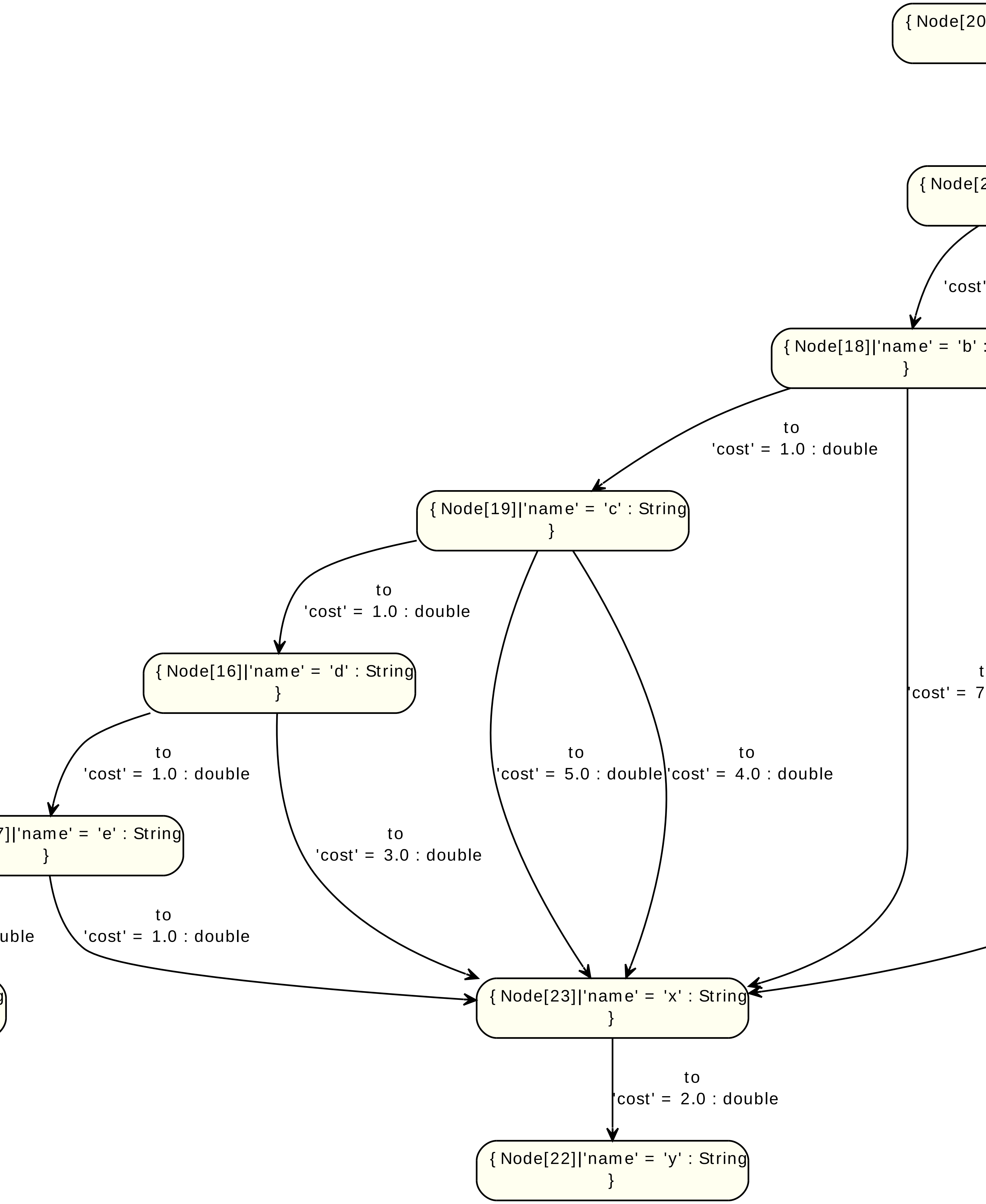
Example response

- **200:** `OK`
- **Content-Type:** `application/json`

```
{
  "weight" : 2.0,
  "start" : "http://localhost:7474/db/data/node/29",
  "nodes" : [ "http://localhost:7474/db/data/node/29", "http://localhost:7474/db/data/node/30", "http://localh
  "length" : 2,
  "relationships" : [ "http://localhost:7474/db/data/relationship/33", "http://localhost:7474/db/data/relation
  "end" : "http://localhost:7474/db/data/node/32"
}
```

6.11.4. Execute a Dijkstra algorithm with weights on relationships

Final Graph:


{ Node[20
{ Node[2
'cost'
{ Node[18]|'name' = 'b' :
}
to
'cost' = 1.0 : double
{ Node[19]|'name' = 'c' : String
}
to
'cost' = 1.0 : double
{ Node[16]|'name' = 'd' : String
}
t
'cost' = 7
to
'cost' = 1.0 : double
to
'cost' = 5.0 : double
to
'cost' = 4.0 : double
7]|'name' = 'e' : String
}
to
'cost' = 3.0 : double
to
'cost' = 1.0 : double
uble
{ Node[23]|'name' = 'x' : String
}
to
'cost' = 2.0 : double
{ Node[22]|'name' = 'y' : String
}

Example request

- **POST** `http://localhost:7474/db/data/node/20/path`
- **Accept:** `application/json`
- **Content-Type:** `application/json`

```
{
  "to":"http://localhost:7474/db/data/node/23",
   "cost_property":"cost",
   "relationships":{
    "type":"to",
     "direction":"out"
  },
   "algorithm":"dijkstra"
}
```

Example response

- **200:** OK
- **Content-Type:** `application/json`

```
{
  "weight" : 6.0,
  "start" : "http://localhost:7474/db/data/node/20",
  "nodes" : [ "http://localhost:7474/db/data/node/20", "http://localhost:7474/db/data/node/21", "http://localh
  "length" : 6,
  "relationships" : [ "http://localhost:7474/db/data/relationship/20", "http://localhost:7474/db/data/relations
  "end" : "http://localhost:7474/db/data/node/23"
}
```

6.12. Batch operations

Caution

Batch support is currently *experimental*. Expect this part of the API to change.

6.12.1. Execute multiple operations in batch

This lets you execute multiple API calls through a single HTTP call, significantly improving performance for large insert and update operations.

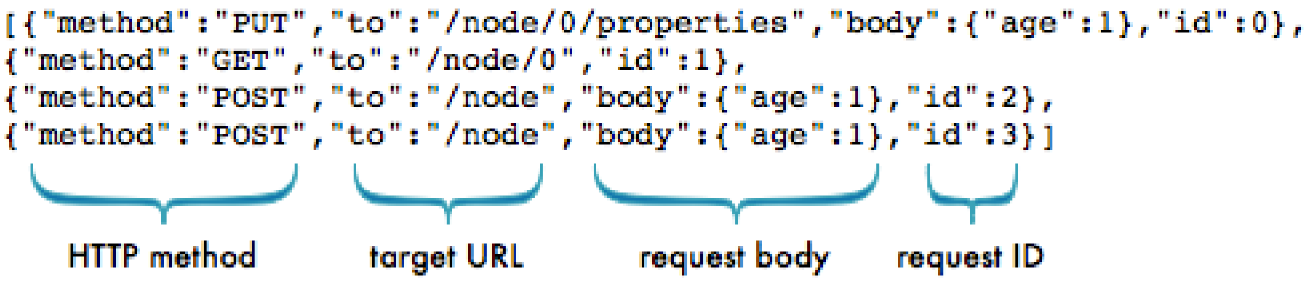
The batch service expects an array of job descriptions as input, each job description describing an action to be performed via the normal server API.

This service is transactional. If any of the operations performed fails (returns a non-2xx HTTP status code), the transaction will be rolled back and all changes will be undone.

Each job description should contain a `path` attribute, with a value relative to the data API root (so http://localhost:7474/db/data/node becomes just /node), and a `method` attribute containing HTTP verb to use.

Optionally you may provide a `body` attribute, and an `id` attribute to help you keep track of responses, although responses are guaranteed to be returned in the same order the job descriptions are received.

The following figure outlines the different parts of the job descriptions:



```
[{"method":"PUT","to":"/node/0/properties","body":{"age":1},"id":0},
{"method":"GET","to":"/node/0","id":1},
{"method":"POST","to":"/node","body":{"age":1},"id":2},
{"method":"POST","to":"/node","body":{"age":1},"id":3}]
```

Example request

- **`POST`** `http://0.0.0.0:7474/db/data/batch`
- **`Accept:`** `application/json`
- **`Content-Type:`** `application/json`

```
[
  {
    "method":"PUT",
    "to":"/node/0/properties",
    "body":{
      "age":1
    },
    "id":0
  },
  {
    "method":"GET",
    "to":"/node/0",
    "id":1
  },
  {
    "method":"POST",
    "to":"/node",
    "body":{
      "age":1
    },
    "id":2
  },
  {
    "method":"POST",
    "to":"/node",
    "body":{
      "age":1
    },
    "id":3
  }
]
```

Example response

- **200:** OK
- **Content-Type:** application/json

```
[{"id":0,"from":"/node/0/properties"},{"id":1,"body":{
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/0/relationships/out",
  "data" : {
    "age" : 1
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/0/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/0/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/0/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/0",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/0/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/0/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/0/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/0/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/0/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/0/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/0/relationships/in/{-list|&|types}"
},"from":"/node/0"},{"id":2,"location":"http://0.0.0.0:7474/db/data/node/3","body":{
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/3/relationships/out",
  "data" : {
    "age" : 1
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/3/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/3/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/3/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/3",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/3/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/3/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/3/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/3/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/3/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/3/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/3/relationships/in/{-list|&|types}"
},"from":"/node"},{"id":3,"location":"http://0.0.0.0:7474/db/data/node/4","body":{
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/4/relationships/out",
  "data" : {
    "age" : 1
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/4/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/4/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/4/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/4",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/4/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/4/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/4/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/4/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/4/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/4/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/4/relationships/in/{-list|&|types}"
},"from":"/node"}]
```

6.12.2. Refer to items created earlier in the same batch job

The batch operation API allows you to refer to the URI returned from a created resource in subsequent job descriptions, within the same batch call.

Use the `{[JOB ID]}` special syntax to inject URIs from created resources into JSON strings in subsequent job descriptions.

Example request

- **POST** `http://0.0.0.0:7474/db/data/batch`
- **Accept:** `application/json`
- **Content-Type:** `application/json`

```
[
  {
    "method":"POST",
    "to":"/node",
    "id":0,
    "body":{
      "age":1
    }
  },
  {
    "method":"POST",
    "to":"/node",
    "id":1,
    "body":{
      "age":12
    }
  },
  {
    "method":"POST",
    "to":"{0}/relationships",
    "id":3,
    "body":{
      "to":"{1}",
      "data":{
        "name":"bob"
      },
      "type":"KNOWS"
    }
  },
  {
    "method":"POST",
    "to":"/index/relationship/my_rels/name/bob",
    "id":4,
    "body":"{3}"
  }
]
```

Example response

- **200:** OK
- **Content-Type:** application/json

```
[{"id":0,"location":"http://0.0.0.0:7474/db/data/node/7","body":{
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/7/relationships/out",
  "data" : {
    "age" : 1
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/7/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/7/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/7/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/7",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/7/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/7/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/7/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/7/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/7/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/7/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/7/relationships/in/{-list|&|types}"
},"from":"/node"},{"id":1,"location":"http://0.0.0.0:7474/db/data/node/8","body":{
  "outgoing_relationships" : "http://0.0.0.0:7474/db/data/node/8/relationships/out",
  "data" : {
    "age" : 12
  },
  "traverse" : "http://0.0.0.0:7474/db/data/node/8/traverse/{returnType}",
  "all_typed_relationships" : "http://0.0.0.0:7474/db/data/node/8/relationships/all/{-list|&|types}",
  "property" : "http://0.0.0.0:7474/db/data/node/8/properties/{key}",
  "self" : "http://0.0.0.0:7474/db/data/node/8",
  "outgoing_typed_relationships" : "http://0.0.0.0:7474/db/data/node/8/relationships/out/{-list|&|types}",
  "properties" : "http://0.0.0.0:7474/db/data/node/8/properties",
  "incoming_relationships" : "http://0.0.0.0:7474/db/data/node/8/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://0.0.0.0:7474/db/data/node/8/relationships",
  "paged_traverse" : "http://0.0.0.0:7474/db/data/node/8/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://0.0.0.0:7474/db/data/node/8/relationships/all",
  "incoming_typed_relationships" : "http://0.0.0.0:7474/db/data/node/8/relationships/in/{-list|&|types}"
},"from":"/node"},{"id":3,"location":"http://0.0.0.0:7474/db/data/relationship/1","body":{
  "start" : "http://0.0.0.0:7474/db/data/node/7",
  "data" : {
    "name" : "bob"
  },
  "self" : "http://0.0.0.0:7474/db/data/relationship/1",
  "property" : "http://0.0.0.0:7474/db/data/relationship/1/properties/{key}",
  "properties" : "http://0.0.0.0:7474/db/data/relationship/1/properties",
  "type" : "KNOWS",
  "extensions" : {
  },
  "end" : "http://0.0.0.0:7474/db/data/node/8"
},"from":"http://0.0.0.0:7474/db/data/node/7/relationships"},{"id":4,"location":"http://0.0.0.0:7474/db/data/i
  "indexed" : "http://0.0.0.0:7474/db/data/index/relationship/my_rels/name/bob/1",
  "start" : "http://0.0.0.0:7474/db/data/node/7",
  "data" : {
    "name" : "bob"
  },
  "self" : "http://0.0.0.0:7474/db/data/relationship/1",
  "property" : "http://0.0.0.0:7474/db/data/relationship/1/properties/{key}",
  "properties" : "http://0.0.0.0:7474/db/data/relationship/1/properties",
  "type" : "KNOWS",
  "extensions" : {
  },
  "end" : "http://0.0.0.0:7474/db/data/node/8"
},"from":"/index/relationship/my_rels/name/bob"}]
```

6.13. Gremlin Plugin

Gremlin <http://gremlin.tinkerpop.com> is a Groovy based Graph Traversal Language. It provides a very expressive way of explicitly scripting traversals through a Neo4j graph.

The Neo4j Gremlin Plugin provides an endpoint to send Gremlin scripts to the Neo4j Server. The scripts are executed on the server database and the results are returned as Neo4j Node and Relationship representations. This keeps the types throughout the REST API consistent. The results are quite verbose when returning Neo4j `Node`, `Relationship` or `Graph` representations. On the other hand, just return properties like in the Section 6.13.4, "Send a Gremlin Script - JSON encoded with table results" example for responses tailored to specific needs.

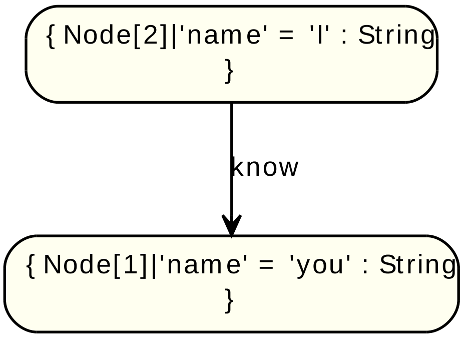
6.13.1. Send a Gremlin Script - URL encoded

Scripts can be sent as URL-encoded

Raw script source

```
i = g.v(2)
i.out
```

Final Graph:


{ Node[2]|'name' = 'I' : String }
know
{ Node[1]|'name' = 'you' : String }

Example request

- **`POST`** `http://localhost:7474/db/data/ext/GremlinPlugin/graphdb/execute_script`
- **`Accept:`** `application/json`
- **`Content-Type:`** `application/x-www-form-urlencoded`

```
script=i+%3D+g.v%282%29%3Bi.out
```

Example response

- **`200:`** `OK`
- **`Content-Type:`** `application/json`

```
[ {
  "outgoing_relationships" : "http://localhost:7474/db/data/node/1/relationships/out",
  "data" : {
    "name" : "you"
  },
  "traverse" : "http://localhost:7474/db/data/node/1/traverse/{returnType}",
  "all_typed_relationships" : "http://localhost:7474/db/data/node/1/relationships/all/{-list|&|types}",
  "property" : "http://localhost:7474/db/data/node/1/properties/{key}",
  "self" : "http://localhost:7474/db/data/node/1",
  "properties" : "http://localhost:7474/db/data/node/1/properties",
  "outgoing_typed_relationships" : "http://localhost:7474/db/data/node/1/relationships/out/{-list|&|types}",
  "incoming_relationships" : "http://localhost:7474/db/data/node/1/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://localhost:7474/db/data/node/1/relationships",
  "paged_traverse" : "http://localhost:7474/db/data/node/1/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://localhost:7474/db/data/node/1/relationships/all",
  "incoming_typed_relationships" : "http://localhost:7474/db/data/node/1/relationships/in/{-list|&|types}"
} ]
```

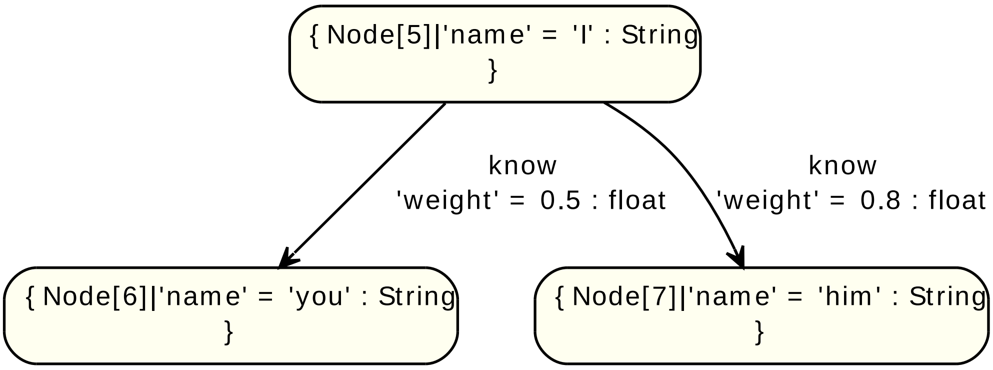
6.13.2. Load a sample graph

Import a graph form a GraphML <http://graphml.graphdrawing.org/> file can be achieved through the Gremlin GraphMLReader. The following script imports a small GraphML file from an URL into Neo4j, resulting in the depicted graph. It then returns a list of all nodes in the graph.

Raw script source

```
g.loadGraphML('https://raw.github.com/neo4j/gremlin-plugin/master/src/data/graphml1.xml')
g.V
```

Final Graph:


{ Node[5]|'name' = 'I' : String }
know
'weight' = 0.5 : float
know
'weight' = 0.8 : float
{ Node[6]|'name' = 'you' : String }
{ Node[7]|'name' = 'him' : String }

Example request

- **POST** `http://localhost:7474/db/data/ext/GremlinPlugin/graphdb/execute_script`
- **Accept:** `application/json`
- **Content-Type:** `application/json`

```
{
  "script":"g.loadGraphML('https://raw.github.com/neo4j/gremlin-plugin/master/src/data/graphml1.xml');g.V;"
}
```

Example response

- **200:** `OK`
- **Content-Type:** `application/json`

```
[ {
  "outgoing_relationships" : "http://localhost:7474/db/data/node/5/relationships/out",
  "data" : {
    "name" : "I"
  },
  "traverse" : "http://localhost:7474/db/data/node/5/traverse/{returnType}",
  "all_typed_relationships" : "http://localhost:7474/db/data/node/5/relationships/all/{-list|&|types}",
  "property" : "http://localhost:7474/db/data/node/5/properties/{key}",
  "self" : "http://localhost:7474/db/data/node/5",
  "properties" : "http://localhost:7474/db/data/node/5/properties",
  "outgoing_typed_relationships" : "http://localhost:7474/db/data/node/5/relationships/out/{-list|&|types}",
  "incoming_relationships" : "http://localhost:7474/db/data/node/5/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://localhost:7474/db/data/node/5/relationships",
  "paged_traverse" : "http://localhost:7474/db/data/node/5/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://localhost:7474/db/data/node/5/relationships/all",
  "incoming_typed_relationships" : "http://localhost:7474/db/data/node/5/relationships/in/{-list|&|types}"
}, {
  "outgoing_relationships" : "http://localhost:7474/db/data/node/6/relationships/out",
  "data" : {
    "name" : "you"
  },
  "traverse" : "http://localhost:7474/db/data/node/6/traverse/{returnType}",
  "all_typed_relationships" : "http://localhost:7474/db/data/node/6/relationships/all/{-list|&|types}",
  "property" : "http://localhost:7474/db/data/node/6/properties/{key}",
  "self" : "http://localhost:7474/db/data/node/6",
  "properties" : "http://localhost:7474/db/data/node/6/properties",
  "outgoing_typed_relationships" : "http://localhost:7474/db/data/node/6/relationships/out/{-list|&|types}",
  "incoming_relationships" : "http://localhost:7474/db/data/node/6/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://localhost:7474/db/data/node/6/relationships",
  "paged_traverse" : "http://localhost:7474/db/data/node/6/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://localhost:7474/db/data/node/6/relationships/all",
  "incoming_typed_relationships" : "http://localhost:7474/db/data/node/6/relationships/in/{-list|&|types}"
}, {
  "outgoing_relationships" : "http://localhost:7474/db/data/node/7/relationships/out",
  "data" : {
    "name" : "him"
  },
  "traverse" : "http://localhost:7474/db/data/node/7/traverse/{returnType}",
  "all_typed_relationships" : "http://localhost:7474/db/data/node/7/relationships/all/{-list|&|types}",
  "property" : "http://localhost:7474/db/data/node/7/properties/{key}",
  "self" : "http://localhost:7474/db/data/node/7",
  "properties" : "http://localhost:7474/db/data/node/7/properties",
  "outgoing_typed_relationships" : "http://localhost:7474/db/data/node/7/relationships/out/{-list|&|types}",
  "incoming_relationships" : "http://localhost:7474/db/data/node/7/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://localhost:7474/db/data/node/7/relationships",
  "paged_traverse" : "http://localhost:7474/db/data/node/7/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://localhost:7474/db/data/node/7/relationships/all",
  "incoming_typed_relationships" : "http://localhost:7474/db/data/node/7/relationships/in/{-list|&|types}"
} ]
```

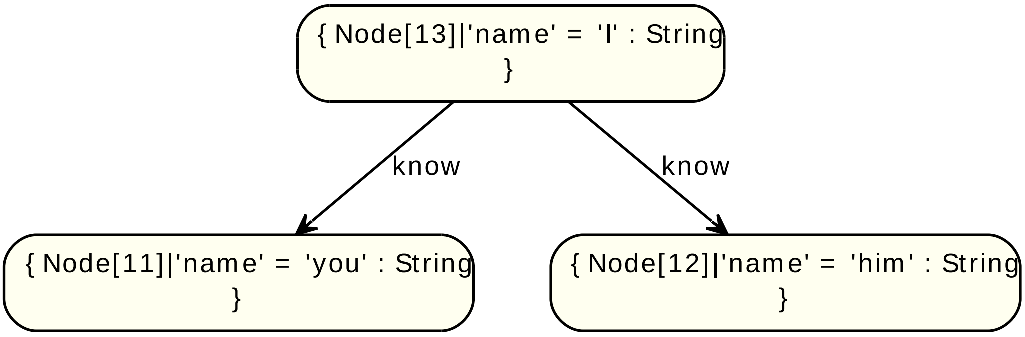
6.13.3. Sort a result using raw Groovy operations

The following script returns a sorted list of all nodes connected via outgoing relationships to node 1, sorted by their `name`-property.

