Neo4j Cookbook

Harness the power of Neo4j to perform complex data analysis over the course of 75 easy-to-follow recipes

Ankur Goel
In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 1 'Chapter title'
- A synopsis of the book’s content
- More information on Neo4j Cookbook

About the Author

Ankur Goel has been passionate about software development for a long time. He is particularly interested in exploring upcoming technologies in the areas of big data, artificial intelligence, and machine learning. He started his career with CDAC High Performance Computing Lab, where he worked on the PARAM series of supercomputers and built real-world use cases around them. He has been working for start-up companies and research labs for most of his career.

He also enjoys working with start-ups that are in the stealth mode stage, and helps them with their technology stack.

He lives in the beautiful city of Chandigarh, India, with his pretty wife and cute newborn daughter.
Neo4j Cookbook

In this connected world, where gigabytes of unstructured information gets generated every second, the Neo4j graph database fits in perfectly to store such data and visualize it from every angle. A graph is the most natural form in which data can be stored and visualized, where every connected edge gives you a new path of the data flow and possible new insights into the data, which are not possible in the traditional data stores. Nearly, every company in a wide variety of domains, such as healthcare, retail, and travel, have realized the true potential of graph databases, and started to explore them for various use cases, such as recommendation, pattern detection, optimizing routes, and tons of other examples.

Many fortune 500 companies have adopted graph databases for a wide array of business-critical use cases and many start-ups are adopting them for new innovative use cases, which were never heard of before. Facebook, LinkedIn, and Twitter are the most well-known users of graph technologies for social web properties.

Neo4j, a graph database by Neo Technologies, is the leading player in the graph database market. It is so easy to use that even a non-technical person can easily browse the data and explore new use cases around it. With this also comes power-packed features that any enterprise database should have.

This cookbook not only provides insights into Neo4j but also into tools, libraries, and visualization tools in a short, concise manner, which will be required frequently when you are exploring Neo4j for a new case, deploying Neo4j to production, or scaling it to gigabytes of connected data. Regardless of whether you are a programmer, database expert, or data scientist, this book has recipes that can be easily learnt and applied.

What This Book Covers

Chapter 1, Getting Started with Neo4j, introduces Neo4j, and the installation of Neo4j on the operating system of your choice. It also helps you move your data from various formats, such as CSV, Geoff, or from other databases, such as RDBMS, OrientDB, and so on.

Chapter 2, Connecting to Neo4j, deals with the various ways in which you can connect to Neo4j with the programming language of your choice, such as Java, Python, Scala, and PHP, and also shows how you can access it over the REST interface, as well as using an embedded version of it.

Chapter 3, The Cypher Query Language, teaches you the most used aspect of Neo4j, which is Cypher. You will learn about queries, which you will need frequently when working with Neo4j.
Chapter 4, *Data Modeling and Leveraging with Neo4j*, explores the data modeling concepts and techniques associated with the graph data in Neo4j; in particular, the property graph model, design constraints for Neo4j, and modeling across multiple domains.

Chapter 5, *Mining the Social Treasure*, helps you build frequently used use cases around social data. Whether you use data from popular social networks, such as Facebook, LinkedIn, or have data of your own, this chapter quickly gets you started with social use cases.

Chapter 6, *Developing Location-based Services with Neo4j*, teaches you the most important aspect of today's data, location, and how to deal with it in Neo4j. You can also learn how to import geospatial data into Neo4j and run queries, such as proximity searches, bounding boxes, and so on.

Chapter 7, *Visualization of Graphs*, shows you how to integrate the Neo4j graph database with the powerful domain of graph visualizations. We will discuss the different alternatives and resources to get started with.

Chapter 8, *Industry Usages of Neo4j*, shows you how different industries, such as healthcare, travel, and retail, use Neo4j in their domains. This all comes with a sample dataset and queries, which you can easily build and execute to see it running.

Chapter 9, *Neo4j Administration and Maintenance*, deals with recipes for deploying Neo4j on different clouds, backup strategies, debugging and security aspects.