Raw script source

```
g.v(13).out.sort{it.name}.toList()
```

Final Graph:


{ Node[13]|'name' = 'I' : String }
know
know
{ Node[11]|'name' = 'you' : String }
{ Node[12]|'name' = 'him' : String }

Example request

- **POST** `http://localhost:7474/db/data/ext/GremlinPlugin/graphdb/execute_script`
- **Accept:** `application/json`
- **Content-Type:** `application/json`

```
{
  "script":"g.v(13).out.sort{it.name}.toList()"
}
```

Example response

- **200:** `OK`
- **Content-Type:** `application/json`

```
[ {
  "outgoing_relationships" : "http://localhost:7474/db/data/node/12/relationships/out",
  "data" : {
    "name" : "him"
  },
  "traverse" : "http://localhost:7474/db/data/node/12/traverse/{returnType}",
  "all_typed_relationships" : "http://localhost:7474/db/data/node/12/relationships/all/{-list|&|types}",
  "property" : "http://localhost:7474/db/data/node/12/properties/{key}",
  "self" : "http://localhost:7474/db/data/node/12",
  "properties" : "http://localhost:7474/db/data/node/12/properties",
  "outgoing_typed_relationships" : "http://localhost:7474/db/data/node/12/relationships/out/{-list|&|types}",
  "incoming_relationships" : "http://localhost:7474/db/data/node/12/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://localhost:7474/db/data/node/12/relationships",
  "paged_traverse" : "http://localhost:7474/db/data/node/12/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://localhost:7474/db/data/node/12/relationships/all",
  "incoming_typed_relationships" : "http://localhost:7474/db/data/node/12/relationships/in/{-list|&|types}"
}, {
  "outgoing_relationships" : "http://localhost:7474/db/data/node/11/relationships/out",
  "data" : {
    "name" : "you"
  },
  "traverse" : "http://localhost:7474/db/data/node/11/traverse/{returnType}",
  "all_typed_relationships" : "http://localhost:7474/db/data/node/11/relationships/all/{-list|&|types}",
  "property" : "http://localhost:7474/db/data/node/11/properties/{key}",
  "self" : "http://localhost:7474/db/data/node/11",
  "properties" : "http://localhost:7474/db/data/node/11/properties",
  "outgoing_typed_relationships" : "http://localhost:7474/db/data/node/11/relationships/out/{-list|&|types}",
  "incoming_relationships" : "http://localhost:7474/db/data/node/11/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://localhost:7474/db/data/node/11/relationships",
  "paged_traverse" : "http://localhost:7474/db/data/node/11/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://localhost:7474/db/data/node/11/relationships/all",
  "incoming_typed_relationships" : "http://localhost:7474/db/data/node/11/relationships/in/{-list|&|types}"
} ]
```

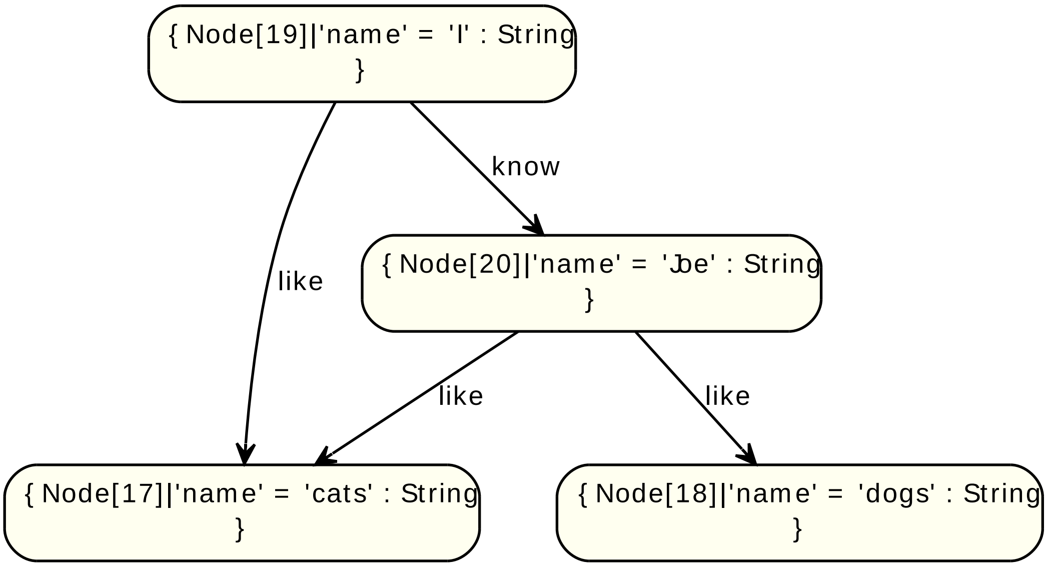
6.13.4. Send a Gremlin Script - JSON encoded with table results

To send a Script JSON encoded, set the payload Content-Type Header. In this example, find all the things that my friends like, and return a table listing my friends by their name, and the names of the things they like in a table with two columns, ignoring the third named step variable `I`. Remember that everything in Gremlin is an iterator - in order to populate the result table `t`, iterate through the pipes with `>> -1`.

Raw script source

```
i = g.v(19)
t= new Table()
i.as('I').out('know').as('friend').out('like').as('likes').table(t,['friend','likes']){it.name}{it.name} >> -1
t
```

Final Graph:


{ Node[19]|'name' = 'I' : String }
know
like
{ Node[20]|'name' = 'Joe' : String }
like
like
{ Node[17]|'name' = 'cats' : String }
{ Node[18]|'name' = 'dogs' : String }

Example request

- **POST** `http://localhost:7474/db/data/ext/GremlinPlugin/graphdb/execute_script`
- **Accept:** `application/json`
- **Content-Type:** `application/json`

```
{
  "script":"i = g.v(19);t= new Table();i.as('I').out('know').as('friend').out('like').as('likes').table(t,['fr
}
```

Example response

- **200:** `OK`
- **Content-Type:** `application/json`

```
{
  "data" : [ [ "Joe", "cats" ], [ "Joe", "dogs" ] ],
  "columns" : [ "friend", "likes" ]
}
```

6.13.5. Set script variables

To set variables in the bindings for the Gremlin Script Engine on the server, you can include a `params` parameter with a String representing a JSON map of variables to set to initial values. These can then be accessed as normal variables within the script.

Final Graph:

Example request

- **POST** `http://localhost:7474/db/data/ext/GremlinPlugin/graphdb/execute_script`
- **Accept:** `application/json`
- **Content-Type:** `application/json`

```
{
  "script":"meaning_of_life",
  "params":{
    "meaning_of_life" : 42.0
  }
}
```

Example response

- **200:** `OK`
- **Content-Type:** `application/json`

```
42.0
```

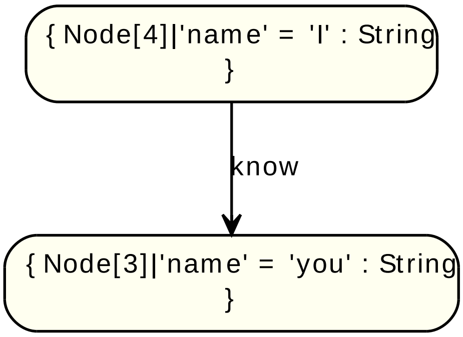
6.13.6. Send a Gremlin Script with variables in a JSON Map

Send a Gremlin Script, as JSON payload and additional parameters

Raw script source

```
g.v(me).out
```

Final Graph:


{ Node[4]|'name' = 'I' : String }
know
{ Node[3]|'name' = 'you' : String }

Example request

- **POST** `http://localhost:7474/db/data/ext/GremlinPlugin/graphdb/execute_script`
- **Accept:** `application/json`
- **Content-Type:** `application/json`

```
{
   "script" : "g.v(me).out",
   "params" : {
     "me" : 4
  }
}
```

Example response

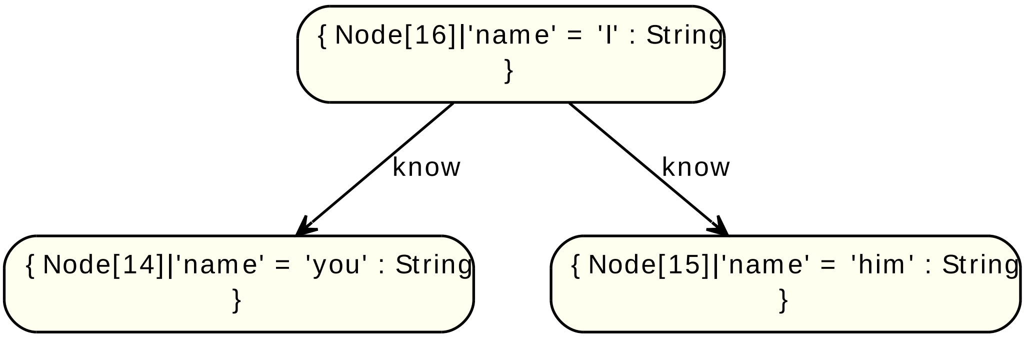
- **200:** `OK`
- **Content-Type:** `application/json`

```
[ {
  "outgoing_relationships" : "http://localhost:7474/db/data/node/3/relationships/out",
  "data" : {
    "name" : "you"
  },
  "traverse" : "http://localhost:7474/db/data/node/3/traverse/{returnType}",
  "all_typed_relationships" : "http://localhost:7474/db/data/node/3/relationships/all/{-list|&|types}",
  "property" : "http://localhost:7474/db/data/node/3/properties/{key}",
  "self" : "http://localhost:7474/db/data/node/3",
  "properties" : "http://localhost:7474/db/data/node/3/properties",
  "outgoing_typed_relationships" : "http://localhost:7474/db/data/node/3/relationships/out/{-list|&|types}",
  "incoming_relationships" : "http://localhost:7474/db/data/node/3/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://localhost:7474/db/data/node/3/relationships",
  "paged_traverse" : "http://localhost:7474/db/data/node/3/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://localhost:7474/db/data/node/3/relationships/all",
  "incoming_typed_relationships" : "http://localhost:7474/db/data/node/3/relationships/in/{-list|&|types}"
} ]
```

6.13.7. Return paths from a Gremlin script

The following script returns a sorted list of all nodes connected via outgoing relationships to node 1, sorted by their `name`-property.

Final Graph:


{ Node[16]|'name' = 'I' : String }
know
know
{ Node[14]|'name' = 'you' : String }
{ Node[15]|'name' = 'him' : String }

Example request

- **POST** `http://localhost:7474/db/data/ext/GremlinPlugin/graphdb/execute_script`
- **Accept:** `application/json`
- **Content-Type:** `application/json`

```
{
  "script":"g.v(16).out.name.paths"
}
```

Example response

- **200:** `OK`
- **Content-Type:** `application/json`

```
[ "[v[16], v[14], you]", "[v[16], v[15], him]" ]
```

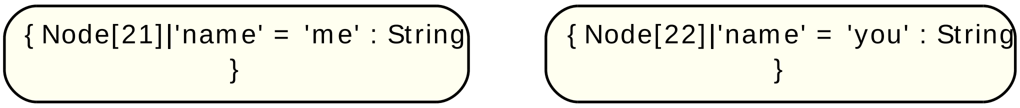
6.13.8. Send an arbitrary Groovy script - Lucene sorting

This example demonstrates that you via the Groovy runtime embedded with the server have full access to all of the servers Java APIs. The below example creates Nodes in the database both via the Blueprints and the Neo4j API indexes the nodes via the native Neo4j Indexing API constructs a custom Lucene sorting and searching returns a Neo4j IndexHits result iterator.

Raw script source

```
import org.neo4j.graphdb.index.*
import org.neo4j.index.lucene.*
import org.apache.lucene.search.*
neo4j = g.getRawGraph()
tx = neo4j.beginTx()
meVertex = g.addVertex([name:'me'])
meNode = meVertex.getRawVertex()
youNode = neo4j.createNode()
youNode.setProperty('name','you')
idxManager = neo4j.index()
personIndex = idxManager.forNodes('persons')
personIndex.add(meNode,'name',meVertex.name)
personIndex.add(youNode,'name',youNode.getProperty('name'))
tx.success()
tx.finish()
query = new QueryContext( 'name:*' ).sort( new Sort(new SortField( 'name',SortField.STRING, true ) ) )
results = personIndex.query( query )
```

Final Graph:


{ Node[21]|'name' = 'me' : String }
{ Node[22]|'name' = 'you' : String }

Example request

- **POST** `http://localhost:7474/db/data/ext/GremlinPlugin/graphdb/execute_script`
- **Accept:** `application/json`
- **Content-Type:** `application/json`

```
{
  "script":"import org.neo4j.graphdb.index.*;import org.neo4j.index.lucene.*;import org.apache.lucene.search.*
}
```

Example response

- **200:** `OK`
- **Content-Type:** `application/json`

```
[ {
  "outgoing_relationships" : "http://localhost:7474/db/data/node/22/relationships/out",
  "data" : {
    "name" : "you"
  },
  "traverse" : "http://localhost:7474/db/data/node/22/traverse/{returnType}",
  "all_typed_relationships" : "http://localhost:7474/db/data/node/22/relationships/all/{-list|&|types}",
  "property" : "http://localhost:7474/db/data/node/22/properties/{key}",
  "self" : "http://localhost:7474/db/data/node/22",
  "properties" : "http://localhost:7474/db/data/node/22/properties",
  "outgoing_typed_relationships" : "http://localhost:7474/db/data/node/22/relationships/out/{-list|&|types}",
  "incoming_relationships" : "http://localhost:7474/db/data/node/22/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://localhost:7474/db/data/node/22/relationships",
  "paged_traverse" : "http://localhost:7474/db/data/node/22/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://localhost:7474/db/data/node/22/relationships/all",
  "incoming_typed_relationships" : "http://localhost:7474/db/data/node/22/relationships/in/{-list|&|types}"
}, {
  "outgoing_relationships" : "http://localhost:7474/db/data/node/21/relationships/out",
  "data" : {
    "name" : "me"
  },
  "traverse" : "http://localhost:7474/db/data/node/21/traverse/{returnType}",
  "all_typed_relationships" : "http://localhost:7474/db/data/node/21/relationships/all/{-list|&|types}",
  "property" : "http://localhost:7474/db/data/node/21/properties/{key}",
  "self" : "http://localhost:7474/db/data/node/21",
  "properties" : "http://localhost:7474/db/data/node/21/properties",
  "outgoing_typed_relationships" : "http://localhost:7474/db/data/node/21/relationships/out/{-list|&|types}",
  "incoming_relationships" : "http://localhost:7474/db/data/node/21/relationships/in",
  "extensions" : {
  },
  "create_relationship" : "http://localhost:7474/db/data/node/21/relationships",
  "paged_traverse" : "http://localhost:7474/db/data/node/21/paged/traverse/{returnType}{?pageSize,leaseTime}",
  "all_relationships" : "http://localhost:7474/db/data/node/21/relationships/all",
  "incoming_typed_relationships" : "http://localhost:7474/db/data/node/21/relationships/in/{-list|&|types}"
} ]
```

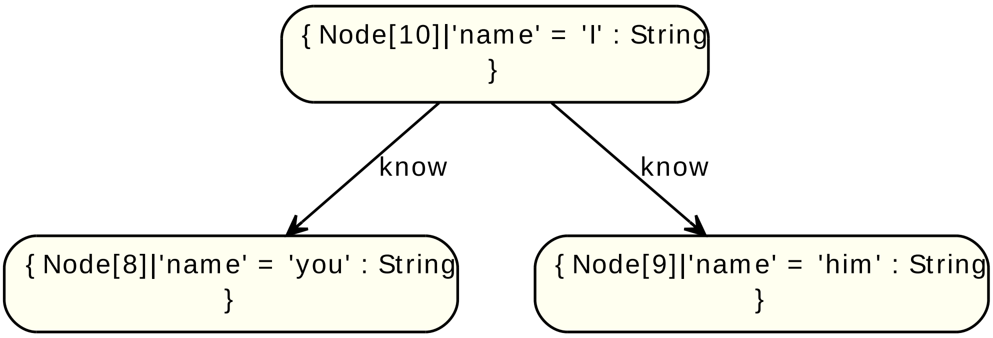
6.13.9. Emit a sample graph

Exporting a graph can be done by simple emitting the appropriate String.

Raw script source

```
writer = new GraphMLWriter(g)
out = new java.io.ByteArrayOutputStream()
writer.outputGraph(out)
result = out.toString()
```

Final Graph:


{ Node[10]|'name' = 'I' : String }
know
know
{ Node[8]|'name' = 'you' : String }
{ Node[9]|'name' = 'him' : String }

Example request

- **POST** `http://localhost:7474/db/data/ext/GremlinPlugin/graphdb/execute_script`
- **Accept:** `application/json`
- **Content-Type:** `application/json`

```
{
  "script":"writer = new GraphMLWriter(g);out = new java.io.ByteArrayOutputStream();writer.outputGraph(out);res
}
```

Example response

- **200:** `OK`
- **Content-Type:** `application/json`

```
"<?xml version=\"1.0\" ?><graphml xmlns=\"http://graphml.graphdrawing.org/xmlns\"><key id=\"name\" for=\"node\
```

6.13.10. HyperEdges - find user roles in groups

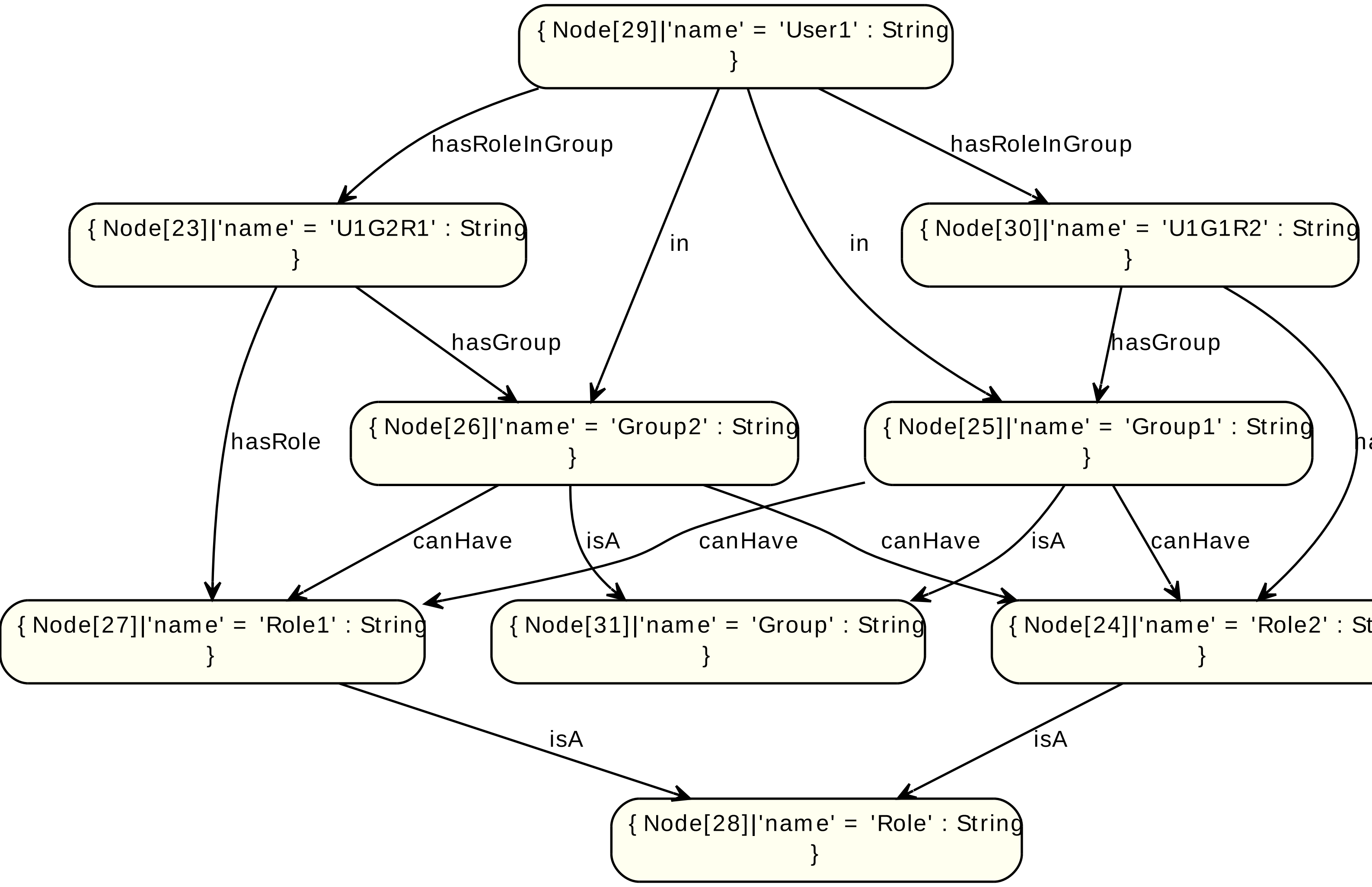
Imagine a user being part of different groups. A group can have different roles, and a user can be part of different groups. He also can have different roles in different groups apart from the membership. The association of a User, a Group and a Role can be referred to as a *HyperEdge*. However, it can be easily modeled in a property graph as a node that captures this n-ary relationship, as depicted below in the `U1G2R1` node.

To find out in what roles a user is for a particular groups (here *Group2*), the following script can traverse this HyperEdge node and provide answers.

Raw script source

```
g.v(29).out('hasRoleInGroup').as('hyperedge').out('hasGroup').filter{it.name=='Group2'}.back('hyperedge').out(
```

Final Graph:


{ Node[29]|'name' = 'User1' : String }
hasRoleInGroup
hasRoleInGroup
in
in
{ Node[23]|'name' = 'U1G2R1' : String }
{ Node[30]|'name' = 'U1G1R2' : String }
hasGroup
hasGroup
hasRole
{ Node[26]|'name' = 'Group2' : String }
{ Node[25]|'name' = 'Group1' : String }
ha
canHave
isA
canHave
canHave
isA
canHave
{ Node[27]|'name' = 'Role1' : String }
{ Node[31]|'name' = 'Group' : String }
{ Node[24]|'name' = 'Role2' : St
isA
isA
{ Node[28]|'name' = 'Role' : String }

Example request

- **POST** `http://localhost:7474/db/data/ext/GremlinPlugin/graphdb/execute_script`
- **Accept:** `application/json`
- **Content-Type:** `application/json`

```
{
  "script":"g.v(29).out('hasRoleInGroup').as('hyperedge').out('hasGroup').filter{it.name=='Group2'}.back('hype
}
```

Example response

- **200:** OK
- **Content-Type:** application/json

```
[ "Role1" ]
```

6.14. Cypher Plugin

The Neo4j Cypher Plugin enables querying with the Chapter 4, *Cypher Query Language*. The results are returned as a list of string headers (`columns`), and a `data` part, consisting of a list of all rows, every row consisting of a list of REST representations of the field value - `Node`, `Relationship` or any simple value like `String`.

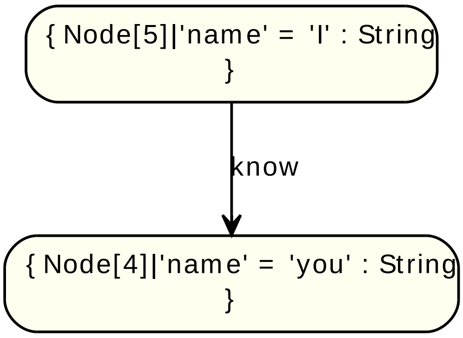
6.14.1. Send a Query

Documentation not available

Cypher query

```
start x  = (5) return x.dummy
```

Final Graph:


{ Node[5]|'name' = 'I' : String }
know
{ Node[4]|'name' = 'you' : String }

Example request

- **`POST`** `http://localhost:7474/db/data/ext/CypherPlugin/graphdb/execute_query`
- **`Accept:`** `application/json`
- **`Content-Type:`** `application/json`

```
{"query": "start x  = (5) return x.dummy"}
```

Example response

- **`400:`** `Bad Request`
- **`Content-Type:`** `application/json`

```
{
  "message" : "dummy property not found for NodeImpl#5.",
  "exception" : "org.neo4j.graphdb.NotFoundException: dummy property not found for NodeImpl#5.",
  "stacktrace" : [ "org.neo4j.kernel.impl.core.Primitive.newPropertyNotFoundException(Primitive.java:173)", "o
}
```

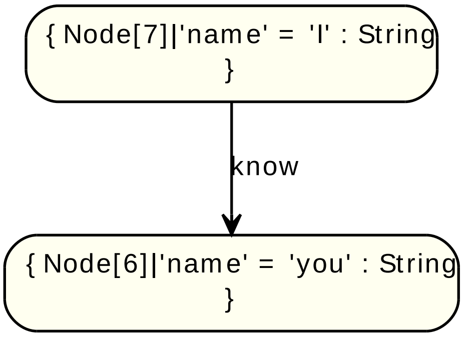
6.14.2. Return paths

Paths can be returned together with other return types by just specifying returns.

Cypher query

```
start x  = (7) match path = (x--friend) return path, friend.name
```

Final Graph:


{ Node[7]|'name' = 'I' : String }
know
{ Node[6]|'name' = 'you' : String }

Example request

- **POST** `http://localhost:7474/db/data/ext/CypherPlugin/graphdb/execute_query`
- **Accept:** `application/json`
- **Content-Type:** `application/json`

```
{"query": "start x  = (7) match path = (x--friend) return path, friend.name"}
```

Example response

- **200:** `OK`
- **Content-Type:** `application/json`

```
{
  "data" : [ [ {
    "start" : "http://localhost:7474/db/data/node/7",
    "nodes" : [ "http://localhost:7474/db/data/node/7", "http://localhost:7474/db/data/node/6" ],
    "length" : 3,
    "relationships" : [ "http://localhost:7474/db/data/relationship/3" ],
    "end" : "http://localhost:7474/db/data/node/6"
  }, "you" ] ],
  "columns" : [ "path", "friend.name" ]
}
```

Chapter 7. Indexing

Indexing in Neo4j can be done in two different ways:

1. The database itself is a *natural index* consisting of its relationships of different types between nodes. For example a tree structure can be layered on top of the data and used for index lookups performed by a traverser.
2. Separate index engines can be used, with Apache Lucene <http://lucene.apache.org/java/3_1_0/index.html> being the default backend included with Neo4j.

This chapter demonstrate how to use the second type of indexing, focusing on Lucene.

7.1. Introduction

Indexing operations are part of the Neo4j index API <http://components.neo4j.org/neo4j/1.4.2/apidocs/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 <http://components.neo4j.org/neo4j/1.4.2/apidocs/org/neo4j/graphdb/Node.html> or relationships <http://components.neo4j.org/neo4j/1.4.2/apidocs/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.

Note

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

7.2. 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 Section 7.8, "Relationship indexes" for more information specific to relationship indexes.

See Section 7.10, "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" );
```

7.3. Delete

Indexes can be deleted. When deleting, the entire contents of the index will be removed as well as its associated configuration. A new 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() <http://components.neo4j.org/neo4j/1.4.2/apidocs/org/neo4j/graphdb/index/Index.html#delete%28%29> 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.

7.4. 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:

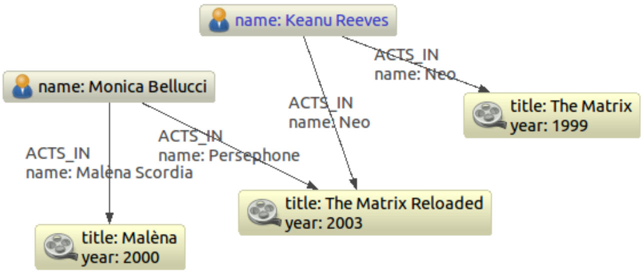
```
// Actors
Node reeves = graphDb.createNode();
actors.add( reeves, "name", "Keanu Reeves" );
Node bellucci = graphDb.createNode();
actors.add( bellucci, "name", "Monica Bellucci" );
// multiple values for a field
actors.add( bellucci, "name", "La Bellucci" );
// Movies
Node theMatrix = graphDb.createNode();
movies.add( theMatrix, "title", "The Matrix" );
movies.add( theMatrix, "year", 1999 );
Node theMatrixReloaded = graphDb.createNode();
movies.add( theMatrixReloaded, "title", "The Matrix Reloaded" );
movies.add( theMatrixReloaded, "year", 2003 );
Node malena = graphDb.createNode();
movies.add( malena, "title", "Malèna" );
movies.add( malena, "year", 2000 );
```

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
DynamicRelationshipType ACTS_IN = DynamicRelationshipType.withName( "ACTS_IN" );
// create relationships
Relationship role1 = reeves.createRelationshipTo( theMatrix, ACTS_IN );
roles.add( role1, "name", "Neo" );
Relationship role2 = reeves.createRelationshipTo( theMatrixReloaded, ACTS_IN );
roles.add( role2, "name", "Neo" );
Relationship role3 = bellucci.createRelationshipTo( theMatrixReloaded, ACTS_IN );
roles.add( role3, "name", "Persephone" );
Relationship role4 = bellucci.createRelationshipTo( malena, ACTS_IN );
roles.add( role4, "name", "Malèna Scordia" );
```

Assuming we set the same key-value pairs as properties as well, our example graph looks like this:



7.5. Remove

Removing <http://components.neo4j.org/neo4j/1.4.2/apidocs/org/neo4j/graphdb/index/Index.html#remove%28T,%20java.lang.String,%20java.lang.Object%29> 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" );
```

7.6. Update

Important

To update an index entry, old one must be removed and a new one added.

Remember that a node or relationship can be associated with any number of key-value pairs in an index, which 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 for that demonstrates how it's done:

```
// create a node with a property
Node fishburn = graphDb.createNode();
fishburn.setProperty( "name", "Fishburn" );
// index it
actors.add( fishburn, "name", fishburn.getProperty( "name" ) );
// update the index entry
actors.remove( fishburn, "name", fishburn.getProperty( "name" ) );
fishburn.setProperty( "name", "Laurence Fishburn" );
actors.add( fishburn, "name", fishburn.getProperty( "name" ) );
```

7.7. Search

An index can be searched in two ways, get <http://components.neo4j.org/neo4j/1.4.2/apidocs/org/neo4j/graphdb/index/Index.html#get%28java.lang.String,%20java.lang.Object%29> and query <http://components.neo4j.org/neo4j/1.4.2/apidocs/org/neo4j/graphdb/index/Index.html#query%28java.lang.String,%20java.lang.Object%29>. 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/java/3_1_0/queryparsersyntax.html> can be used directly with the default indexing backend.

7.7.1. Get

This is how to search for a single exact match:

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

IndexHits <http://components.neo4j.org/neo4j/1.4.2/apidocs/org/neo4j/graphdb/index/IndexHits.html> is an `Iterable` with some additional useful methods. For example getSingle() <http://components.neo4j.org/neo4j/1.4.2/apidocs/org/neo4j/graphdb/index/IndexHits.html#getSingle%28%29> 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();
}
```

Important

In you don’t iterate through all the hits, IndexHits.close() <http://components.neo4j.org/neo4j/1.4.2/apidocs/org/neo4j/graphdb/index/IndexHits.html#close%28%29> must be called explicitly.

7.7.2. 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.
}
```

Note

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

Caution

You can’t have *any whitespace* in the search term with this syntax. See Section 7.11.3, “Querying with Lucene Query objects” for how to do that.

7.8. 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 nodes These extra methods reside in the RelationshipIndex <http://components.neo4j.org/neo4j/1.4.2/apidocs/org/neo4j/graphdb/index/RelationshipIndex.html> interface which extends Index<Relationship> <http://components.neo4j.org/neo4j/1.4.2/apidocs/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.finish();
// and now we can search for it:
IndexHits<Relationship> typeHits;
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.

7.9. Scores

The `IndexHits` interface exposes scoring <http://components.neo4j.org/neo4j/1.4.2/apidocs/org/neo4j/graphdb/index/IndexHits.html#currentScore%28%29> 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 Section 7.11.2, “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() );
}
```

7.10. 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();
```

Tip

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 7.1. 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/java/3_1_0/api/core/org/apache/lucene/analysis/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/java/3_1_0/api/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.

7.11. Extra features for Lucene indexes

7.11.1. 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 <http://components.neo4j.org/neo4j-lucene-index/1.4.2/apidocs/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 ) );
```

Note

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 ) );
```

7.11.2. Sorting

Lucene performs sorting very well, and that is also exposed in the index backend, through the QueryContext <http://components.neo4j.org/neo4j-lucene-index/1.4.2/apidocs/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)
}
```

7.11.3. 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/java/3_1_0/api/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.

7.11.4. Compound queries

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

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

Caution

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

7.11.5. 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 <http://components.neo4j.org/neo4j-lucene-index/1.4.2/apidocs/org/neo4j/index/lucene/QueryContext.html> class:

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

7.11.6. Caching

If your index lookups becomes a performance bottle neck, caching can be enabled for certain keys in certain indexes (key locations) to speed up get requests. The caching is implemented with an LRU <http://en.wikipedia.org/wiki/Cache_algorithms#Least_Recently_Used> cache so that only the most recently accessed results are cached (with "results" meaning a query result of a get request, not a single entity). You can control the size of the cache (the maximum number of results) per index key.

```
Index<Node> index = graphDb.index().forNodes( "actors" );
( (LuceneIndex<Node>) index ).setCacheCapacity( "name", 300000 );
```

Caution

This setting is not persisted after shutting down the database. This means: set this value after each startup of the database if you want to keep it.

7.12. Batch insertion

Neo4j has a batch insertion mode intended for initial imports, which must run in a single thread and bypasses transactions and other checks in favor of performance. Indexing during batch insertion is done using BatchInserterIndex <http://components.neo4j.org/neo4j/1.4.2/apidocs/org/neo4j/graphdb/index/BatchInserterIndex.html> which are provided via BatchInserterIndexProvider <http://components.neo4j.org/neo4j/1.4.2/apidocs/org/neo4j/graphdb/index/BatchInserterIndexProvider.html>. An example:

```
BatchInserter inserter = new BatchInserterImpl( "target/neo4jdb-batchinsert" );
BatchInserterIndexProvider indexProvider = new LuceneBatchInserterIndexProvider( inserter );
BatchInserterIndex actors = indexProvider.nodeIndex( "actors", MapUtil.stringMap( "type", "exact" ) );
actors.setCacheCapacity( "name", 100000 );

Map<String, Object> properties = MapUtil.map( "name", "Keanu Reeves" );
long node = inserter.createNode( properties );
actors.add( node, properties );

// Make sure to shut down the index provider
indexProvider.shutdown();
inserter.shutdown();
```

The configuration parameters are the same as mentioned in Section 7.10, “Configuration and fulltext indexes”.

7.12.1. Best practices

Here are some pointers to get the most performance out of `BatchInserterIndex`:

- Try to avoid flushing <http://components.neo4j.org/neo4j/1.4.2/apidocs/org/neo4j/graphdb/index/BatchInserterIndex.html#flush%28%29> too often because each flush will result in all additions (since last flush) to be visible to the querying methods, and publishing those changes can be a performance penalty.
- Have (as big as possible) phases where one phase is either only writes or only reads, and don’t forget to flush after a write phase so that those changes becomes visible to the querying methods.
- Enable caching <http://components.neo4j.org/neo4j/1.4.2/apidocs/org/neo4j/graphdb/index/BatchInserterIndex.html#setCacheCapacity%28java.lang.String,%20int%29> for keys you know you’re going to do lookups for later on to increase performance significantly (though insertion performance may degrade slightly).

7.13. Automatic Indexing

Neo4j provides a single index for nodes and one for relationships in each database that automatically follow property values as they are added, deleted and changed on database primitives. This functionality is called *auto indexing* and is controlled both from the database configuration Map and through its own API.

Caution

This is an experimental feature. Expect changes in the API and do not rely on it for production data handling.

7.13.1. Configuration

By default Auto Indexing is off for both Nodes and Relationships. To enable it on database startup set the configuration options `Config.NODE_AUTO_INDEXING` and `Config.RELATIONSHIP_AUTO_INDEXING` to the string "`true`".

If you just enable auto indexing as above, then still *no* property will be auto indexed. To define which property names you want the auto indexer to monitor as a configuration parameter, set the `Config.{NODE,RELATIONSHIP}_KEYS_INDEXABLE` option to a String that is a comma separated concatenation of the property names you want auto indexed.

```
/*
 * Creating the configuration, adding nodeProp1 and nodeProp2 as
 * auto indexed properties for Nodes and relProp1 and relProp2 as
 * auto indexed properties for Relationships. Only those will be
 * indexed. We also have to enable auto indexing for both these
 * primitives explicitly.
 */
Map<String, String> config = new HashMap<String, String>();
config.put( Config.NODE_KEYS_INDEXABLE, "nodeProp1, nodeProp2" );
config.put( Config.RELATIONSHIP_KEYS_INDEXABLE, "relProp1, relProp2" );
config.put( Config.NODE_AUTO_INDEXING, "true" );
config.put( Config.RELATIONSHIP_AUTO_INDEXING, "true" );

EmbeddedGraphDatabase graphDb = new EmbeddedGraphDatabase(
        getStoreDir( "testConfig" ), config );

Transaction tx = graphDb.beginTx();
Node node1 = null, node2 = null;
Relationship rel = null;
try
{
    // Create the primitives
    node1 = graphDb.createNode();
    node2 = graphDb.createNode();
    rel = node1.createRelationshipTo( node2,
            DynamicRelationshipType.withName( "DYNAMIC" ) );

    // Add indexable and non-indexable properties
    node1.setProperty( "nodeProp1", "nodeProp1Value" );
    node2.setProperty( "nodeProp2", "nodeProp2Value" );
    node1.setProperty( "nonIndexed", "nodeProp2NonIndexedValue" );
    rel.setProperty( "relProp1", "relProp1Value" );
    rel.setProperty( "relPropNonIndexed", "relPropValueNonIndexed" );

    // Make things persistent
    tx.success();
}
catch ( Exception e )
```

```
{
    tx.failure();
}
finally
{
    tx.finish();
}
```

7.13.2. Search

The usefulness of the auto indexing functionality comes of course from the ability to actually query the index and retrieve results. To that end, you can acquire a `ReadableIndex` object from the `AutoIndexer` that exposes all the query and get methods of a full `Index` <http://components.neo4j.org/neo4j/1.4.2/apidocs/org/neo4j/graphdb/index/Index.html> with exactly the same functionality. Continuing from the previous example, accessing the index is done like this:

```
// Get the Node auto index
ReadableIndex<Node> autoNodeIndex = graphDb.index().getNodeAutoIndexer().getAutoIndex();
// node1 and node2 both had auto indexed properties, get them
assertEquals( node1,
        autoNodeIndex.get( "nodeProp1", "nodeProp1Value" ).getSingle() );
assertEquals( node2,
        autoNodeIndex.get( "nodeProp2", "nodeProp2Value" ).getSingle() );
// node2 also had a property that should be ignored.
assertFalse( autoNodeIndex.get( "nonIndexed",
        "nodeProp2NonIndexedValue" ).hasNext() );

// Get the relationship auto index
ReadableIndex<Relationship> autoRelIndex = graphDb.index().getRelationshipAutoIndexer().getAutoIndex();
// One property was set for auto indexing
assertEquals( rel,
        autoRelIndex.get( "relProp1", "relProp1Value" ).getSingle() );
// The rest should be ignored
assertFalse( autoRelIndex.get( "relPropNonIndexed",
        "relPropValueNonIndexed" ).hasNext() );
```

7.13.3. Runtime Configuration

The same options that are available during database creation via the configuration can also be set during runtime via the `AutoIndexer` API.

Gaining access to the `AutoIndexer` API and adding two `Node` and one +Relationship properties to auto index is done like so:

```

// Start without any configuration
EmbeddedGraphDatabase graphDb = new EmbeddedGraphDatabase(
        getStoreDir( "testAPI" ) );

// Get the Node AutoIndexer, set nodeProp1 and nodeProp2 as auto
// indexed.
AutoIndexer<Node> nodeAutoIndexer = graphDb.index().getNodeAutoIndexer();
nodeAutoIndexer.startAutoIndexingProperty( "nodeProp1" );
nodeAutoIndexer.startAutoIndexingProperty( "nodeProp2" );

// Get the Relationship AutoIndexer
AutoIndexer<Relationship> relAutoIndexer = graphDb.index().getRelationshipAutoIndexer();
relAutoIndexer.startAutoIndexingProperty( "relProp1" );

// None of the AutoIndexers are enabled so far. Do that now
nodeAutoIndexer.setEnabled( true );
relAutoIndexer.setEnabled( true );

```

Parameters to the AutoIndexers passed through the Configuration and settings made through the API are cumulative. So you can set some beforehand known settings, do runtime checks to augment the initial configuration and then enable the desired auto indexers - the final configuration is the same regardless of the method used to reach it.

7.13.4. Updating the Automatic Index

Updates to the auto indexed properties happen of course automatically as you update them. Removal of properties from the auto index happens for two reasons. One is that you actually removed the property. The other is that you stopped autoindexing on a property. When the latter happens, any primitive you touch and it has that property, it is removed from the auto index, regardless of any operations on the property. When you start or stop auto indexing on a property, no auto update operation happens currently. If you need to change the set of auto indexed properties and have them re-indexed, you currently have to do this by hand. An example will illustrate the above better:

```
/*
 * Creating the configuration
 */
Map<String, String> config = new HashMap<String, String>();
config.put( Config.NODE_KEYS_INDEXABLE, "nodeProp1, nodeProp2" );
config.put( Config.NODE_AUTO_INDEXING, "true" );

EmbeddedGraphDatabase graphDb = new EmbeddedGraphDatabase(
        getStoreDir( "mutations" ), config );

Transaction tx = graphDb.beginTx();
Node node1 = null, node2 = null, node3 = null, node4 = null;
try
{
    // Create the primitives
    node1 = graphDb.createNode();
    node2 = graphDb.createNode();
    node3 = graphDb.createNode();
    node4 = graphDb.createNode();

    // Add indexable and non-indexable properties
    node1.setProperty( "nodeProp1", "nodeProp1Value" );
    node2.setProperty( "nodeProp2", "nodeProp2Value" );
    node3.setProperty( "nodeProp1", "nodeProp3Value" );
    node4.setProperty( "nodeProp2", "nodeProp4Value" );

    // Make things persistent
    tx.success();
}
catch ( Exception e )
{
    tx.failure();
}
finally
{
    tx.finish();
}

/*
 *  Here both nodes are indexed. To demonstrate removal, we stop
 *  autoindexing nodeProp1.
 */
AutoIndexer<Node> nodeAutoIndexer = graphDb.index().getNodeAutoIndexer();
nodeAutoIndexer.stopAutoIndexingProperty( "nodeProp1" );

tx = graphDb.beginTx();
```

```
try
{
    /*
     * nodeProp1 is no longer auto indexed. It will be
     * removed regardless. Note that node3 will remain.
     */
    node1.setProperty( "nodeProp1", "nodeProp1Value2" );
    /*
     * node2 will be auto updated
     */
    node2.setProperty( "nodeProp2", "nodeProp2Value2" );
    /*
     * remove node4 property nodeProp2 from index.
     */
    node4.removeProperty( "nodeProp2" );
    // Make things persistent
    tx.success();
}
catch ( Exception e )
{
    tx.failure();
}
finally
{
    tx.finish();
}

// Verify
ReadableIndex<Node> nodeAutoIndex = nodeAutoIndexer.getAutoIndex();
// node1 is completely gone
assertFalse( nodeAutoIndex.get( "nodeProp1", "nodeProp1Value" ).hasNext() );
assertFalse( nodeAutoIndex.get( "nodeProp1", "nodeProp1Value2" ).hasNext() );
// node2 is updated
assertFalse( nodeAutoIndex.get( "nodeProp2", "nodeProp2Value" ).hasNext() );
assertEquals( node2,
        nodeAutoIndex.get( "nodeProp2", "nodeProp2Value2" ).getSingle() );
/*
 * node3 is still there, despite its nodeProp1 property not being monitored
 * any more because it was not touched, in contrast with node1.
 */
assertEquals( node3,
        nodeAutoIndex.get( "nodeProp1", "nodeProp3Value" ).getSingle() );
// Finally, node4 is removed because the property was removed.
assertFalse( nodeAutoIndex.get( "nodeProp2", "nodeProp4Value" ).hasNext() );
```

Caution

If you start the database with auto indexing enabled but different auto indexed properties than the last run, then already auto-indexed entities will be deleted as you work with them. Make sure that the monitored set is what you want before enabling the functionality.

Chapter 8. Graph Algorithms

Neo4j graph algorithms is a component that contains Neo4j implementations of some common algorithms for graphs. It includes algorithms like:

- Shortest paths,
- all paths,
- all simple paths,
- Dijkstra and
- A*.

8.1. Introduction

The graph algorithms are found in the `neo4j-graph-algo` component, which is included in the standard Neo4j download.

- Javadocs <http://components.neo4j.org/neo4j/1.4.2/apidocs/org/neo4j/graphalgo/package-summary.html>
- Download <http://search.maven.org/#search%7Cgav%7C1%7Cg%3A%22org.neo4j%22%20AND%20a%3A%22neo4j-graph-algo%22>
- Source code <https://github.com/neo4j/community/tree/1.4.2/graph-algo>

For information on how to use neo4j-graph-algo as a dependency with Maven and other dependency management tools, see `org.neo4j:neo4j-graph-algo` `<http://search.maven.org/#search%7Cgav%7C1%7Cg%3A%22org.neo4j%22%20AND%20a%3A%22neo4j-graph-algo%22>` Note that it should be used with the same version of `org.neo4j:neo4j-kernel` `<http://search.maven.org/#search%7Cgav%7C1%7Cg%3A%22org.neo4j%22%20AND%20a%3A%22neo4j-kernel%22>`. Different versions of the graph-algo and kernel components are not compatible in the general case. Both components are included transitively by the `org.neo4j:neo4j` `<http://search.maven.org/#search%7Cgav%7C1%7Cg%3A%22org.neo4j%22%20AND%20a%3A%22neo4j%22>` artifact which makes it simple to keep the versions in sync.

The starting point to find and use graph algorithms is `GraphAlgoFactory` `<http://components.neo4j.org/neo4j/1.4.2/apidocs/org/neo4j/graphalgo/GraphAlgoFactory.html>`.

For examples, see Section 12.6, “Graph Algorithm examples” (embedded database) and Section 6.11, “Built-in Graph Algorithms” (REST API).

Chapter 9. High Availability

Note
The High Availability features are only available in the Neo4j Enterprise Edition.

Neo4j High Availability or “Neo4j HA” provides the following two main features:

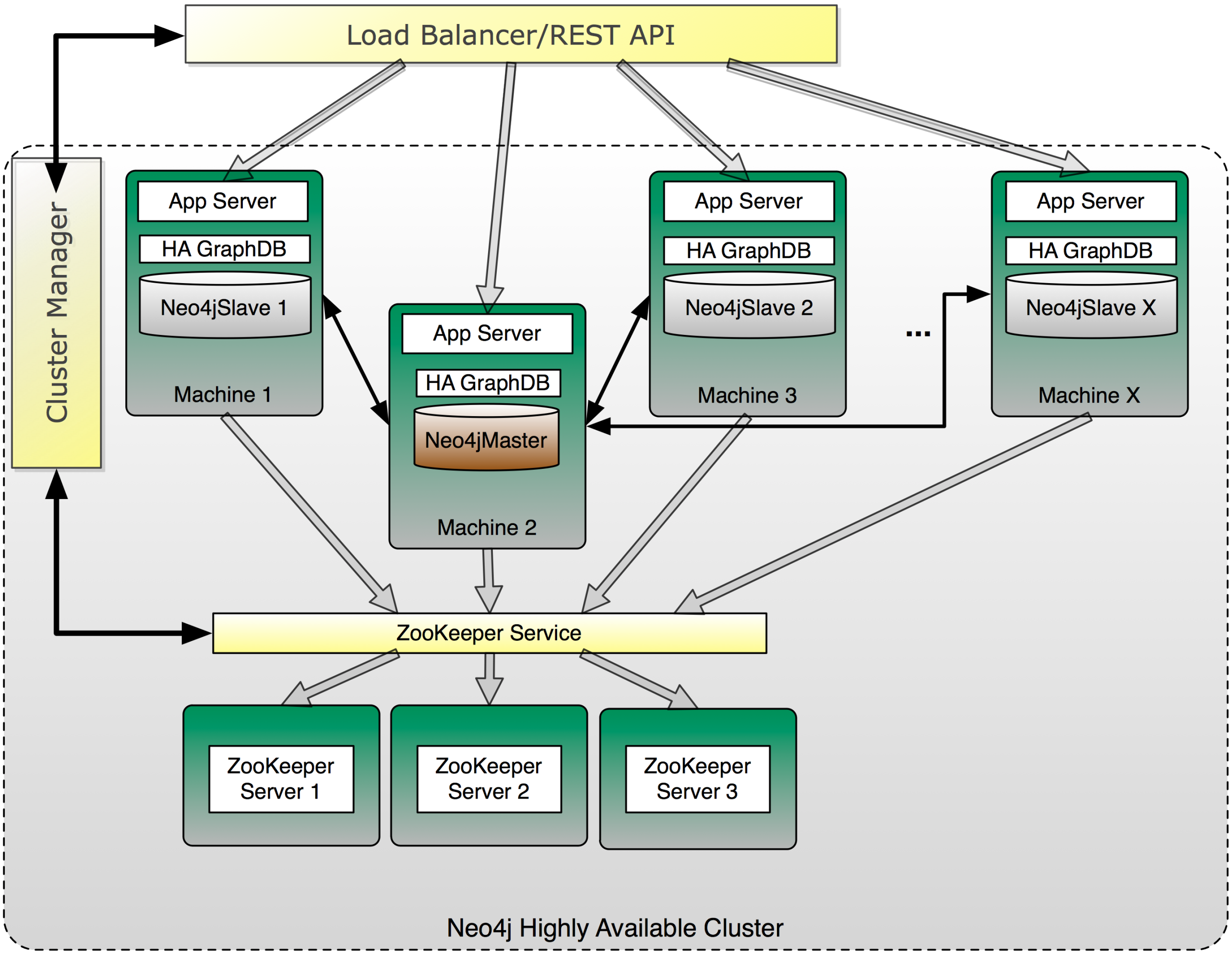
1. It enables a *fault-tolerant database architecture*, where several Neo4j slave databases can be configured to be exact replicas of a single Neo4j master database. This allows the end-user system to be fully functional and both read and write to the database in the event of hardware failure.
2. It enables a *horizontally scaling read-mostly architecture* that enables the system to handle more read load than a single Neo4j database instance can handle.