Chapter 10, *Scaling Neo4j*, teaches you how to develop applications with Neo4j to handle high volumes of data. You will learn about different aspects while scaling different types of applications over Neo4j.
1

Getting Started with Neo4j

In this chapter, we will cover the following recipes:

- Single node installation of Neo4j over Linux
- Single node installation of Neo4j over Windows
- Single node installation of Neo4j over Mac OS X
- Creating your first graph with Neo4j
- Importing data from the CSV format to Neo4j
- Importing data from RDMS to Neo4j
- Importing data in the Geoff format to Neo4j
- Importing data from OrientDB to Neo4j
- Importing data from InfiniteGraph to Neo4j
- Importing data from the DEX graph database to Neo4j
- Common configurations of Neo4j
- Running multiple instances of Neo4j over a single machine
- Building Neo4j from the source
Introduction

Neo4j is a highly scalable, fully transactional ACID (atomicity, consistency, isolation, and durability) graph database that stores data structured as graphs. It allows developers to achieve excellent performance in queries over large, complex graph datasets and at the same time, it is very simple and intuitive to use. This chapter consists of readymade recipes that allow users to hit the ground running with Neo4j. There are several recipes to set up Neo4j over a wide array of platforms, such as Linux, Windows, Mac, Android, and so on. Neo4j runs in different configuration modes: server and embedded inside application. Both of these configuration modes has been fully explained in this chapter. This chapter also includes common configurations of the key configuration files.

Single node installation of Neo4j over Linux

Neo4j is a highly scalable graph database that runs over all the common platforms; it can be used as is or can be embedded inside applications as well. The following recipe will show you how to set up a single instance of Neo4j over the Linux operating system.

Getting ready

Perform the following steps to get started with this recipe:

- Download the community edition of Neo4j from http://www.neo4j.org/download for the Linux platform:
  
  $ wget http://dist.neo4j.org.neo4j-community-2.2.0-M02-unix.tar.gz

- Check whether Java is installed for your operating system or not by typing this in the shell prompt:
  
  $ echo $JAVA_HOME

  If this command produces no output, install JDK/JRE for your Linux distribution and also set the JAVA_HOME path
Chapter 1

How to do it...

Now, let’s install Neo4j over the Linux operating system, which is simple, as shown in the following steps:

1. Extract the TAR file by using the following command:
   $$\texttt{tar -zxvf neo4j-community-<version>-unix.tar.gz}$$
   $$\texttt{ls}$$
2. Go to the bin directory under the root folder:
   $$\texttt{cd <neo4j-community-version>/bin/}$$
3. Start the Neo4j graph database server:
   $$\texttt{./neo4j start}$$
4. Check whether Neo4j is running or not by using the following command:
   $$\texttt{./neo4j status}$$

Neo4j can also be monitored using the web console. Open `http://<ip>:7474/webadmin`, as shown in the following screenshot:

![Neo4j Web Console](image)

The preceding diagram is a screenshot of the web console of Neo4j, through which the server can be monitored and different Cypher queries can be run on the graph database.
How it works...

Neo4j comes with prebuilt binaries over the Linux operating system, which can be extracted and run over. Neo4j comes with both web-based and terminal-based consoles, over which the Neo4j graph database can be explored.

See also

During installation, you may face several kind of issues, such as the maximum number of files you can keep open at once and so on. For more information, check out http://neo4j.com/docs/stable/server-installation.html#linux-install.

Single node installation of Neo4j over Windows

Neo4j is a highly scalable graph database that runs over all the common platforms; it can be used as is or can be embedded inside applications. The following recipe will show you how to set up a single instance of Neo4j over the Windows operating system.

Getting ready

Perform the following steps to get started with this recipe:

- Download the Windows installer from http://www.neo4j.org/download. This has both 32-bit and 64-bit prebuilt binaries.
- Check whether Java is installed for the operating system or not by typing this in the cmd prompt:
  
  ```
  echo %JAVA_HOME%
  ```

  If this command throws no output, install JDK/JRE for your Windows distribution and also set the JAVA_HOME path.
How to do it...

Now, let's install Neo4j over the Windows operating system, which is simple, as shown here:

1. Run the installer by clicking on the downloaded file:

The preceding screenshot shows the Windows installer running.