9.1. Architecture

Neo4j HA has been designed to make the transition from single machine to multi machine operation simple, by not having to change the already existing application.

Consider an existing application with Neo4j embedded and running on a single machine. To deploy such an application in a multi machine setup the only required change is to switch the creation of the `GraphDatabaseService` from `EmbeddedGraphDatabase` to `HighlyAvailableGraphDatabase`. Since both implement the same interface, no additional changes are required.

Figure 9.1. Typical setup when running multiple Neo4j instances in HA mode



When running Neo4j in HA mode there is always a single master and zero or more slaves. Compared to other master-slave replication setups Neo4j HA can handle writes on a slave so there is no need to redirect writes to the master.

A slave will handle writes by synchronizing with the master to preserve consistency. Updates will however propagate from the master to other slaves eventually so a write from one slave is not immediately visible on all other slaves. This is the only difference between multiple machines running in HA mode compared to single machine operation. All other ACID characteristics are the same.

9.2. Setup and configuration

Neo4j HA can be set up to accommodate differing requirements for load, fault tolerance and available hardware.

Within a cluster, Neo4j HA uses Apache ZooKeeper [1] for master election and propagation of general cluster and machine status information. ZooKeeper can be seen as a distributed coordination service. Neo4j HA requires a ZooKeeper service for initial master election, new master election (current master failing) and to publish general status information about the current Neo4j HA cluster (for example when a machine joined or left the cluster). Read operations through the `GraphDatabaseService` API will always work and even writes can survive ZooKeeper failures if a master is present.

ZooKeeper requires a majority of the ZooKeeper instances to be available to operate properly. This means that the number of ZooKeeper instances should always be an odd number since that will make best use of available hardware.

To further clarify the fault tolerance characteristics of Neo4j HA here are a few example setups:

9.2.1. Small

- 3 physical (or virtual) machines
- 1 ZooKeeper instance running on each machine
- 1 Neo4j HA instance running on each machine

This setup is conservative in the use of hardware while being able to handle moderate read load. It can fully operate when at least 2 of the ZooKeeper instances are running. Since the ZooKeeper service and Neo4j HA are running together on each machine this will in most scenarios mean that only one server is allowed to go down.

9.2.2. Medium

- 5-7+ machines
- ZooKeeper running on 3, 5 or 7 machines
- Neo4j HA can run on 5+ machines

This setup may mean that two different machine setups have to be managed (some machines run both ZooKeeper and Neo4j HA). The fault tolerance will depend on how many machines there are that are running ZooKeeper. With 3 ZooKeeper instances the cluster can survive one ZooKeeper going down, with 5 it can survive 2 and with 7 it can handle 3 ZooKeeper instances failing. The number of Neo4j HA instances that can fail for normal operations is theoretically all but 1 (but for each required master election the ZooKeeper service must be available).

9.2.3. Large

- 8+ total machines
- 3+ Neo4j HA machines
- 5+ Zookeeper, on separate dedicated machines

In this setup all ZooKeeper instances are running on separate machines as a dedicated ZooKeeper service. The dedicated ZooKeeper cluster can handle half of the instances, minus 1, going down. The

---

[1] http://hadoop.apache.org/zookeeper/

Neo4j HA cluster will be able to operate with at least a single live machine. Adding more Neo4j HA instances is very easy in this setup since Zookeeper is operating as a separate service.

9.2.4. Installation Notes

For installation instructions of a High Availability cluster please visit the Neo4j Wiki [2].

Note that while the `HighlyAvailableGraphDatabase` supports the same API as the `EmbeddedGraphDatabase`, it does have additional configuration parameters.

Table 9.1. HighlyAvailableGraphDatabase configuration parameters

Parameter Name	Value	Example value	Required?
`ha.machine_id`	integer >= 0	`1`	yes
`ha.server`	(auto-discovered) host & port to bind when acting as master	`my-domain.com:6001`	no

`ha.zoo_keeper_servers`	comma delimited zookeeper connections	`localhost:2181,`
		`localhost:2182,`
		`localhost:2183`
			yes

`ha.pull_interval`	interval for polling master from a slave, in seconds	`30`	no

Caution

Neo4j's HA setup depends on ZooKeeper which makes certain assumptions about the state of the underlying operating system. In particular ZooKeeper expects that the system time on each machine is set correctly, synchronized with respect to each other. If this is not true, then Neo4j HA will appear to misbehave, caused by seemingly random ZooKeeper hiccups.

---

[2] http://wiki.neo4j.org/content/High_Availability_Cluster

9.3. How Neo4j HA operates

A Neo4j HA cluster operates cooperatively, coordinating activity through Zookeeper.

On startup a Neo4j HA instance will connect to the ZooKeeper service to register itself and ask, "who is master?" If some other machine is master, the new instance will start as slave and connect to that master. If the machine starting up was the first to register — or should become master according to the master election algorithm — it will start as master.

When performing a write transaction on a slave each write operation will be synchronized with the master (locks will be acquired on both master and slave). When the transaction commits it will first occur on the master. If the master commit is successful the transaction will be committed on the slave as well. To ensure consistency, a slave has to be up to date with the master before performing a write operation. This is built into the communication protocol between the slave and master, so that updates will happen automatically if needed.

When performing a write on the master it will execute in the same way as running in normal embedded mode. Currently the master will not push updates to the slave. Instead, slaves can be configured to have a pull interval. Without polling, updates will only happen on slaves whenever they synchronize a write with the master.

Having all writes go through slaves has the benefit that the data will be replicated on two machines. This is recommended to avoid rollbacks in case of a master failure that could potentially happen when the new master is elected.

Whenever a machine becomes unavailable the ZooKeeper service will detect that and remove it from the cluster. If the master goes down a new master will automatically be elected. Normally a new master is elected and started within just a few seconds and during this time no writes can take place (the write will throw an exception). A machine that becomes available after being unavailable will automatically reconnect to the cluster. The only time this is not true is when an old master had changes that did not get replicated to any other machine. If the new master is elected and performs changes before the old master recovers, there will two different versions of the data. The old master will not be able to attach itself to the cluster and will require maintenance (replace the wrong version of the data with the one running in the cluster).

All this can be summarized as:

- Slaves can handle write transactions.
- Updates to slaves are eventual consistent.
- Neo4j HA is fault tolerant and (depending on ZooKeeper setup) can continue to operate from X machines down to a single machine.
- Slaves will be automatically synchronized with the master on a write operation.
- If the master fails a new master will be elected automatically.
- Machines will be reconnected automatically to the cluster whenever the issue that caused the outage (network, maintenance) is resolved.
- Transactions are atomic, consistent and durable but eventually propagated out to other slaves.
- If the master goes down any running write transaction will be rolled back and during master election no write can take place.
- Reads are highly available.

Chapter 10. Operations

This chapter describes how to maintain a Neo4j installation. This includes topics such as backing up the database and monitoring the health of the database as well as diagnosing issues.

10.1. Backup

Note

The Backup features are only available in the Neo4j Enterprise Edition.

Backups are performed over the network live from a running graph database onto a local copy. There are two types of backup: full and incremental.

A *full backup* copies the database files without acquiring any locks, allowing for continued operations on the target instance. This of course means that while copying, transactions will continue and the store will change. For this reason, the transaction that was running when the backup operation started is noted and, when the copy operation completes, all transactions from the latter down to the one happening at the end of the copy are replayed on the backup files. This ensures that the backed up data represent a consistent and up-to-date snapshot of the database storage.

In contrast, *incremental backup* does not copy store files - instead it copies the logs of the transactions that have taken place since the last full or incremental backup which are then replayed over an existing backup store. This makes incremental backups far more efficient that doing full backups every time but they also require that a *full backup* has taken place before they are executed.

Regardless of the mode a backup is created, the resulting files represent a consistent database snapshot and they can be used to boot up a Neo4j instance.

Warning

Due to certain limitations and bugs in the JVM around file locking on some platforms performing full backups could leave a running instance to a graph database in a vulnerable state. You are therefore adviced not to perform any full backups against a running graph database until a workaround has been found. Instead of doing the initial full backup shut down the database, copy the database directory to your backup location and your database again start again. Incremental backups after that to that location are safe to perform.

The database to be backed up is specified using a URI with syntax

<running mode>://<host>[:port]{,<host>[:port]*}

Running mode must be defined and is either *single* for non-HA or *ha* for HA clusters. The <host>[:port] part points to a host running the database, on port *port* if not the default. The additional *host:port* arguments are useful for passing multiple ZooKeeper instances

Important

Backups can only be performed on databases which have the configuration parameter `enable_online_backup=true` set. That will make the backup service available on the default port (6362). To enable the backup service on a different port use for example `enable_online_backup=port=9999` instead.

10.1.1. Embedded and Server

To perform a backup from a running embedded or server database run:

```
# Performing a full backup
./neo4j-backup -full -from single://192.168.1.34 -to /mnt/backup/neo4j-backup

# Performing an incremental backup
./neo4j-backup -incremental -from single://192.168.1.34 -to /mnt/backup/neo4j-backup
```

```
# Performing an incremental backup where the service is registered on a custom port
./neo4j-backup -incremental -from single://192.168.1.34:9999 -to /mnt/backup/neo4j-backup
```

10.1.2. High Availability

To perform a backup on an HA cluster you specify one or more ZooKeeper services managing that cluster.

```
# Performing a full backup from HA cluster, specifying two possible ZooKeeper services
./neo4j-backup -full -from ha://192.168.1.15:2181,192.168.1.16:2181 -to /mnt/backup/neo4j-backup

# Performing an incremental backup from HA cluster, specifying only one ZooKeeper service
./neo4j-backup -incremental -from ha://192.168.1.15:2181 -to /mnt/backup/neo4j-backup
```

10.1.3. Restoring Your Data

The Neo4j backups are fully functional databases. To use a backup, all you need to do replace your database folder with the backup.

10.2. Security

Neo4j in itself does not enforce security on the data level. However, there are different aspects that should be considered when using Neo4j in different scenarios.

10.2.1. Securing access to the Neo4j Server

The Neo4j server currently does not enforce security on the REST access layer. This should be taken care of by external means. We strongly recommend to front a running Neo4j Server with a proxy like Apache `mod_proxy` [1]. This provides a number of advantages:

- Control access to the Neo4j server to specific IP addresses, URL patterns and IP ranges. This can be used to make for instance only the `/db/data` namespace accessible to non-local clients, while the `/db/admin` URLs only respond to a specific IP address.

```
<Proxy *>
  Order Deny,Allow
  Deny from all
  Allow from 192.168.0
</Proxy>
```

- Run Neo4j Server as a non-root user on a Linux/Unix system on a port < 1000 (e.g. port 80) using

```
ProxyPass /neo4jdb/data http://localhost:7474/db/data
ProxyPassReverse /neo4jdb/data http://localhost:7474/db/data
```

- Simple load balancing in a clustered environment to load-balance read load using the Apache `mod_proxy_balancer` [2] plugin

```
<Proxy balancer://mycluster>
BalancerMember http://192.168.1.50:80
BalancerMember http://192.168.1.51:80
</Proxy>
ProxyPass /test balancer://mycluster
```

---

[1] http://httpd.apache.org/docs/2.2/mod/mod_proxy.html

10.3. Monitoring

Note

Most of the monitoring features are only available in the Advanced and Enterprise editions 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.

10.3.1. JMX

How to connect to a Neo4j instance using JMX and JConsole

First, start your embedded database or the Neo4j Server, for instance using

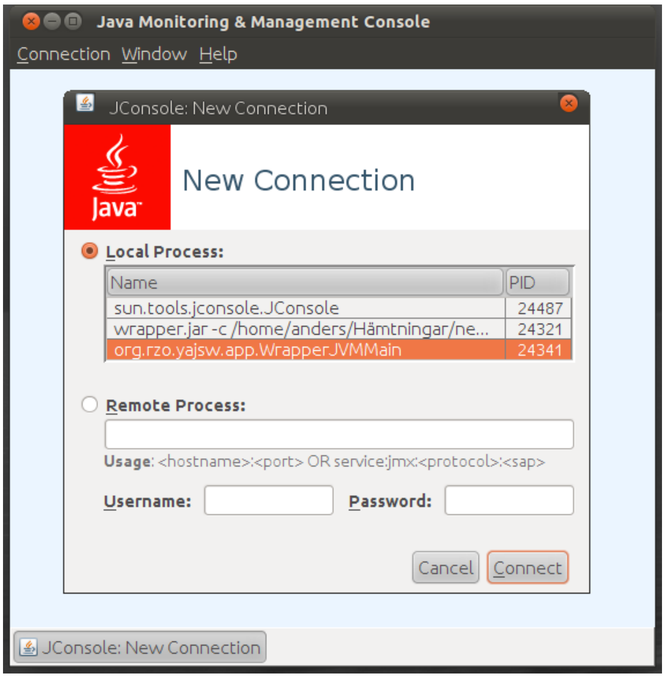
```
$NEO4j_SERVER_HOME/bin/neo4j start
```

Now, start JConsole with

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

Connect to the process running your Neo4j database instance:

Figure 10.1. Connecting JConsole to the Neo4j Java process



● **Local Process:**

Name	PID
sun.tools.jconsole.JConsole	24487
wrapper.jar -c /home/anders/Hämtningar/ne...	24321
org.rzo.yajsw.app.WrapperJVMMain	24341

○ **Remote Process:**

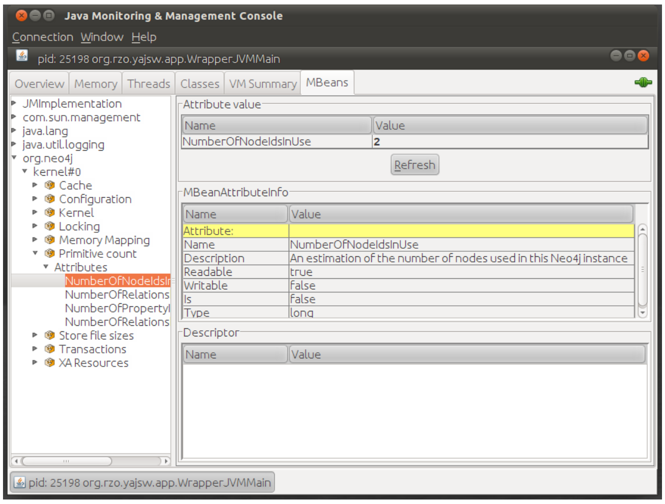
Usage: <hostname>:<port> OR service:jmx:<protocol>:<sap>

Username:

Password:

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.

Figure 10.2. Neo4j MBeans View



Name	Value
NumberOfNodeIdsInUse	2

Name	Value
Attribute:	
Name	NumberOfNodeIdsInUse
Description	An estimation of the number of nodes used in this Neo4j instance
Readable	true
Writable	false
Is	false
Type	long

Name	Value

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. For instance, the number of node IDs in use can be obtained with code like:

```
Neo4jManager manager = new Neo4jManager( graphDb.getManagementBean( Kernel.class ) );
long nodeIDsInUse    = manager.getPrimitivesBean.getNumberOfNodeIdsInUse();
```

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.

Reference of supported JMX MBeans

Table 10.1. MBeans exposed by the Neo4j Kernel

Name	**Description**
`org.neo4j:instance=kernel#0,name=Memory Mapping`	The status of Neo4j memory mapping
`org.neo4j:instance=kernel#0, name=Locking`	Information about the Neo4j lock status
`org.neo4j:instance=kernel#0, name=Transactions`	Information about the Neo4j transaction manager
`org.neo4j:instance=kernel#0,name=Cache`	Information about the caching in Neo4j
`org.neo4j:instance=kernel#0, name=Configuration`	The configuration parameters used to configure Neo4j
`org.neo4j:instance=kernel#0, name=Primitive count`	Estimates of the numbers of different kinds of Neo4j primitives
`org.neo4j:instance=kernel#0,name=XA Resources`	Information about the XA transaction manager
`org.neo4j:instance=kernel#0,name=Store file sizes`	Information about the sizes of the different parts of the Neo4j graph store
`org.neo4j:instance=kernel#0,name=Kernel`	Information about the Neo4j kernel

Name	**Description**
`org.neo4j:instance=kernel#0,name=High Availability`	Information an High Availability cluster, if enabled.

Table 10.2. MBean Memory Mapping

Attribute	**Description**	**Type**
`MemoryPools`	Get information about each pool of memory mapped regions from store files with memory mapping enabled	String

Table 10.3. MBean Locking

Attribute	**Description**	**Type**
`NumberOfAvertedDeadlocks`	The number of lock sequences that would have lead to a deadlock situation that Neo4j has detected and adverted (by throwing DeadlockDetectedException).	Integer

Table 10.4. MBean Transactions

Attribute	**Description**	**Type**
`NumberOfOpenTransactions`	The number of currently open transactions	Integer
`PeakNumberOfConcurrentTransactions`	The highest number of transactions ever opened concurrently	Integer
`NumberOfOpenedTransactions`	The total number started transactions	Integer
`NumberOfCommittedTransactions`	The total number of committed transactionss	Integer

Table 10.5. MBean Cache

Attribute	**Description**	**Type**
`CacheType`	The type of cache used by Neo4j	String
`NodeCacheSize`	The number of Nodes currently in cache	Integer
`RelationshipCacheSize`	The number of Relationships currently in cache	Integer
`clear()`	clear all caches	function, void

Table 10.6. MBean Configuration

Attribute	**Description**	**Type**
`store_dir`	Relative path for where the Neo4j storage directory is located	String
`rebuild_idgenerators_fast`	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

Attribute	**Description**	**Type**
`logical_log`	Relative path for where the Neo4j logical log is located	String
`neostore.propertystore.db.index.keys.mapped_memory`	The size to allocate for memory mapping the store for property key strings	String
`neostore.propertystore.db.strings.mapped_memory`	The size to allocate for memory mapping the string property store	String
`neostore.propertystore.db.arrays.mapped_memory`	The size to allocate for memory mapping the array property store	String
`neo_store`	Relative path for where the Neo4j storage information file is located	String
`neostore.relationshipstore.db.mapped_memory`	The size to allocate for memory mapping the relationship store	String
`neostore.propertystore.db.index.mapped_memory`	The size to allocate for memory mapping the store for property key indexes	String
`create`	Configuration attribute	String
`enable_remote_shell`	Enable a remote shell server which shell clients can log in to	String
`neostore.propertystore.db.mapped_memory`	The size to allocate for memory mapping the property value store	Integer
`neostore.nodestore.db.mapped_memory`	The size to allocate for memory mapping the node store	String
`dir`	Configuration attribute	String

Table 10.7. MBean Primitive count

Attribute	**Description**	**Type**
`NumberOfNodeIdsInUse`	An estimation of the number of nodes used in this Neo4j instance	Integer
`NumberOfRelationshipIdsInUse`	An estimation of the number of relationships used in this Neo4j instance	Integer
`NumberOfPropertyIdsInUse`	An estimation of the number of properties used in this Neo4j instance	Integer
`NumberOfRelationshipTypeIdsInUse`	The number of relationship types used in this Neo4j instance	Integer

Table 10.8. MBean XA Resources

Attribute	**Description**	**Type**
`XaResources`	Information about all XA resources managed by the transaction manager	String

Table 10.9. MBean Store file sizes

Attribute	Description	Type
`TotalStoreSize`	The total disk space used by this Neo4j instance, in bytes.	Integer
`LogicalLogSize`	The amount of disk space used by the current Neo4j logical log, in bytes.	Integer
`ArrayStoreSize`	The amount of disk space used to store array properties, in bytes.	Integer
`NodeStoreSize`	The amount of disk space used to store nodes, in bytes.	Integer
`PropertyStoreSize`	The amount of disk space used to store properties (excluding string values and array values), in bytes.	Integer
`RelationshipStoreSize`	The amount of disk space used to store relationships, in bytes.	Integer
`StringStoreSize`	The amount of disk space used to store string properties, in bytes.	Integer

Table 10.10. MBean Kernel

Attribute	Description	Type
`ReadOnly`	Whether this is a read only instance.	boolean
`MBeanQuery`	An ObjectName that can be used as a query for getting all management beans for this Neo4j instance.	String
`KernelStartTime`	The time from which this Neo4j instance was in operational mode	Date
`StoreCreationDate`	The time when this Neo4j graph store was created	Date
`StoreId`	An identifier that uniquely identifies this Neo4j graph store	String
`StoreLogVersion`	The current version of the Neo4j store logical log	String
`KernelVersion`	The version of Neo4j	String
`StoreDirectory`	The location where the Neo4j store is located	String

Table 10.11. MBean High Availability

Attribute	Description	Type
`MachineId`	The cluster machine id of this instance	String
`Master`	True, if this Neo4j instance is currently Master in the cluster	boolean
`ConnectedSlaves`	A list of conencted slaves in this cluster	String

Attribute	Description	Type
`InstancesInCluster`	Information about the other Neo4j instances in this HA cluster	String

Part II. Tutorials

Chapter 11. Graph Database Concepts

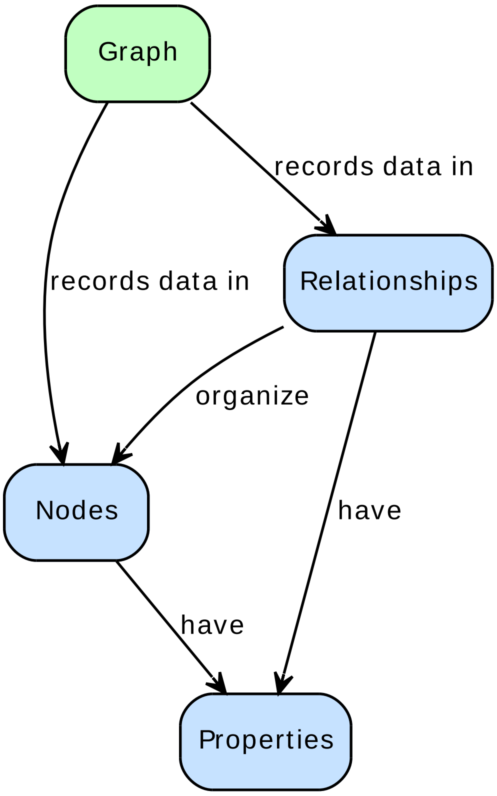
11.1. What is a Graph Database?

A graph database stores data in a graph, the most generic of data structures, capable of elegantly representing any kind of data in a highly accessible way. Let's follow along some graphs, using them to express graph concepts. We'll "read" a graph by following arrows around the diagram to form sentences.

11.1.1. A Graph contains Nodes and Relationships

> "A Graph —records data in–> Nodes —which have–> Properties"

The simplest possible graph is a single Node, a record that has named values referred to as Properties. A Node could start with a single Property and grow to a few million, though that can get a little awkward. At some point it makes sense to distribute the data into multiple nodes, organized with explicit Relationships.



11.1.2. Relationships organize the Graph

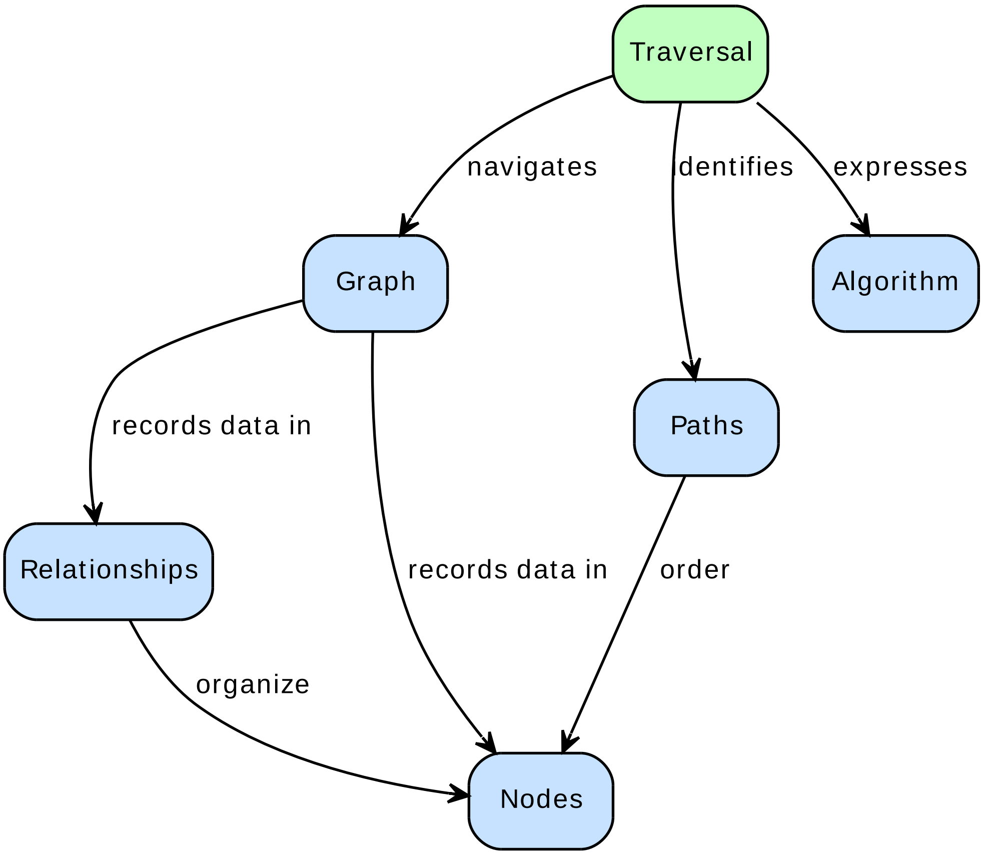
> "Nodes —are organized by–> Relationships —which also have–> Properties"

Relationships organize Nodes into arbitrary structures, allowing a Graph to resemble a List, a Tree, a Map, or a compound Entity – any of which can be combined into yet more complex, richly interconnected structures.

11.1.3. Query a Graph with a Traversal

> "A Traversal —navigates–> a Graph; it —identifies–> Paths —which order–> Nodes"

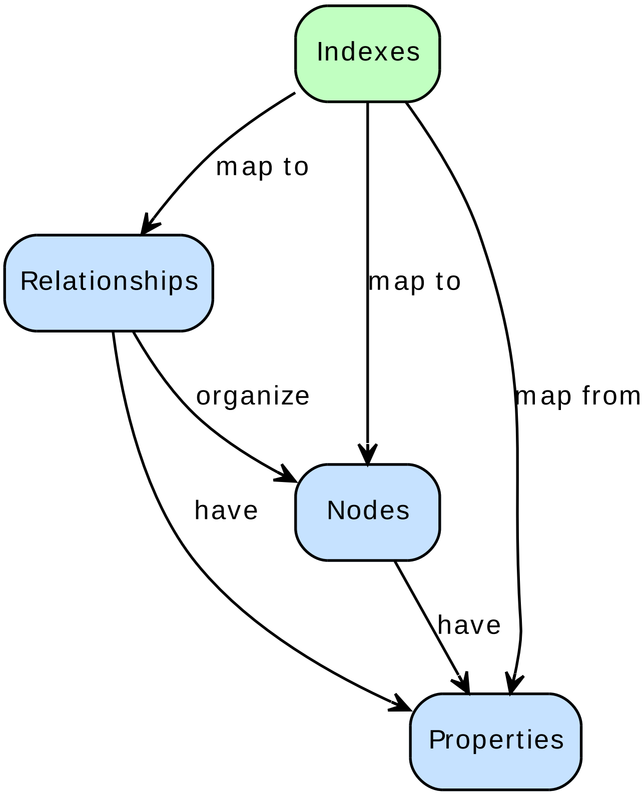
A Traversal is how you query a Graph, navigating from starting Nodes to related Nodes according to an algorithm, finding answers to questions like “what music do my friends like that I don’t yet own,” or “if this power supply goes down, what web services are affected?”



11.1.4. Indexes look-up Nodes or Relationships

> “An Index —maps from–> Properties —to either–> Nodes or Relationships”

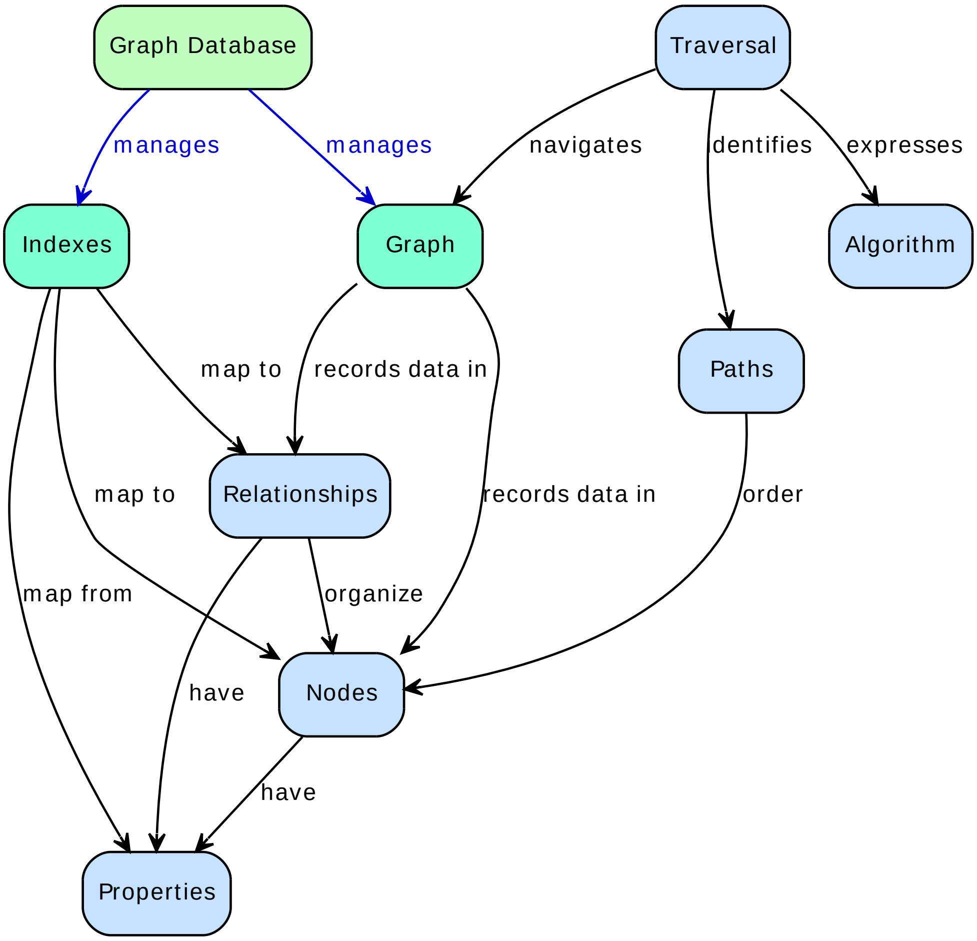
Often, you want to find a specific Node or Relationship according to a Property it has. Rather than traversing the entire graph, use an Index to perform a look-up, for questions like “find the Account for username master-of-graphs.”



11.1.5. Neo4j is a Graph Database

> “A Graph Database —manages a–> Graph and —also manages related–> Indexes”

Neo4j is a commercially supported open-source graph database. It was designed and built from the ground-up to be a reliable database, optimized for graph structures instead of tables. Working with Neo4j, your application gets all the expressiveness of a graph, with all the dependability you expect out of a database.


Graph Database
Traversal
manages
manages
navigates
dentifies
expresses
Indexes
Graph
Algorithm
map to
records data in
Paths
map to
Relationships
records data in
order
map from
organize
have
Nodes
have
Properties

11.2. Comparing Database Models

A Graph Database stores data structured in the Nodes and Relationships of a graph. How does this compare to other persistence models? Because a graph is a generic structure, let's compare how a few models would look in a graph.

11.2.1. A Graph Database transforms a RDBMS

Topple the stacks of records in a relational database while keeping all the relationships, and you'll see a graph. Where an RDBMS is optimized for aggregated data, Neo4j is optimized for highly connected data.

Figure 11.1. RDBMS

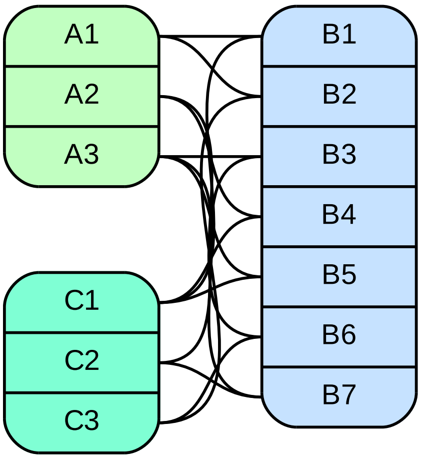
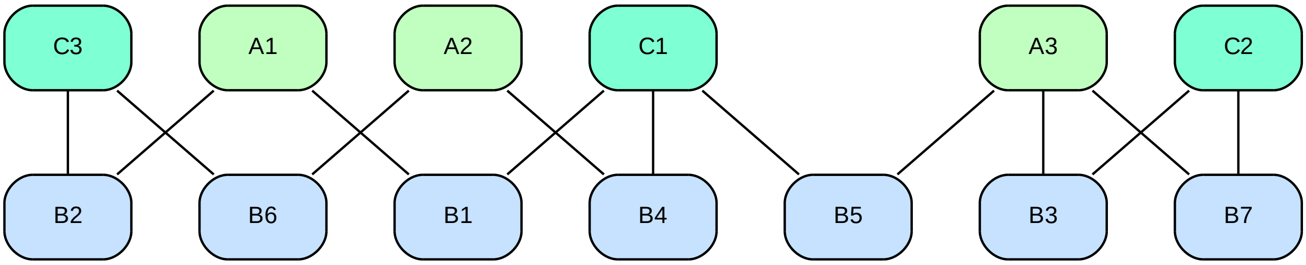


Figure 11.2. Graph Database as RDBMS



11.2.2. A Graph Database elaborates a Key-Value Store

A Key-Value model is great for lookups of simple values or lists. When the values are themselves interconnected, you've got a graph. Neo4j lets you elaborate the simple data structures into more complex, interconnected data.

Figure 11.3. Key-Value Store

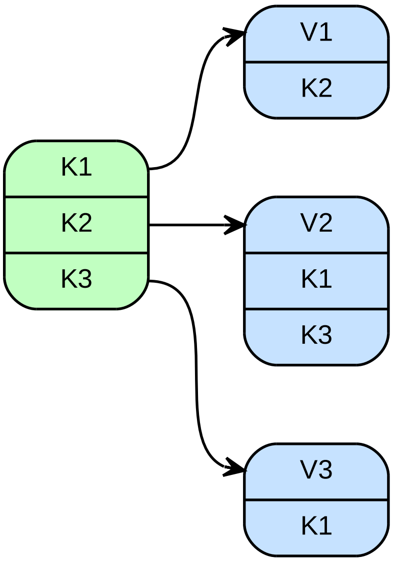
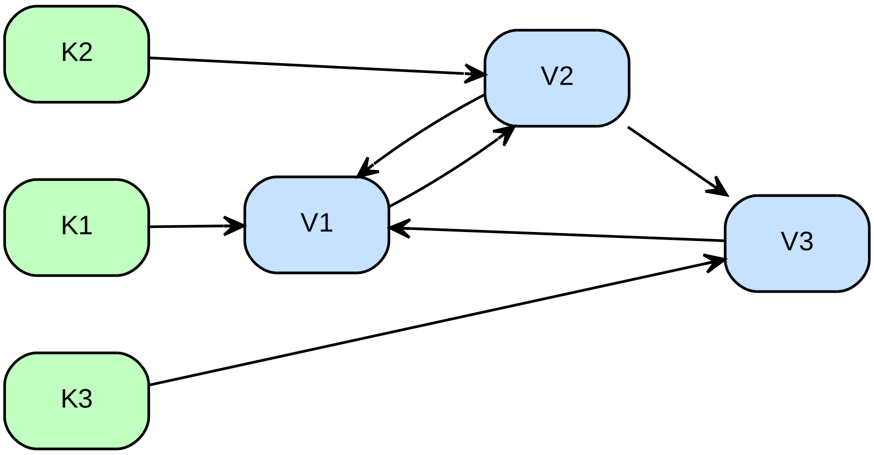


Figure 11.4. Graph Database as Key-Value Store



11.2.3. A Graph Database relates Column-Family

Column Family (BigTable-style) databases are an evolution of key-value, using "families" to allow grouping of rows. Stored in a graph, the families could become hierarchical, and the relationships among data becomes explicit.

11.2.4. A Graph Database navigates a Document Store

The container hierarchy of a document database accommodates nice, schema-free data that can easily be represented as a tree. Which is of course a graph. Refer to other documents (or document elements) within that tree and you have a more expressive representation of the same data. When in Neo4j, those relationships are easily navigable.

Figure 11.5. Document Store

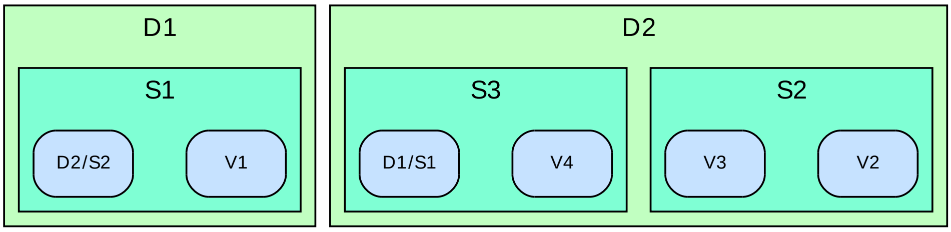
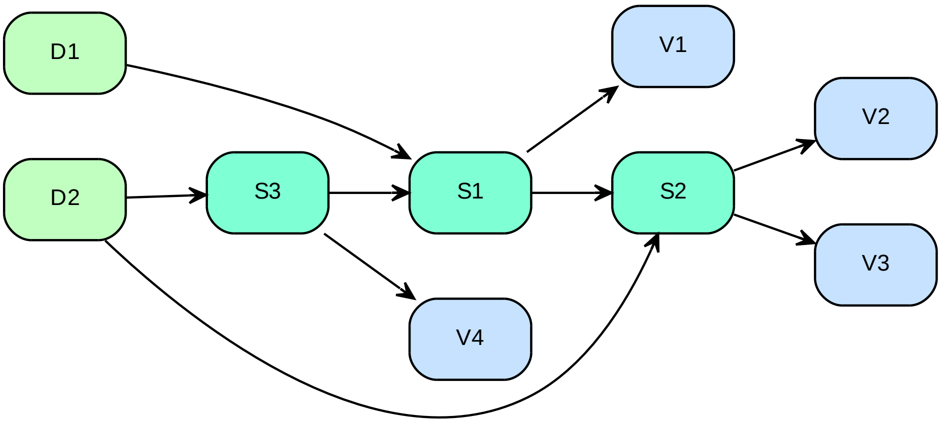


Figure 11.6. Graph Database as Document Store

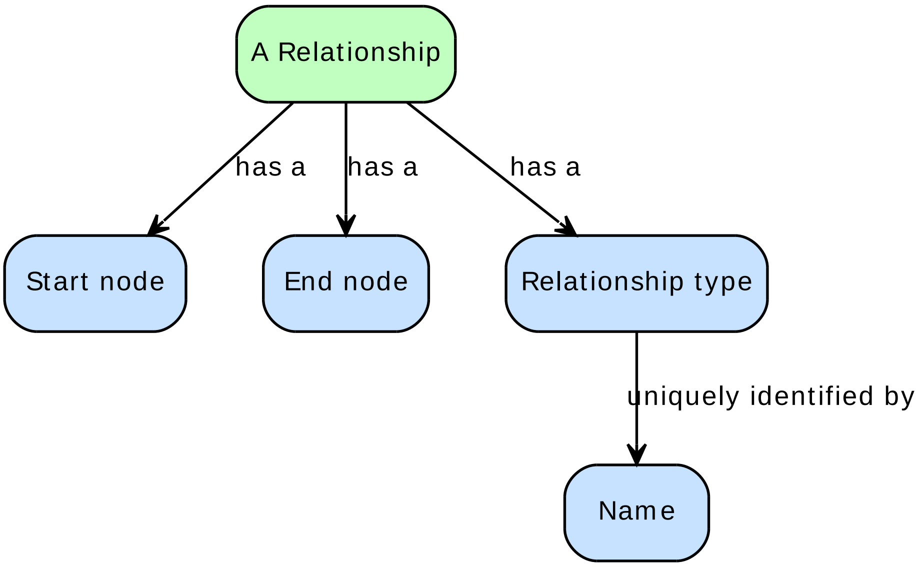


11.3. The Neo4j Graph Database

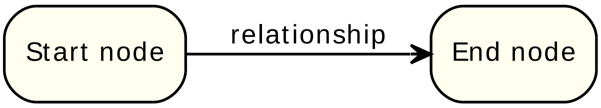
This chapter will introduce more details on the data model and behavior of Neo4j.

11.3.1. Relationships

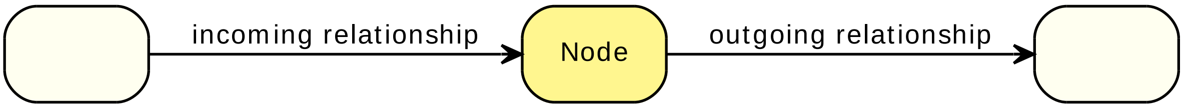
Relationships between nodes are a key part of a graph database. They allow for finding related data.



A relationship connects two nodes, and is guaranteed to have valid start and end nodes.



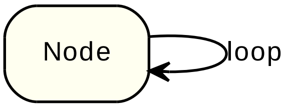
As relationships are always directed, they can be viewed as outgoing or incoming relative to a node, which is useful when traversing the graph:



Relationships are equally well traversed in either direction. This means that there is no need to add duplicate relationships in the opposite direction (with regard to traversal or performance).

While relationships always have a direction, you can ignore the direction where it is not useful in your application.

Note that a node can have relationships to itself as well:



To further enhance graph traversal all relationships have a relationship type. The following example shows a simple social network with two relationship types.

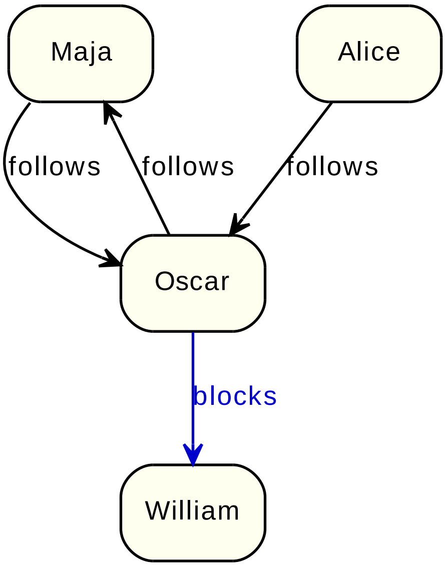
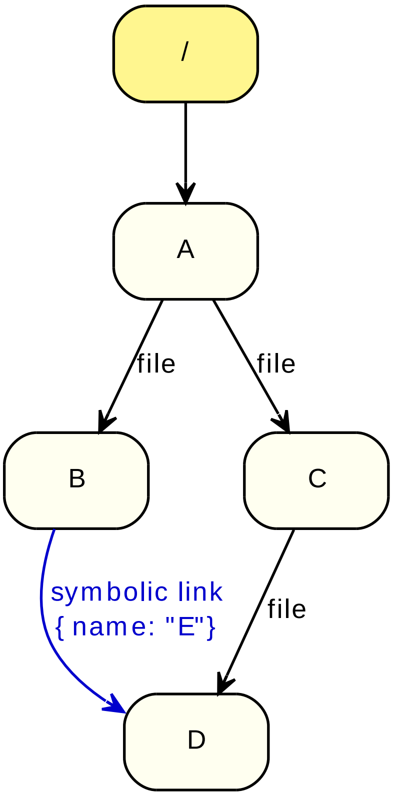


Table 11.1. Using relationship direction and type

What	How
get who a person follows	outgoing `follows` relationships, depth one
get the followers of a person	incoming `follows` relationships, depth one
get who a person blocks	outgoing `blocks` relationships, depth one
get who a person is blocked by	incoming `blocks` relationships, depth one

This example is a simple model of a file system, which includes symbolic links:



Depending on what you are looking for, you will use the direction and type of relationships during traversal.

What	How
get the full path of a file	incoming `file` relationships
get all paths for a file	incoming `file` and `symbolic link` relationships
get all files in a directory	outgoing `file` and `symbolic link` relationships, stop at depth one
get all files in a directory, excluding symbolic links	outgoing `file` relationships, stop at depth one
get all files in a directory, recursively	outgoing `file` and `symbolic link` relationships

11.3.2. Properties

Properties are key-value pairs where the key 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.

Note

`null` is not a valid property value. Nulls can instead be modeled by the absence of a key.

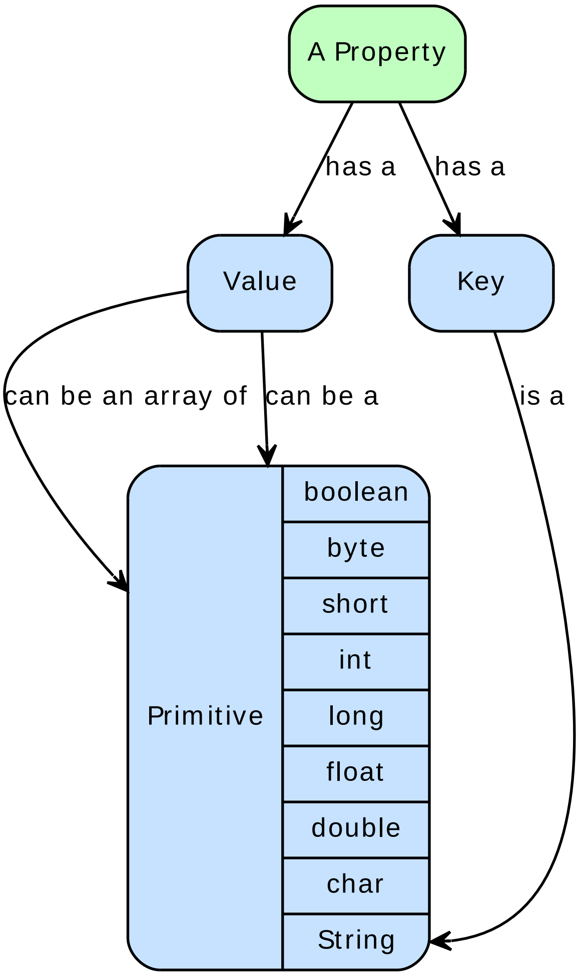


Table 11.2. Property value types

Type	Description	Value range
`boolean`		`true`/`false`

Type	**Description**	**Value range**
`byte`	8-bit integer	`-128` to `127`, inclusive
`short`	16-bit integer	`-32768` to `32767`, inclusive
`int`	32-bit integer	`-2147483648` to `2147483647`, inclusive
`long`	64-bit integer	`-9223372036854775808` to `9223372036854775807`, inclusive
`float`	32-bit IEEE 754 floating-point number	
`double`	64-bit IEEE 754 floating-point number	
`char`	16-bit unsigned integers representing Unicode characters	`u0000` to `uffff` (`0` to `65535`)
`String`	sequence of Unicode characters	

For further details on float/double values, see Java Language Specification <http://java.sun.com/docs/books/jls/third_edition/html/typesValues.html#4.2.3>.

Chapter 12. Using Neo4j embedded in Java applications

12.1. 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.

12.1.1. Add Neo4j to the build path

Get the Neo4j libraries from one of these sources:

- Extract a Neo4j download <http://neo4j.org/download/> zip/tarball, 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/helios/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/idea/webhelp/libraries-global-libraries-and-the-configure-library-dialog.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 proejct node, see Managing a Project's Classpath <http://netbeans.org/kb/docs/java/project-setup.html#projects-classpath>.

12.1.2. Add Neo4j as a dependency

For an overview of the main Neo4j artifacts, see Table 1.2, "Neo4j editions". The artifacts listed there are top-level artifacts that will transitively include the actual Neo4j implementation. You can either go with the top-level artifact or include the individual components directly. The examples included here use the top-level artifact approach.

Maven

Maven dependency.

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

```
...
</project>
```

Where `${neo4j-version}` is the intended version and the `artifactId` is found in Table 1.2, “Neo4j editions”.

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="${neo4j-version}"/>
  ..
</dependencies>
..
```

Where `${neo4j-version}` is the intended version and the `name` is found in Table 1.2, “Neo4j editions”.

12.1.3. Starting and stopping

To create a new database or ópen an existing one you instantiate an `EmbeddedGraphDatabase` `<http://components.neo4j.org/neo4j/1.4.2/apidocs/org/neo4j/kernel/EmbeddedGraphDatabase.html>`.