2. After the installation is complete, when you run the software, it will ask for the database location. Choose the location carefully, as the entire graph database will be stored in this folder:
Getting Started with Neo4j

The preceding screenshot shows the Windows installer asking for the graph database's location.

3. The Neo4j browser can be opened by entering http://localhost:7474/ in the browser. The following screenshot depicts Neo4j started over the Windows platform:

![](image)

How it works...

Neo4j comes with prebuilt binaries over the Windows operating system, which can be extracted and run over. Neo4j comes with both web-based and terminal-based consoles, over which the Neo4j graph database can be explored.

See also

During installation, you might face several kinds of issues such as max open files and so on. For more information, check out http://neo4j.com/docs/stable/server-installation.html#windows-install.

Single node installation of Neo4j over Mac OS X

Neo4j is a highly scalable graph database that runs over all the common platforms; it can be used as in a mode and can also be embedded inside applications. The following recipe will show you how to set up a single instance of Neo4j over the OS X operating system.
Getting ready

Perform the following steps to get started with this recipe:

- Download the binary version of Neo4j from http://www.neo4j.org/download for the Mac OS X platform and the community edition, as shown in the following command:
  
  $ wget http://dist.neo4j.org/neo4j-community-2.2.0-M02-unix.tar.gz

- Check whether Java is installed for the operating system or not by typing this over the cmd prompt:
  
  $ echo $JAVA_HOME

  If this command throws no output, install JDK/JRE for your Mac OS X distribution and also set the JAVA_HOME path.

How to do it...

Now, let's install Neo4j over the OS X operating system, which is very simple, as shown in the following steps:

1. Extract the TAR file using the following command:

   $ tar -zxvf neo4j-community-<version>-unix.tar.gz

   $ ls

2. Go to the bin directory under the root folder:

   $ cd <neo4j-community-version>/bin/

3. Start the Neo4j graph database server:

   $ ./neo4j start

4. Check whether Neo4j is running or not by using the following command:

   $ ./neo4j status

How it works...

Neo4j comes with prebuilt binaries over the OS X operating system, which can be extracted and run over. Neo4j comes with both web-based and terminal-based consoles, over which the Neo4j graph database can be explored.
There's more...

Neo4j over Mac OS X can also be installed using brew, which has been explained here.

Run the following commands over the shell:

$ brew update
$ brew install neo4j

After this, Neo4j can be started by using the start option with the Neo4j command:

$ neo4j start

This will start the Neo4j server, which can be accessed from the default URL (http://localhost:7474).

The installation can be reached by using the following commands:

$ cd /usr/local/Cellar/neo4j/
$ cd {NEO4J_VERSION}/libexec/

You can learn more about OS X installation from http://neo4j.com/docs/stable/server-installation.html#osx-install.

Creating your first graph with Neo4j

After the successful setup of Neo4j on an operating system of our choice, now it's time to say Hello World to Neo4j, which means it's time to create our first graph by using Neo4j.

We know that any graph consists of nodes and edges, where edges represent the relationships between nodes.

Consider an example where there are two persons, Alice and Bob, who know each other. So, in graph terminology, Alice will be node A and Bob will be node B. The technical representation of this example can be done as follows:

- **Nodes**: A and B
- **Edges**: A——— knows ———B
The preceding diagram shows nodes and edges, where edges represent the properties between the nodes.

**Getting ready**

To get started with this recipe, install Neo4j by using the earlier recipes of this chapter.

**How to do it...**

There are many ways to create a graph with Neo4j. However, in order to create our first graph, we will use the Neo4j shell that comes with Neo4j by default and can be intuitively operated from both the command line and the shell.

For our first graph, consider a scenario where London and Paris are two cities that are connected by the following flights:

- Airline X, which connects London to Paris daily (start time: 1400 hours)
- Airline Y, which connects Paris to London daily (start time: 2300 hours)
Let's get started to create our first graph using the Neo4j shell. To do so, perform the following steps:

1. Start the Neo4j server by using the following command:
   
   ```bash
   ${NEO4J_ROOT}/bin/neo4j start
   ```
   
   The detailed steps to start the Neo4j server has been described in the previous recipes.