```
GraphDatabaseService graphDb = new EmbeddedGraphDatabase( DB_PATH );
registerShutdownHook( graphDb );
```

Note

The `EmbeddedGraphDatabase` instance can be shared among multiple threads. Note however that you can’t create multiple instances poting 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 example before it's completed)
    Runtime.getRuntime().addShutdownHook( new Thread()
    {
        @Override
        public void run()
        {
            graphDb.shutdown();
        }
    } );
}
```

If you want a *read-only view* of the database, use `EmbeddedReadOnlyGraphDatabase` `<http://components.neo4j.org/neo4j/1.4.2/apidocs/org/neo4j/kernel/EmbeddedReadOnlyGraphDatabase.html>`.

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

```
Map<String, String> config = EmbeddedGraphDatabase.loadConfigurations( pathToConfig
                                                                       + "neo4j.properties" );
GraphDatabaseService graphDb = new EmbeddedGraphDatabase(
        "target/database/location", config );
```

Or you could of course create you own `Map<String, String>` programatically and use that instead.

For configuration settings, see Chapter 2, *Configuration & Performance*.

12.2. Hello world

Create and access nodes and relationships.

To begin with, we define the relationship types we want to use:

```
private static enum ExampleRelationshipTypes implements RelationshipType
{
    EXAMPLE
}
```

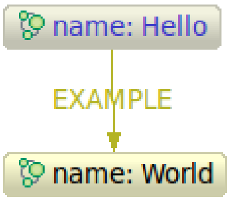
The next step is to start the database server:

```
GraphDatabaseService graphDb = new EmbeddedGraphDatabase( DB_PATH );
registerShutdownHook( graphDb );
```

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:

```
// Encapsulate operations in a transaction
Transaction tx = graphDb.beginTx();
try
{
    Node firstNode = graphDb.createNode();
    firstNode.setProperty( NAME_KEY, "Hello" );
    Node secondNode = graphDb.createNode();
    secondNode.setProperty( NAME_KEY, "World" );

    firstNode.createRelationshipTo( secondNode,
        ExampleRelationshipTypes.EXAMPLE );

    String greeting = firstNode.getProperty( NAME_KEY ) + " "
        + secondNode.getProperty( NAME_KEY );
    System.out.println( greeting );
```

At this point this is how the database looks:



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

```
    // let's remove the data before committing
    firstNode.getSingleRelationship( ExampleRelationshipTypes.EXAMPLE,
            Direction.OUTGOING ).delete();
    firstNode.delete();
    secondNode.delete();

    tx.success();
}
finally
{
    tx.finish();
}
```

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

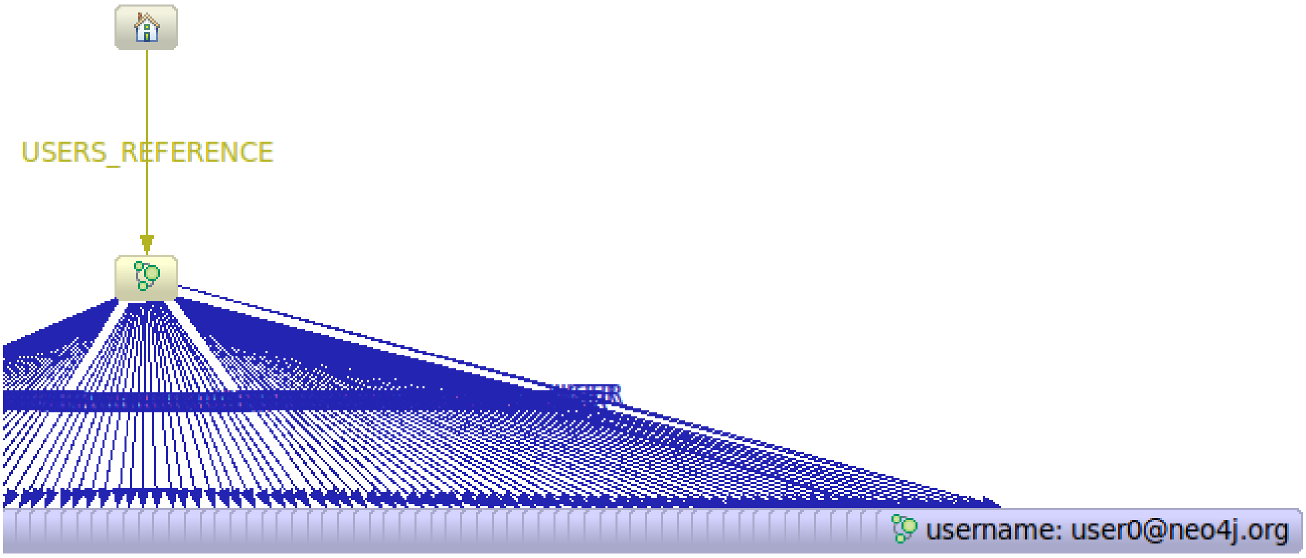
```
graphDb.shutdown();
```

Full source code: EmbeddedNeo4j.java <https://github.com/neo4j/community/blob/1.4.2/embedded-examples/src/main/java/org/neo4j/examples/EmbeddedNeo4j.java>

12.3. User database with index

You have a user database, and want to retrieve users by name. To begin with, this is the structure of the database we want to create:

Figure 12.1. Node space view of users



That is, the reference node is connected to a users-reference node to which all users are connected.

To begin with, we define the relationship types we want to use:

```
private static enum RelTypes implements RelationshipType
{
    USERS_REFERENCE,
    USER
}
```

Then 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 EmbeddedGraphDatabase( DB_PATH );
nodeIndex = graphDb.index().forNodes( "nodes" );
registerShutdownHook();
```

It’s time to add the users:

```
Transaction tx = graphDb.beginTx();
try
{
    // Create users sub reference node (see design guidelines on
    // http://wiki.neo4j.org/ )
    Node usersReferenceNode = graphDb.createNode();
    graphDb.getReferenceNode().createRelationshipTo(
        usersReferenceNode, RelTypes.USERS_REFERENCE );
    // Create some users and index their names with the IndexService
    for ( int id = 0; id < 100; id++ )
    {
        Node userNode = createAndIndexUser( idToUserName( id ) );
        usersReferenceNode.createRelationshipTo( userNode,
            RelTypes.USER );
    }
```

And here’s how to find a user by Id:

```
int idToFind = 45;
Node foundUser = nodeIndex.get( USERNAME_KEY,
    idToUserName( idToFind ) ).getSingle();
System.out.println( "The username of user " + idToFind + " is "
    + foundUser.getProperty( USERNAME_KEY ) );
```

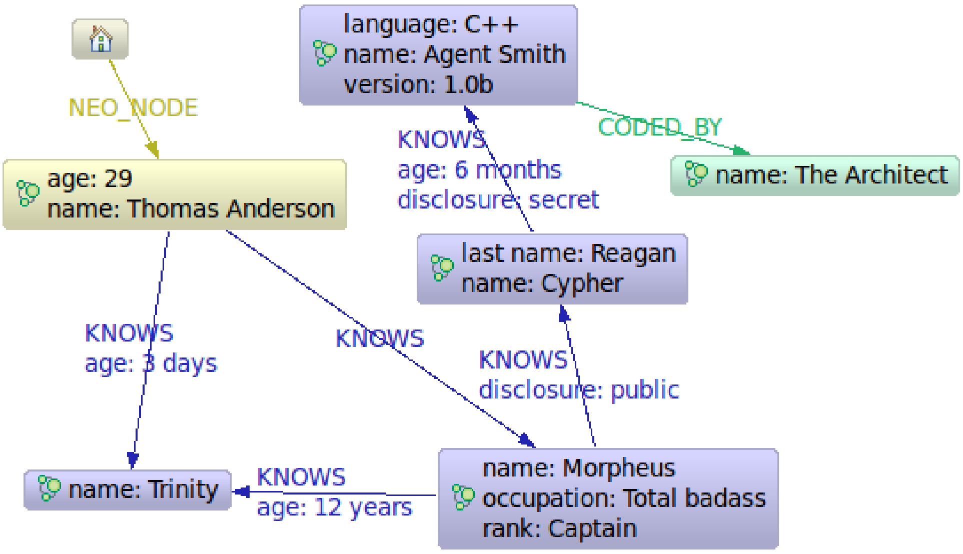
Full source code: EmbeddedNeo4jWithIndexing.java <https://github.com/neo4j/community/blob/1.4.2/embedded-examples/src/main/java/org/neo4j/examples/EmbeddedNeo4jWithIndexing.java>

12.4. Traversal

12.4.1. The Matrix

This is the first node space we want to traverse into:

Figure 12.2. Matrix node space view



Friends and friends of friends.

```
private static Traverser getFriends( final Node person )
{
    return person.traverse( Order.BREADTH_FIRST,
            StopEvaluator.END_OF_GRAPH,
            ReturnableEvaluator.ALL_BUT_START_NODE, RelTypes.KNOWS,
            Direction.OUTGOING );
}
```

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

```
Traverser friendsTraverser = getFriends( neoNode );
int numberOfFriends = 0;
for ( Node friendNode : friendsTraverser )
{
    System.out.println( "At depth "
            + friendsTraverser.currentPosition()
            .depth() + " => "
            + friendNode.getProperty( "name" ) );
}
```

Who coded the Matrix?

```
private static Traverser findHackers( final Node startNode )
{
    return startNode.traverse( Order.BREADTH_FIRST,
            StopEvaluator.END_OF_GRAPH, new ReturnableEvaluator()
    {
        @Override
        public boolean isReturnableNode(
                final TraversalPosition currentPos )
        {
            return !currentPos.isStartNode()
            && currentPos.lastRelationshipTraversed()
            .isType( RelTypes.CODED_BY );
        }
    }, RelTypes.CODED_BY, Direction.OUTGOING, RelTypes.KNOWS,
    Direction.OUTGOING );
}
```

Print out the result:

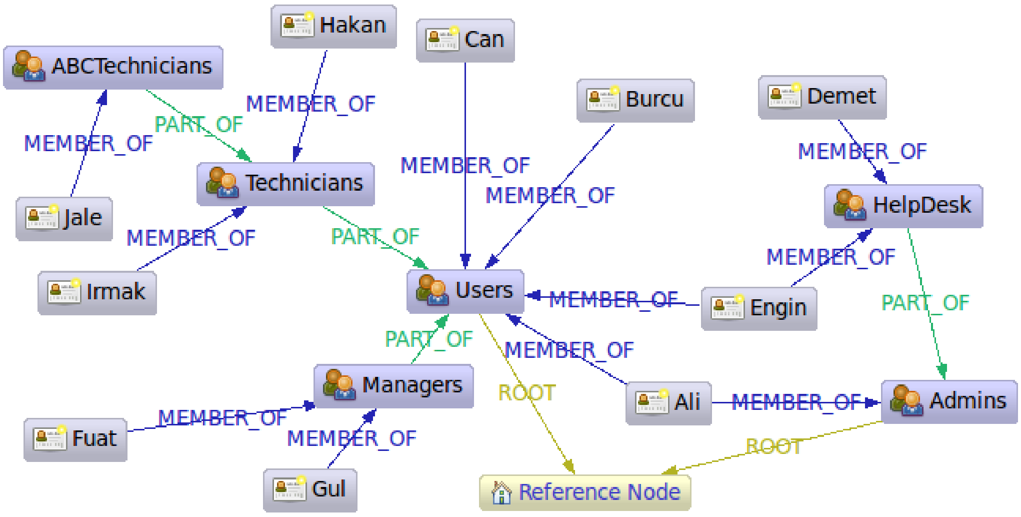
```
Traverser traverser = findHackers( getNeoNode() );
int numberOfHackers = 0;
for ( Node hackerNode : traverser )
{
    System.out.println( "At depth " + traverser.currentPosition()
            .depth() + " => " + hackerNode.getProperty( "name" ) );
}
```

Full source code: MatrixTest.java <https://github.com/neo4j/community/blob/1.4.2/embedded-examples/src/test/java/org/neo4j/examples/MatrixTest.java>

12.4.2. User roles

Here we have users assigned to groups, and groups containing other groups. This is the full node space of our example:

Figure 12.3. User roles node space view



Get the admins.

```
Node admins = getGroupByName( "Admins" );
Traverser traverser = admins.traverse(
        Traverser.Order.BREADTH_FIRST, StopEvaluator.END_OF_GRAPH,
        ReturnableEvaluator.ALL_BUT_START_NODE, RoleRels.PART_OF,
        Direction.INCOMING, RoleRels.MEMBER_OF, Direction.INCOMING );
for ( Node part : traverser )
{
    System.out.println( part.getProperty( NAME )
                        + " "
                        + ( traverser.currentPosition().depth() - 1 ) );
}
```

Get the group memberships of a user.

```
Node jale = getUserByName( "Jale" );
Traverser traverser = jale.traverse( Traverser.Order.DEPTH_FIRST,
        StopEvaluator.END_OF_GRAPH,
        ReturnableEvaluator.ALL_BUT_START_NODE, RoleRels.MEMBER_OF,
        Direction.OUTGOING, RoleRels.PART_OF, Direction.OUTGOING );
for ( Node membership : traverser )
{
    System.out.println( membership.getProperty( NAME )
                        + " "
                        + ( traverser.currentPosition().depth() - 1 ) );
}
```

Get all groups.

```
Node referenceNode = graphDb.getReferenceNode();
Traverser traverser = referenceNode.traverse(
        Traverser.Order.BREADTH_FIRST, StopEvaluator.END_OF_GRAPH,
        ReturnableEvaluator.ALL_BUT_START_NODE, RoleRels.ROOT,
        Direction.INCOMING, RoleRels.PART_OF, Direction.INCOMING );
for ( Node group : traverser )
{
    System.out.println( group.getProperty( NAME ) );
}
```

Get all members of all groups.

```
Node referenceNode = graphDb.getReferenceNode();
Traverser traverser = referenceNode.traverse(
        Traverser.Order.BREADTH_FIRST, StopEvaluator.END_OF_GRAPH,
        new ReturnableEvaluator()
        {
            @Override
            public boolean isReturnableNode(
                    TraversalPosition currentPos )
            {
                if ( currentPos.isStartNode() )
                {
                    return false;
                }
                Relationship rel = currentPos.lastRelationshipTraversed();
                return rel.isType( RoleRels.MEMBER_OF );
            }
        }, RoleRels.ROOT,
        Direction.INCOMING, RoleRels.PART_OF, Direction.INCOMING,
        RoleRels.MEMBER_OF, Direction.INCOMING );
for ( Node group : traverser )
{
    System.out.println( group.getProperty( NAME ) );
}
```

Full source code: RolesTest.java <https://github.com/neo4j/community/blob/1.4.2/embedded-examples/src/test/java/org/neo4j/examples/RolesTest.java>

12.4.3. New traversal framework

Note

The following examples use a new experimental traversal API. It shares the underlying implementation with the old traversal API, so performance-wise they should be equal. However, expect the new API to evolve and thus undergo changes.

The Matrix

The traversals from the Matrix example above, this time using the new traversal API:

Friends and friends of friends.

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

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

```
Traverser friendsTraverser = getFriends( neoNode );
int numberOfFriends = 0;
for ( Path friendPath : friendsTraverser )
{
    System.out.println( "At depth " + friendPath.length() + " => "
                        + friendPath.endNode()
                                .getProperty( "name" ) );
}
```

Who coded the Matrix?

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

Print out the result:

```
Traverser traverser = findHackers( getNeoNode() );
int numberOfHackers = 0;
for ( Path hackerPath : traverser )
{
    System.out.println( "At depth " + hackerPath.length() + " => "
                        + hackerPath.endNode()
                                .getProperty( "name" ) );
}
```

Full source code: MatrixNewTravTest.java <https://github.com/neo4j/community/blob/1.4.2/embedded-examples/src/test/java/org/neo4j/examples/MatrixNewTravTest.java>

User roles

The traversals from the User roles example above, this time using the new traversal API:

Get the admins.

```
Node admins = getGroupByName( "Admins" );
TraversalDescription td = Traversal.description()
        .breadthFirst()
        .relationships( RoleRels.PART_OF, Direction.INCOMING )
        .relationships( RoleRels.MEMBER_OF, Direction.INCOMING )
        .evaluator( Evaluators.excludeStartPosition() );
for ( Path path : td.traverse( admins ) )
{
    Node part = path.endNode();
    System.out.println( part.getProperty( NAME ) + " "
                        + ( path.length() - 1 ) );
}
```

Get the group memberships of a user.

```
Node jale = getUserByName( "Jale" );
TraversalDescription td = Traversal.description()
        .depthFirst()
        .relationships( RoleRels.MEMBER_OF, Direction.OUTGOING )
        .relationships( RoleRels.PART_OF, Direction.OUTGOING )
        .evaluator( Evaluators.excludeStartPosition() );
for ( Path path : td.traverse( jale ) )
{
    Node membership = path.endNode();
    System.out.println( membership.getProperty( NAME ) + " "
                        + ( path.length() - 1 ) );
}
```

Get all groups.

```
Node referenceNode = graphDb.getReferenceNode();
TraversalDescription td = Traversal.description()
        .breadthFirst()
        .relationships( RoleRels.ROOT, Direction.INCOMING )
        .relationships( RoleRels.PART_OF, Direction.INCOMING )
        .evaluator( Evaluators.excludeStartPosition() );
for ( Node group : td.traverse( referenceNode )
        .nodes() )
{
    System.out.println( group.getProperty( NAME ) );
}
```

Get all members of all groups.

```
Node referenceNode = graphDb.getReferenceNode();
TraversalDescription td = Traversal.description()
        .breadthFirst()
        .relationships( RoleRels.ROOT, Direction.INCOMING )
        .relationships( RoleRels.MEMBER_OF, Direction.INCOMING )
        .relationships( RoleRels.PART_OF, Direction.INCOMING )
        .evaluator(
                Evaluators.pruneWhereLastRelationshipTypeIs( RoleRels.MEMBER_OF ) );
for ( Node group : td.traverse( referenceNode ).nodes() )
{
    System.out.println( group.getProperty( NAME ) );
}
```

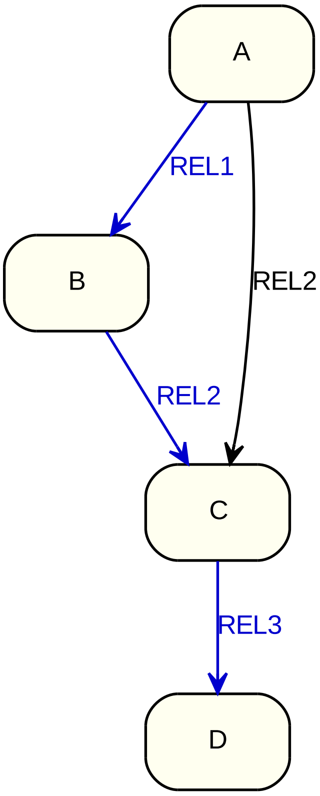
Full source code: RolesNewTravTest.java <https://github.com/neo4j/community/blob/1.4.2/embedded-examples/src/test/java/org/neo4j/examples/RolesNewTravTest.java>

Walking an ordered path

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

Create a toy graph.

```
Node A = db.createNode();
Node B = db.createNode();
Node C = db.createNode();
Node D = db.createNode();
A.createRelationshipTo( B, REL1 );
B.createRelationshipTo( C, REL2 );
C.createRelationshipTo( D, REL3 );
A.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:

Walk the path.

```
final ArrayList<RelationshipType> orderedPathContext = new ArrayList<RelationshipType>();
orderedPathContext.add( REL1 );
orderedPathContext.add( withName( "REL2" ) );
orderedPathContext.add( withName( "REL3" ) );
TraversalDescription td = Traversal.description().evaluator(
        new Evaluator()
        {
            @Override
            public Evaluation evaluate( final Path path )
            {
                if ( path.length() == 0 )
                {
                    return Evaluation.EXCLUDE_AND_CONTINUE;
                }
                String currentName = path.lastRelationship().getType().name();
                String relationshipType = orderedPathContext.get(
                        path.length() - 1 ).name();
                if ( path.length() == orderedPathContext.size() )
                {
                    if ( currentName.equals( relationshipType ) )
                    {
                        return Evaluation.INCLUDE_AND_PRUNE;
                    }
                    else
                    {
                        return Evaluation.EXCLUDE_AND_PRUNE;
                    }
                }
                else
                {
                    if ( currentName.equals( relationshipType ) )
                    {
                        return Evaluation.EXCLUDE_AND_CONTINUE;
                    }
                    else
                    {
                        return Evaluation.EXCLUDE_AND_PRUNE;
                    }
                }
            }
        } );
Traverser t = td.traverse( db.getNodeById( 1 ) );
for ( Path path : t )
{
    System.out.println( path );
}
```

Full source code: OrderedPathTest.java <https://github.com/neo4j/community/blob/1.4.2/embedded-examples/src/test/java/org/neo4j/examples/orderedpath/OrderedPathTest.java>

12.4.4. Social network

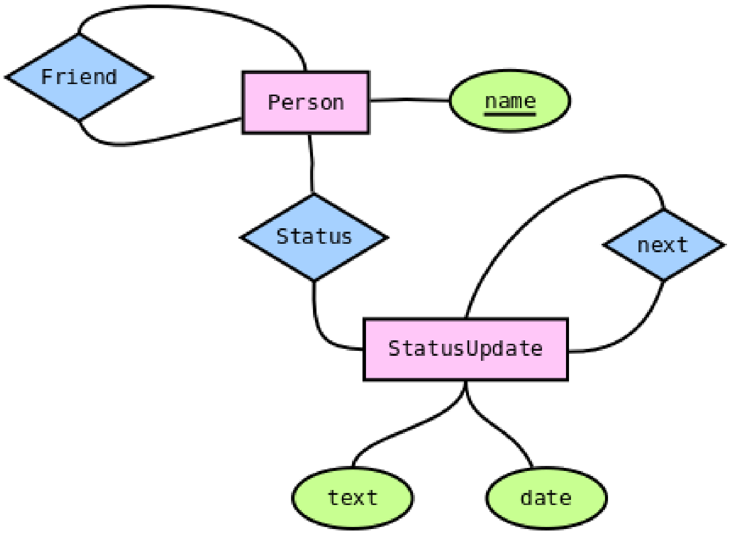
Note

The following example uses the new experimental 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.

Simple social model

Figure 12.4. 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.

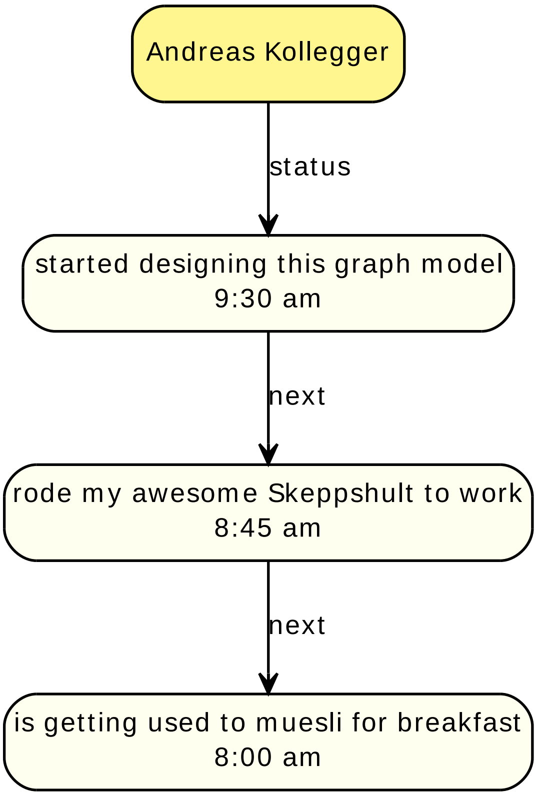
- `Person`
  - `friend`: relates two distinct `Person` instances (no self-reference)
  - `status`: connects to the most recent `StatusUpdate`

- `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:


Andreas Kollegger
status
started designing this graph model
9:30 am
next
rode my awesome Skeppshult to work
8:45 am
next
is getting used to muesli for breakfast
8:00 am

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

```
TraversalDescription traversal = Traversal.description().
        depthFirst().
        relationships( NEXT ).
        filter( Traversal.returnAll() );
```

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;
```

Full source code: socnet <https://github.com/neo4j/community/tree/1.4.2/embedded-examples/src/main/java/org/neo4j/examples/socnet>

12.5. 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).

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() + "]";
}
```

Full source code: Person.java <https://github.com/neo4j/community/blob/1.4.2/embedded-examples/src/main/java/org/neo4j/examples/socnet/Person.java>

12.6. Graph Algorithm examples

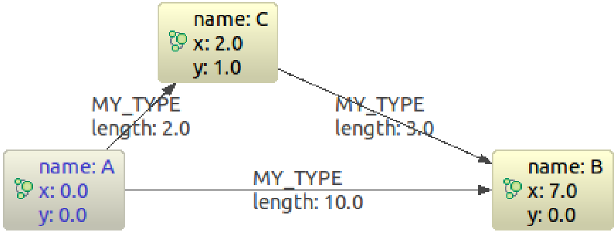
Calculating the shortest path (least number of relationships) between two nodes:

```
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(
        Traversal.expanderForTypes( ExampleTypes.MY_TYPE, Direction.OUTGOING ), 15 );
Iterable<Path> paths = finder.findAllPaths( startNode, endNode );
```

Using Dijkstra’s algorithm <http://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(
        Traversal.expanderForTypes( ExampleTypes.MY_TYPE, Direction.BOTH ), "cost" );

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

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

Using A* <http://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>()
{
    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(
        Traversal.expanderForAllTypes(),
        CommonEvaluators.doubleCostEvaluator( "length" ), estimateEvaluator );
WeightedPath path = astar.findSinglePath( nodeA, nodeB );
```

Full source code: PathFindingExamplesTest.java <https://github.com/neo4j/community/blob/1.4.2/embedded-examples/src/test/java/org/neo4j/examples/PathFindingExamplesTest.java>

12.7. Reading a management attribute

Reading a management attribute

The EmbeddedGraphDatabase <http://components.neo4j.org/neo4j/1.4.2/apidocs/org/neo4j/kernel/EmbeddedGraphDatabase.html> class includes a convenience method <http://components.neo4j.org/neo4j/1.4.2/apidocs/org/neo4j/kernel/EmbeddedGraphDatabase.html#getManagementBean%28java.lang.Class%29> to get instances of 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.

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

```
private static Date getStartTimeFromManagementBean(
        GraphDatabaseService graphDbService )
{
    // use EmbeddedGraphDatabase to access management beans
    EmbeddedGraphDatabase graphDb = (EmbeddedGraphDatabase) graphDbService;
    Kernel kernel = graphDb.getManagementBean( Kernel.class );
    Date startTime = kernel.getKernelStartTime();
    return startTime;
}
```

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

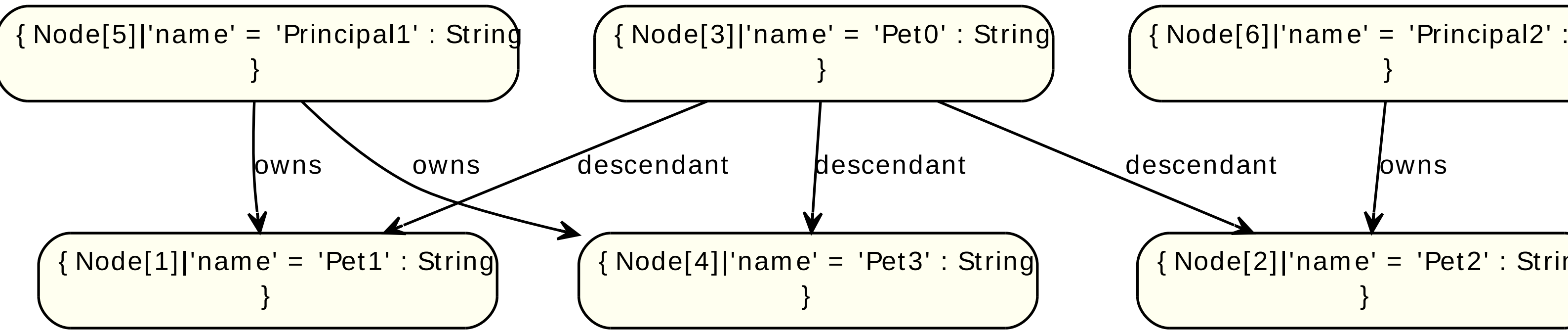
- For all editions, see the org.neo4j.jmx <http://components.neo4j.org/neo4j-jmx/1.4.2/apidocs/org/neo4j/jmx/package-summary.html> package.
- For the Advanced and Enterprise editions, see the org.neo4j.management <http://components.neo4j.org/neo4j-management/1.4.2/apidocs/org/neo4j/management/package-summary.html> package as well.

Full source code: JmxTest.java <https://github.com/neo4j/community/blob/1.4.2/embedded-examples/src/test/java/org/neo4j/examples/JmxTest.java>

12.8. 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.

descendants1


{ Node[5]|'name' = 'Principal1' : String
}
{ Node[3]|'name' = 'Pet0' : String
}
{ Node[6]|'name' = 'Principal2' :
}
owns
owns
descendant
descendant
descendant
owns
{ Node[1]|'name' = 'Pet1' : String
}
{ Node[4]|'name' = 'Pet3' : String
}
{ Node[2]|'name' = 'Pet2' : Strin
}

In order to return which 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` so that 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 = Traversal.description().uniqueness(Uniqueness.NODE_PATH ).evaluator( new Evaluator()
{
    @Override
    public Evaluation evaluate( Path path )
    {
        if(path.endNode().equals( target )) {
            return Evaluation.INCLUDE_AND_PRUNE;
        }
        return Evaluation.EXCLUDE_AND_CONTINUE;
    }
} );

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

This will return the following paths:

```
(3)--[descendant,0]-->(1)<--[owns,3]--(5)
(3)--[descendant,2]-->(4)<--[owns,5]--(5)
```

Part III. Tools

Chapter 13. Web Administration

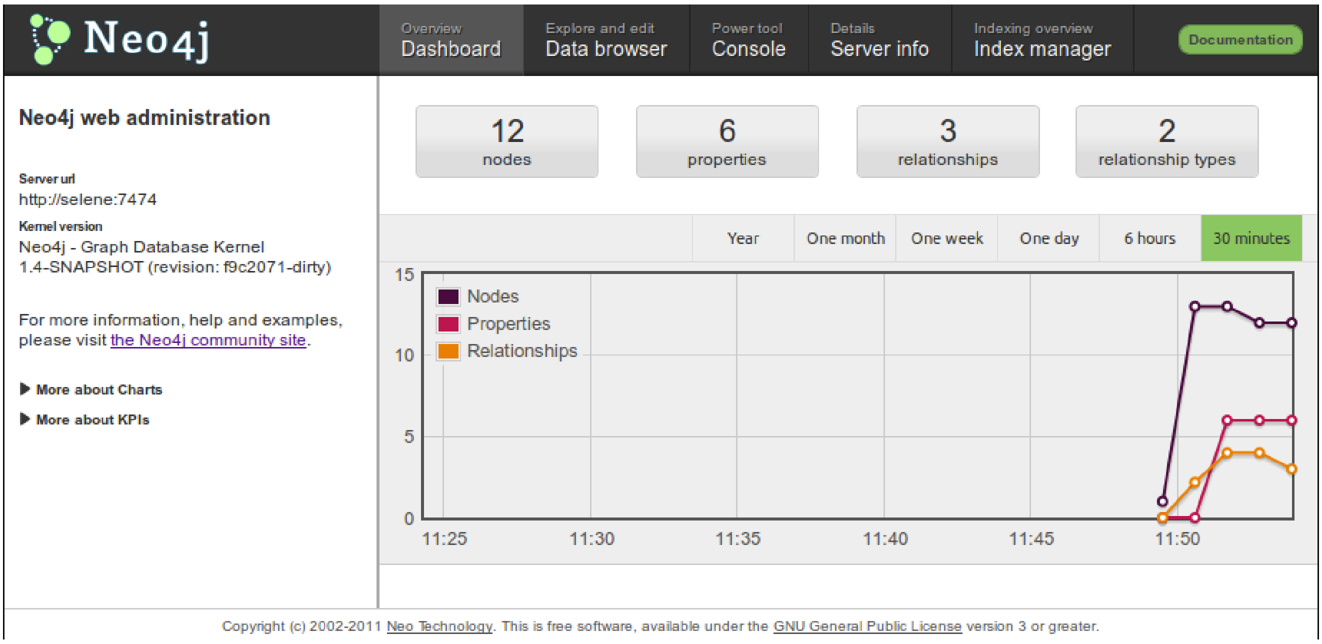
The Neo4j Web Administration is the primary user interface for Neo4j. With it, you can:

- monitor the Neo4j Server
- manipulate and browse data
- interact with the database via various consoles
- view raw data management objects (JMX MBeans)

13.1. Dashboard tab

The Dashboard tab provides an overview of a running Neo4j instance.

Figure 13.1. Web Administration Dashboard



Neo4j web administration

Server url
http://selene:7474

Kernel version
Neo4j - Graph Database Kernel
1.4-SNAPSHOT (revision: f9c2071-dirty)

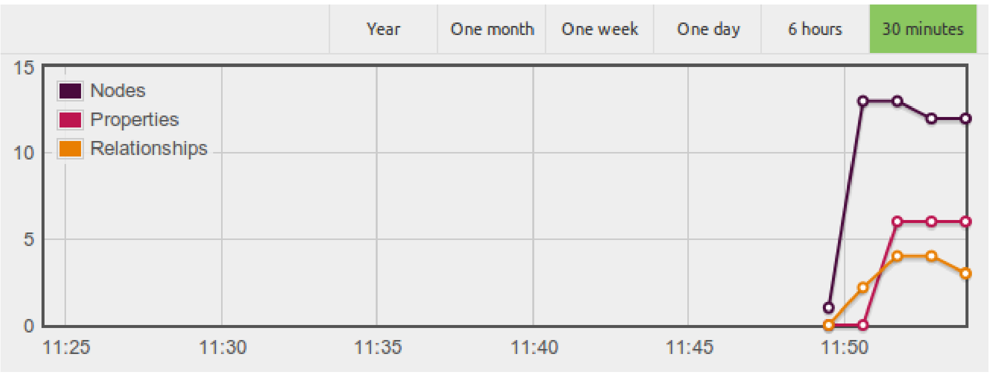
For more information, help and examples, please visit the Neo4j community site.

Copyright (c) 2002-2011 Neo Technology. This is free software, available under the GNU General Public License version 3 or greater.

13.1.1. Entity chart

The charts show entity counts over time: node, relationship and properties.

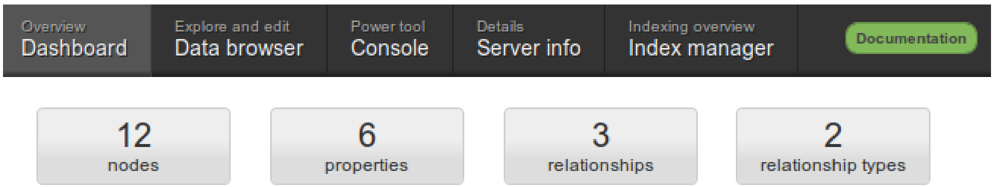
Figure 13.2. Entity charting



13.1.2. Status monitoring

Below the entity chart is a collection of status panels, displaying current resource usage.

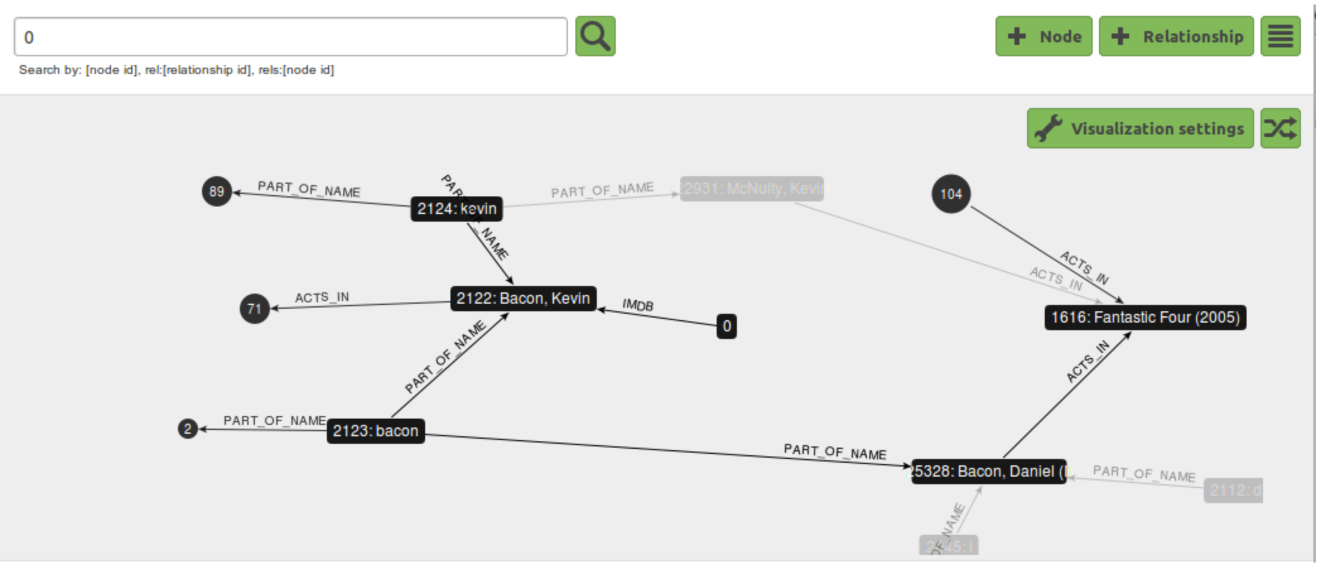
Figure 13.3. Status indicator panels



13.2. Data tab

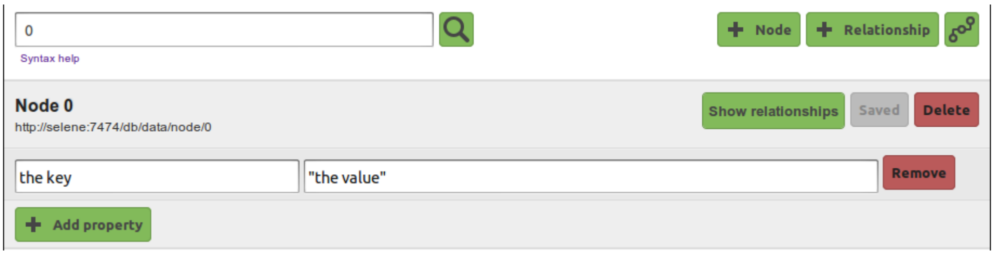
Use the Data tab to browse, add or modify nodes, relationships and their properties.

Figure 13.4. Browsing and manipulating data



0

Figure 13.5. Editing properties



0

Node 0

http://selene:7474/db/data/node/0

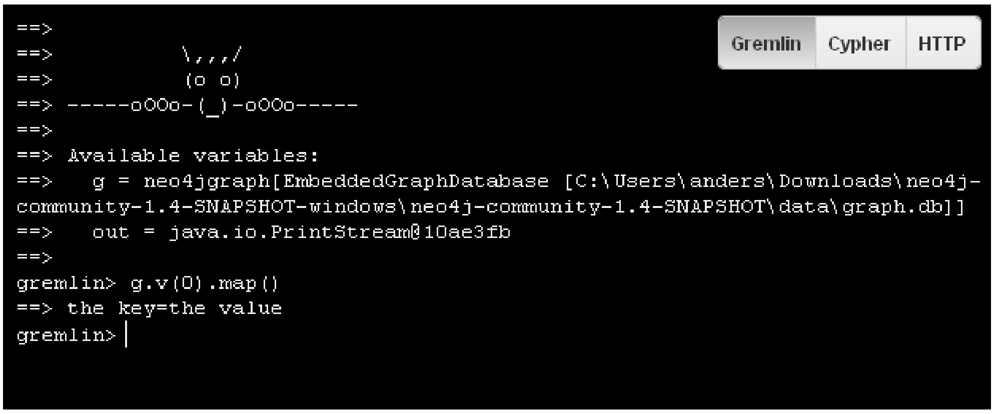
the key | "the value"

13.3. Console tab

The Console tab gives:

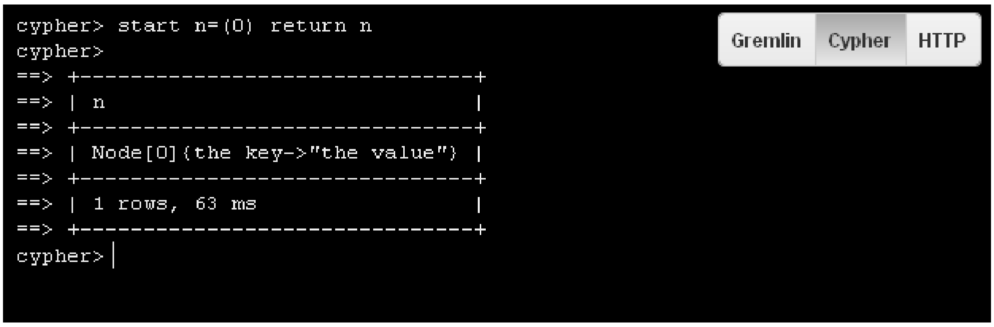
- scripting access to the database via the Gremlin <http://gremlin.tinkerpop.com> scripting engine,
- query access via Cypher,
- HTTP access via the HTTP console.

Figure 13.6. Traverse data with Gremlin



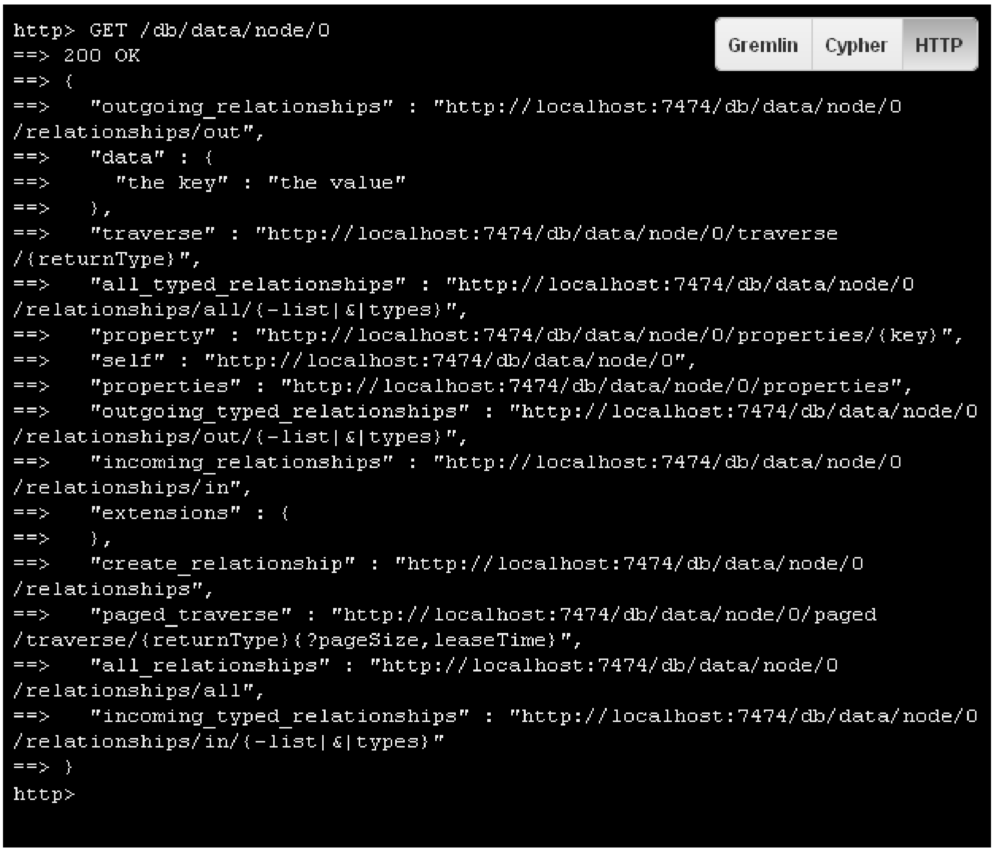
```
==>
==>          \,,,/
==>          (o o)
==> -----oOOo-(_)-oOOo-----
==>
==> Available variables:
==>   g = neo4jgraph[EmbeddedGraphDatabase [C:\Users\anders\Downloads\neo4j-
community-1.4-SNAPSHOT-windows\neo4j-community-1.4-SNAPSHOT\data\graph.db]]
==>   out = java.io.PrintStream@10ae3fb
==>
gremlin> g.v(0).map()
==> the key=the value
gremlin>
```

Figure 13.7. Query data with Cypher



```
cypher> start n=(0) return n
cypher>
==> +------------------------------+
==> | n                            |
==> +------------------------------+
==> | Node[0]{the key->"the value"} |
==> +------------------------------+
==> | 1 rows, 63 ms                |
==> +------------------------------+
cypher>
```

Figure 13.8. Interact over HTTP

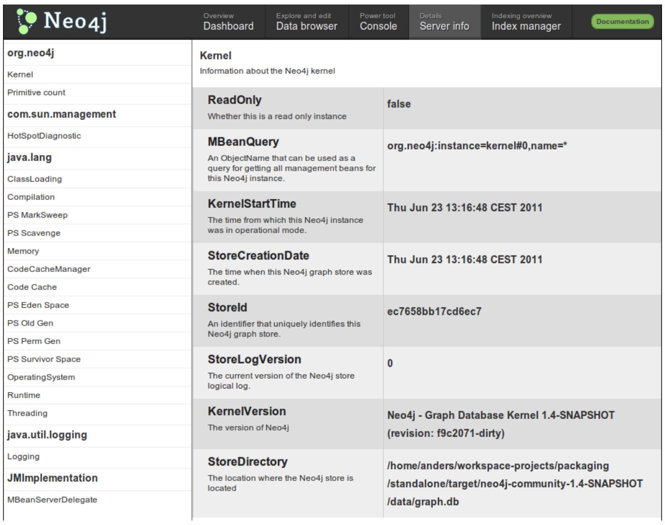


```
http> GET /db/data/node/0
==> 200 OK
==> {
==>   "outgoing_relationships" : "http://localhost:7474/db/data/node/0
/relationships/out",
==>   "data" : {
==>     "the key" : "the value"
==>   },
==>   "traverse" : "http://localhost:7474/db/data/node/0/traverse
/{returnType}",
==>   "all_typed_relationships" : "http://localhost:7474/db/data/node/0
/relationships/all/{-list|&|types}",
==>   "property" : "http://localhost:7474/db/data/node/0/properties/{key}",
==>   "self" : "http://localhost:7474/db/data/node/0",
==>   "properties" : "http://localhost:7474/db/data/node/0/properties",
==>   "outgoing_typed_relationships" : "http://localhost:7474/db/data/node/0
/relationships/out/{-list|&|types}",
==>   "incoming_relationships" : "http://localhost:7474/db/data/node/0
/relationships/in",
==>   "extensions" : {
==>   },
==>   "create_relationship" : "http://localhost:7474/db/data/node/0
/relationships",
==>   "paged_traverse" : "http://localhost:7474/db/data/node/0/paged
/traverse/{returnType}{?pageSize,leaseTime}",
==>   "all_relationships" : "http://localhost:7474/db/data/node/0
/relationships/all",
==>   "incoming_typed_relationships" : "http://localhost:7474/db/data/node/0
/relationships/in/{-list|&|types}"
==> }
http>
```

13.4. The Server Info tab

The Server Info tab provides raw access to all available management objects (see Section 10.3, “Monitoring” for details).

Figure 13.9. JMX Attributes



Kernel

Information about the Neo4j kernel

ReadOnly Whether this is a read only instance	false
MBeanQuery An ObjectName that can be used as a query for getting all management beans for this Neo4j instance.	org.neo4j:instance=kernel#0,name=*
KernelStartTime The time from which this Neo4j instance was in operational mode.	Thu Jun 23 13:16:48 CEST 2011
StoreCreationDate The time when this Neo4j graph store was created.	Thu Jun 23 13:16:48 CEST 2011
StoreId An identifier that uniquely identifies this Neo4j graph store.	ec7658bb17cd6ec7
StoreLogVersion The current version of the Neo4j store logical log.	0
KernelVersion The version of Neo4j	Neo4j - Graph Database Kernel 1.4-SNAPSHOT (revision: f9c2071-dirty)
StoreDirectory The location where the Neo4j store is located	/home/anders/workspace-projects/packaging/standalone/target/neo4j-community-1.4-SNAPSHOT/data/graph.db

Chapter 14. Neo4j Shell

Neo4j shell is a command-line shell for browsing the graph, much like how the Unix shell along with commands like `cd`, `ls` and `pwd` can be used to browse your local file system. It consists of two parts:

- a lightweight client that sends commands via RMI and
- a server that processes those commands and sends the result back to the client.

It’s a nice tool for development and debugging. This guide will show you how to get it going!

14.1. Starting the shell

When used together with Neo4j started as a server, simply issue the following at the command line:

```
./bin/neo4j-shell
```

For the full list of options, see the reference in the Shell manual page.

To connect to a running Neo4j database, use Section 14.1.4, "Read-only mode" for local databases and see Section 14.1.1, "Enabling the shell server" for remote databases.

You need to make sure that the shell jar file is on the classpath when you start up your Neo4j instance.

14.1.1. Enabling the shell server

Shell is enabled from the configuration of the Neo4j kernel, see Section 5.2, "Server Configuration". Here's some sample configurations:

```
# Using default values
enable_remote_shell = true
# ...or specify custom port, use default values for the others
enable_remote_shell = port=1234
```

When using the Neo4j server, see Section 5.2, "Server Configuration" for how to add configuration settings in that case.

There are two ways to start the shell, either by connecting to a remote shell server or by pointing it to a Neo4j store path.

14.1.2. Connecting to a shell server

To start the shell and connect to a running server, run:

```
neo4j-shell
```

Alternatively supply `-port` and `-name` options depending on how the remote shell server was enabled. Then you'll get the shell prompt like this:

```
neo4j-sh (0)$
```

14.1.3. Pointing the shell to a path

To start the shell by just pointing it to a Neo4j store path you run the shell jar file. Given that the right neo4j-kernel-<version>.jar and jta jar files are in the same path as your neo4j-shell-<version>.jar file you run it with:

```
$ neo4j-shell -path path/to/neo4j-db
```

14.1.4. Read-only mode

By issuing the `-readonly` switch when starting the shell with a store path, changes cannot be made to the database during the session.

```
$ neo4j-shell -readonly -path path/to/neo4j-db
```

14.1.5. Run a command and then exit

It is possible to tell the shell to just start, execute a command and then exit. This opens up for uses of background jobs and also handling of huge output of f.ex. an "*ls*" command where you then could pipe the output to "*less*" or another reader of your choice, or even to a file. So some examples of usage:

```
$ neo4j-shell -c "cd -a 24 && set name Mattias"
$ neo4j-shell -c "trav -r KNOWS" | less
```

14.2. Passing options and arguments

Passing options and arguments to your commands is very similar to many CLI commands in an *nix environment. Options are prefixed with a `-` and can contain one or more options. Some options expect a value to be associated with it. Arguments are string values which aren't prefixed with `-`. Let's look at `ls` as an example:

`ls -r -f KNOWS:out -v 12345` will make a verbose listing of node `12345`'s outgoing relationships of type `KNOWS`. The node id, `12345`, is an argument to `ls` which tells it to do the listing on that node instead of the current node (see `pwd` command). However a shorter version of this can be written:

`ls -rfv KNOWS:out 12345`. Here all three options are written together after a single `-` prefix. Even though `f` is in the middle it gets associated with the `KNOWS:out` value. The reason for this is that the `ls` command doesn't expect any values associated with the `r` or `v` options. So, it can infer the right values for the rights options.

14.3. Enum options

Some options expects a value which is one of the values in an enum, f.ex. direction part of relationship type filtering where there's `INCOMING`, `OUTGOING` and `BOTH`. All such values can be supplied in an easier way. It's enough that you write the start of the value and the interpreter will find what you really meant. F.ex. `out`, `in`, `i` or even `INCOMING`.

14.4. Filters

Some commands makes use of filters for varying purposes. F.ex. `-f` in `ls` and in `trav`. A filter is supplied as a json <http://www.json.org/> object (w/ or w/o the surrounding `{}` brackets. Both keys and values can contain regular expressions for a more flexible matching. An example of a filter could be `.*url.*:http.*neo4j.*,name:Neo4j`. The filter option is also accompanied by the options `-i` and `-l` which stands for `ignore case` (ignore casing of the characters) and `loose matching` (it's considered a match even if the filter value just matches a part of the compared value, not necessarily the entire value). So for a case-insensitive, loose filter you can supply a filter with `-f -i -l` or `-fil` for short.

14.5. Node titles

To make it easier to navigate your graph the shell can display a title for each node, f.ex. in `ls -r`. It will display the relationships as well as the nodes on the other side of the relationships. The title is displayed together with each node and its best suited property value from a list of property keys.

If you're standing on a node which has two `KNOWS` relationships to other nodes it'd be difficult to know which friend is which. The title feature addresses this by reading a list of property keys and grabbing the first existing property value of those keys and displays it as a title for the node. So you may specify a list (with or without regular expressions), f.ex: `name,title.*,caption` and the title for each node will be the property value of the first existing key in that list. The list is defined by the client (you) using the `TITLE_KEYS` environment variable and the default being `.*name.*,.*title.*`

14.6. How to use (individual commands)

The shell is modeled after Unix shells like bash that you use to walk around your local file system. It has some of the same commands, like `cd` and `ls`. When you first start the shell (see instructions above), you will get a list of all the available commands. Use `man <command>` to get more info about a particular command. Some notes:

14.6.1. Current node/relationship and path

You have a current node/relationship and a "current path" (like a current working directory in bash) that you've traversed so far. You start at the reference node <http://api.neo4j.org/current/org/neo4j/graphdb/GraphDatabaseService.html#getReferenceNode()> and can then `cd` your way through the graph (check your current path at any time with the `pwd` command). `cd` can be used in different ways:

- `cd <node-id>` will traverse one relationship to the supplied node id. The node must have a direct relationship to the current node.
- `cd -a <node-id>` will do an absolute path change, which means the supplied node doesn't have to have a direct relationship to the current node.
- `cd -r <relationship-id>` will traverse to a relationship instead of a node. The relationship must have the current node as either start or end point. To see the relationship ids use the `ls -vr` command on nodes.
- `cd -ar <relationship-id>` will do an absolute path change which means the relationship can be any relationship in the graph.
- `cd` will take you back to the reference node, where you started in the first place.
- `cd ..` will traverse back one step to the previous location, removing the last path item from your current path (`pwd`).
- `cd start` *(only if your current location is a relationship)*. Traverses to the start node of the relationship.
- `cd end` *(only if your current location is a relationship)*. Traverses to the end node of the relationship.

14.6.2. Listing the contents of a node/relationship

List contents of the current node/relationship (or any other node) with the `ls` command. Please note that it will give an empty output if the current node/relationship has no properties or relationships (for example in the case of a brand new graph). `ls` can take a node id as argument as well as filters, see Section 14.4, "Filters" and for information about how to specify direction see Section 14.3, "Enum options". Use `man ls` for more info.

14.6.3. Creating nodes and relationships

You create new nodes by connecting them with relationships to the current node. For example, `mkrel -t A_RELATIONSHIP_TYPE -d OUTGOING -c` will create a new node (`-c`) and draw to it an `OUTGOING` relationship of type `A_RELATIONSHIP_TYPE` from the current node. If you already have two nodes which you'd like to draw a relationship between (without creating a new node) you can do for example, `mkrel -t A_RELATIONSHIP_TYPE -d OUTGOING -n <other-node-id>` and it will just create a new relationship between the current node and that other node.

14.6.4. Setting, renaming and removing properties

Property operations are done with the `set`, `mv` and `rm` commands. These commands operates on the current node/relationship. * `set <key> <value>` with optionally the `-t` option (for value type) sets a property. Supports every type of value that Neo4j supports. Examples of a property of type `int`:

```
$ set -t int age 29
```

And an example of setting a `double[]` property:

```
$ set -t double[] my_values [1.4,12.2,13]
```

Example of setting a `String` property containing a JSON string:

```
mkrel -c -d i -t DOMAIN_OF --np "{'app':'foobar'}"
```

- `rm <key>` removes a property.
- `mv <key> <new-key>` renames a property from one key to another.

14.6.5. Deleting nodes and relationships

Deletion of nodes and relationships is done with the `rmnode` and `rmrel` commands. `rmnode` can delete nodes, if the node to be deleted still has relationships they can also be deleted by supplying -f option. `rmrel` can delete relationships, it tries to ensure connectedness in the graph, but relationships can be deleted regardless with the -f option. `rmrel` can also delete the node on the other side of the deleted relationship if it's left with no more relationships, see -d option.

14.6.6. Environment variables

The shell uses environment variables a-la bash to keep session information, such as the current path and more. The commands for this mimics the bash commands `export` and `env`. For example you can at anytime issue a `export STACKTRACES=true` command to set the `STACKTRACES` environment variable to `true`. This will then result in stacktraces being printed if an exception or error should occur. List environment variables using `env`

14.6.7. Executing groovy/python scripts

The shell has support for executing scripts, such as Groovy <http://groovy.codehaus.org> and Python <http://www.python.org> (via Jython <http://www.jython.org>). As of now the scripts (*.groovy, *.py) must exist on the server side and gets called from a client with for example, `gsh --renamePerson 1234 "Mathias" "Mattias" --doSomethingElse` where the scripts renamePerson.groovy and doSomethingElse.groovy must exist on the server side in any of the paths given by the `GSH_PATH` environment variable (defaults to .:src:src/script). This variable is like the java classpath, separated by a `:`. The python/jython scripts can be executed with the `jsh` in a similar fashion, however the scripts have the .py extension and the environment variable for the paths is `JSH_PATH`.

When writing the scripts assume that there's made available an `args` variable (a String[]) which contains the supplied arguments. In the case of the `renamePerson` example above the array would contain `["1234", "Mathias", "Mattias"]`. Also please write your outputs to the `out` variable, such as `out.println( "My tracing text" )` so that it will be printed at the shell client instead of the server.

14.6.8. Traverse

You can traverse the graph with the `trav` command which allows for simple traversing from the current node. You can supply which relationship types (w/ regex matching) and optionally direction as well as property filters for matching nodes. In addition to that you can supply a command line to execute for each match. An example: `trav -o depth -r KNOWS:both,HAS_.*:incoming -c "ls $n"`. Which means traverse depth first for relationships with type `KNOWS` disregarding direction and incoming relationships with type matching `HAS_.\*` and do a `ls <matching node>` for each match. The node filtering is supplied with the `-f` option, see Section 14.4, "Filters". See Section 14.3, "Enum options” for the traversal order option. Even relationship types/directions are supplied using the same format as filters.

14.6.9. Query with Cypher

You can use Cypher to query the graph. For that, use the `start` command.

- `start n = (0) return n` will give you a listing of the node with ID 0

14.6.10. Indexing

It’s possible to query and manipulate indexes via the index command. Example: `index -i persons name` (will index the name for the current node or relationship in the "persons" index).

- `-g` will do exact lookup in the index and display hits. You can supply -c with a command to be executed for each hit.
- `-q` will ask the index a query and display hits. You can supply -c with a command to be executed for each hit.
- `--cd` will change current location to the hit from the query. It’s just a convenience for using the -c option.
- `--ls` will do a listing of the contents for each hit. It’s just a convenience for using the -c option.
- `-i` will index a key-value pair in an index for the current node/relationship. If no value is given the property value for that key for the current node is used as value.
- `-r` will remove a key-value pair (if it exists) from an index for the current node/relationship. Key and value is optional.

14.7. Extending the shell: Adding your own commands

Of course the shell is extendable and has a generic core which has nothing to do with Neo4j… only some of the commands <http://components.neo4j.org/neo4j-shell/1.4.2/apidocs/org/neo4j/shell/App.html> do.

So you say you'd like to start a Neo4j graph database <http://api.neo4j.org/current/org/neo4j/graphdb/GraphDatabaseService.html>, enable the remote shell and add your own apps to it so that your apps and the standard Neo4j apps co-exist side by side? Well, here's an example of how an app could look like:

```
import org.neo4j.helpers.Service;
import org.neo4j.shell.kernel.apps.GraphDatabaseApp;

@Service.Implementation( App.class )
public class LsRelTypes extends GraphDatabaseApp
{
    @Override
    protected String exec( AppCommandParser parser, Session session, Output out )
            throws ShellException, RemoteException
    {
        GraphDatabaseService graphDb = getServer().getDb();
        out.println( "Types:" );
        for ( RelationshipType type : graphDb.getRelationshipTypes() )
        {
            out.println( type.name() );
        }
        return null;
    }
}
```

And you could now use it in the shell by typing `lsreltypes` (its name is based on the class name) if `getName` method isn't overridden.

If you'd like it to display some nice help information when using the `help` (or `man`) app, override the `getDescription` method for a general description and use `addValueType` method to add descriptions about (and logic to) the options you can supply when using your app.

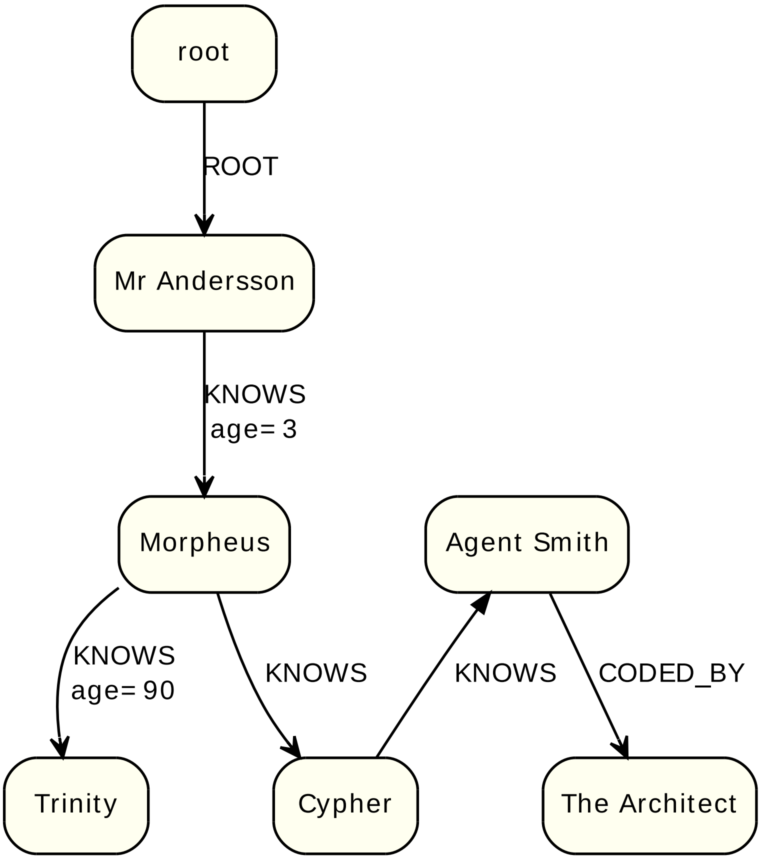
Know that the apps reside server-side so if you have a running server and starts a remote client to it from another JVM you can't add your apps on the client.

14.8. An example shell session

```
# where are we?
neo4j-sh (0)$ pwd
Current is (0)
(0)


# On the current node, set the key "name" to value "Jon"
neo4j-sh (0)$ set name "Jon"

# send a cypher query
neo4j-sh (Jon,0)$ start n=(0) return n
+----------------------+
| n                    |
+----------------------+
| Node[0]{name->"Jon"} |
+----------------------+
1 rows, 44 ms


# make an incoming relationship of type LIKES, create the end node with the node properties specified.
neo4j-sh (Jon,0)$ mkrel -c -d i -t LIKES --np "{'app':'foobar'}"

# where are we?
neo4j-sh (Jon,0)$ ls
*name =[Jon]
(me) <-[LIKES]-- (1)


# change to the newly created node
neo4j-sh (Jon,0)$ cd 1

# list relationships, including relationshship id
neo4j-sh (1)$ ls -avr
(me) --[LIKES,0]-> (Jon,0)


# create one more KNOWS relationship and the end node
neo4j-sh (1)$ mkrel -c -d i -t KNOWS --np "{'name':'Bob'}"

# print current history stack
neo4j-sh (1)$ pwd
Current is (1)
(Jon,0)-->(1)


# verbose list relationships
neo4j-sh (1)$ ls -avr
(me) --[LIKES,0]-> (Jon,0)
(me) <-[KNOWS,1]-- (Bob,2)
```

14.9. A Matrix example

This example is creating a graph of the characters in the Matrix via the shell and then executing Cypher queries against it:



Neo4j is configured for autoindexing in this case with

```
node_auto_indexing=true
node_keys_indexable=name,age

relationship_auto_indexing=true
relationship_keys_indexable=ROOT,KNOWS,CODED_BY
```

in the neo4j configuration file.

The following is a sample shell session creating the Matrix graph and querying it.

```
# create the Thomas Andersson node
neo4j-sh (0)$ mkrel -t ROOT -c -v
Node (1) created
Relationship [ROOT,0] created


# go to the new node
neo4j-sh (0)$ cd 1

# set the name property
neo4j-sh (1)$ set name "Thomas Andersson"

# create Thomas direct friends
neo4j-sh (Thomas Andersson,1)$ mkrel -t KNOWS -cv
Node (2) created
Relationship [KNOWS,1] created
```

```
# go to the new node
neo4j-sh (Thomas Andersson,1)$ cd 2

# set the name property
neo4j-sh (2)$ set name "Trinity"

# go back in the history stack
neo4j-sh (Trinity,2)$ cd ..

# create Thomas direct friends
neo4j-sh (Thomas Andersson,1)$ mkrel -t KNOWS -cv
Node (3) created
Relationship [KNOWS,2] created


# go to the new node
neo4j-sh (Thomas Andersson,1)$ cd 3

# set the name property
neo4j-sh (3)$ set name "Morpheus"

# create relationship to Trinity
neo4j-sh (Morpheus,3)$ mkrel -t KNOWS -n 2

# list the relationships of node 3
neo4j-sh (Morpheus,3)$ ls -rv
(me) --[KNOWS,3]-> (Trinity,2)
(me) <-[KNOWS,2]-- (Thomas Andersson,1)


# change the current position to relationship #2
neo4j-sh (Morpheus,3)$ cd -r 2

# set the age property on the relationship
neo4j-sh [KNOWS,2]$ set -t int age 3

# back to Morpheus
neo4j-sh [KNOWS,2]$ cd ..

# next relationsip
neo4j-sh (Morpheus,3)$ cd -r 3

# set the age property on the relationship
neo4j-sh [KNOWS,3]$ set -t int age 90

# position to the start node of the current relationship
neo4j-sh [KNOWS,3]$ cd start

# new node
neo4j-sh (Morpheus,3)$ mkrel -t KNOWS -c

# list relationships on the current node
neo4j-sh (Morpheus,3)$ ls -r
(me) --[KNOWS]-> (Trinity,2)
(me) --[KNOWS]-> (4)
(me) <-[KNOWS]-- (Thomas Andersson,1)


# go to Cypher
neo4j-sh (Morpheus,3)$ cd 4

# set the name
neo4j-sh (4)$ set name Cypher
```

```
# create new node from Cypher
neo4j-sh (Cypher,4)$ mkrel -ct KNOWS

# list relationships
neo4j-sh (Cypher,4)$ ls -r
(me) --[KNOWS]-> (5)
(me) <-[KNOWS]-- (Morpheus,3)


# go to the Agent Smith node
neo4j-sh (Cypher,4)$ cd 5

# set the name
neo4j-sh (5)$ set name "Agent Smith"

# outgoing relationship and new node
neo4j-sh (Agent Smith,5)$ mkrel -cvt CODED_BY
Node (6) created
Relationship [CODED_BY,6] created


# go there
neo4j-sh (Agent Smith,5)$ cd 6

# set the name
neo4j-sh (6)$ set name "The Architect"

# go to the first node in the history stack
neo4j-sh (The Architect,6)$ cd

# Morpheus' friends, looking up Morpheus by name in the Neo4j autoindex
neo4j-sh (0)$ start morpheus = (node_auto_index, name, 'Morpheus') match morpheus-[:KNOWS]-zionist return zion
+------------------+
| zionist.name     |
+------------------+
| Trinity          |
| Cypher           |
| Thomas Andersson |
+------------------+
3 rows, 28 ms
```

Part IV. Troubleshooting

Chapter 15. Troubleshooting guide

Problem	Cause	Resolution
OutOfMemoryError	Too large top level transactions or leaking transactions not finished properly.	Split updates into smaller transactions. Always make sure transactions are finished properly.
ResourceAcquisitionFailedException or an error message containing the text “The transaction is marked for rollback only”	Leaked non finished transaction tied to the current thread in state marked for rollback only.	Finish transactions properly.
DeadlockDetectedException	Concurrent updates of contended resources or not finishing transactions properly.	See Section 3.4, “Deadlocks”.

Chapter 16. Community support

Get help from the Neo4j open source community, here are some starting points:

- Searchable user mailing list archive <http://www.mail-archive.com/user@lists.neo4j.org/info.html>.
- User mailing list <https://lists.neo4j.org/mailman/listinfo/user>.
- Neo4j wiki <http://wiki.neo4j.org/>
- IRC channel: irc://irc.freenode.net/neo4j

Appendix A. Manpages

The Neo4j Unix manual pages are included on the following pages.

Name

neo4j — Neo4j Server control and management

Synopsis

neo4j <command>

DESCRIPTION

Neo4j is a graph database, perfect for working with highly connected data.

COMMANDS

console
Start the server as an application, running as a foreground proces. Stop the server using `CTRL-C`.

start
Start server as daemon, running as a background process.

stop
Stops a running daemonized server.

restart
Restarts the server.

status
Current running state of the server.

install
Installs the server as a platform-appropriate system service.

remove
Uninstalls the system service.

info
Displays configuration information, such as the current NEO4J_HOME and CLASSPATH.

Usage - Windows

Neo4j.bat

Double-clicking on the Neo4j.bat script will start the server in a console. To quit, just press `control-C` in the console window.

Neo4j.bat install/remove

Neo4j can be installed and run as a Windows Service, running without a console window. You'll need to run the scripts with Administrator priveleges. Just use the Neo4j.bat script with the proper argument:

- Neo4j.bat install - install as a Windows service
  - will install the service
- Neo4j.bat remove - remove the Neo4j service
  - will stop and remove the Neo4j service
- Neo4j.bat start - will start the Neo4j service
  - will start the Neo4j service if installed or a console
  - session otherwise.
- Neo4j.bat stop - stop the Neo4j service if running

- Neo4j.bat restart - restart the Neo4j service if installed
- Neo4j.bat status - report on the status of the Neo4j service
  - returns RUNNING, STOPPED or NOT INSTALLED

FILES

conf/neo4j-server.properties
Server configuration.

conf/neo4j-wrapper.conf
Configuration for service wrapper.

conf/neo4j.properties
Tuning configuration for the database.

Name

neo4j-shell — a command-line tool for exploring and manipulating a graph database

Synopsis

neo4j-shell [*REMOTE OPTIONS*]

neo4j-shell [*LOCAL OPTIONS*]

DESCRIPTION

Neo4j shell is a command-line shell for browsing the graph, much like how the Unix shell along with commands like `cd`, `ls` and `pwd` can be used to browse your local file system. The shell can connect directly to a graph database on the file system. To access local a local database used by other processes, use the readonly mode.

REMOTE OPTIONS

-port *PORT*
: Port of host to connect to (default: 1337).

-host *HOST*
: Domain name or IP of host to connect to (default: localhost).

-name *NAME*
: RMI name, i.e. rmi://<host>:<port>/<name> (default: shell).

-readonly
: Access the database in read-only mode. The read-only mode enables browsing a database that is used by other processes.

LOCAL OPTIONS

-path *PATH*
: The path to the database directory. If there is no database at the location, a new one will e created.

-pid *PID*
: Process ID to connect to.

-readonly
: Access the database in read-only mode. The read-only mode enables browsing a database that is used by other processes.

-c *COMMAND*
: Command line to execute. After executing it the shell exits.

-config *CONFIG*
: The path to the Neo4j configuration file to be used.

EXAMPLES

Examples for remote:

```
neo4j-shell
neo4j-shell -port 1337
neo4j-shell -host 192.168.1.234 -port 1337 -name shell
neo4j-shell -host localhost -readonly
```

Examples for local:

```
neo4j-shell -path /path/to/db
```

```
neo4j-shell -path /path/to/db -config /path/to/neo4j.config
neo4j-shell -path /path/to/db -readonly
```

Name

neo4j-coordinator — Neo4j Coordinator for High-Availability clusters

Synopsis

neo4j-coordinator <command>

DESCRIPTION

Neo4j Coordinator is a server which provides coordination for a Neo4j High Availability Data cluster. A "coordination cluster" must be started and available before the "data cluster" can be started. This server is a member of the cluster.

COMMANDS

console
: Start the server as an application, running as a foreground proces. Stop the server using `CTRL-C`.

start
: Start server as daemon, running as a background process.

stop
: Stops a running daemonized server.

restart
: Restarts a running server.

status
: Current running state of the server

install
: Installs the server as a platform-appropriate system service.

remove
: Uninstalls the system service

FILES

conf/coord.cfg
: Coordination server configuration.

conf/coord-wrapper.cfg
: Configuration for service wrapper.

data/coordinator/myid
: Unique identifier for coordinator instance.

Name

neo4j-coordinator-shell — Neo4j Coordinator Shell interactive interface

Synopsis

neo4j-coordinator-shell -server <host:port> [<cmd> <args>]

DESCRIPTION

Neo4j Coordinator Shell provides an interactive text-based interface to a running Neo4j Coordinator server.

OPTIONS

-server *HOST:PORT*
Connects to a Neo4j Coordinator at the specified host and port.