2. The Neo4j shell can be invoked by two methods. The first method is to simply type in the following command (under the same `<neo4j_Home_Directory>/bin` directory):
   
   ```bash
   ${NEO4J_ROOT}/bin/neo4j-shell
   ```
   
   The output of this command is shown as follows:

   ```
   $ ./neo4j-shell
   Welcome to the Neo4j Shell! Enter 'help' for a list of commands
   NOTE: Remote Neo4j graph database service 'shell' at port 1337
   ```

   The nodes are created using the `mknode` command as follows:

   ```bash
   neo4j-sh (0) $ mknode London
   neo4j-sh (0) $ mknode Paris
   ```

3. Let's create a node and enter this node by using the `cd` option with `mknode`:

   ```bash
   neo4j-sh (0) $ mknode --cd --np "{"name":"London"}"
   ```

   The `np` option can be used to specify as many properties as you want with that node.

4. Now, we will create another node with the name Paris:

   ```bash
   neo4j-sh (0) $ mknode --np "{"name":Paris}" -v
   ```
5. Next, we will create a relationship between them by executing the following commands from the command line:

```bash
neo4j-sh (London,2)$ mkrel -d OUTGOING -t CONNECTED <nodeid from preceding command> -rp "{'Airline':'X','Start-Time':'1400'}"
```

```bash
neo4j-sh (London,2)$ ls
*name = [London]
(me)-[:CONNECTED]->(Paris,3)
```

The `mkrel` command is used to create a relationship. To see the options in detail, type `man mkrel` in the Neo4j shell.

Let's create another relationship, as demonstrated by the following commands:

```bash
neo4j-sh (London,2)$ mkrel -d INCOMING -t CONNECTED <nodeid> -rp "{'Airline':'Y','Start-Time':'2300'}"
```

```bash
neo4j-sh (London,2)$ cd 3
neo4j-sh (Paris,3)$ ls
*name = [Paris]
(me)<-[::CONNECTED]-(London,2)
```

6. Let's visualize our first graph in the browser. For this, go to the Neo4j webadmin URL and then click on Data Browser; you will see something similar to the following screenshot:

![Data Visualization Screenshot](image)

We can see two nodes, 2 and 3, in the data visualization, which are connected to each other.
How it works...

The Neo4j shell comes with the handy utilities of `mknode` to create new nodes with properties and with `mkrel` to create relationships among them.

Nodes in Neo4j are analogous to files in the Unix filesystem, except with one major difference. The difference is that when you create a file in any directory, a relationship automatically gets created between the parent directory and the file. Using this relationship, we can browse the filesystem, whereas `mknode` in Neo4j creates disjointed nodes that cannot be browsed, as they don’t have any relationship between them.

There’s more...

To study more about the `mknode` and `mkrel` commands, use the `man` pages under the Neo4j shell. If you want to delete an entire graph that you have just created, the following are the steps to do so:

1. Stop the Neo4j server by using the following command:
   
   `$ ./neo4j stop`

2. Delete the `graph.db` file under the data directory (assuming that you are using the default configuration):

   `$ rm -rf data/graph.db`

   Once deleted, the data is not recoverable.

Importing data from the CSV format to Neo4j

Graph data comes in different formats, and sometimes it’s a combination of two or more formats. It is very important to learn about the various ways to import data, which is in different formats into Neo4j. In this recipe, you will learn how to import data present in the CSV file format into the Neo4j graph database server. A sample CSV file is shown as follows:

```
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>type</td>
<td>name</td>
</tr>
<tr>
<td>1</td>
<td>node</td>
<td>London</td>
</tr>
<tr>
<td>2</td>
<td>node</td>
<td>Paris</td>
</tr>
<tr>
<td>3</td>
<td>node</td>
<td>Tokyo</td>
</tr>
<tr>
<td>4</td>
<td>node</td>
<td>Delhi</td>
</tr>
</tbody>
</table>
```
Getting ready

To get started with this recipe, install Neo4j by using the steps from the earlier recipes of this chapter.

How to do it...

There are several methods that you can use to import data which is in the CSV format or Excel into Neo4j, which are described in the sections that follow.

Using a batch importer

There is an excellent tool written by Michael Hunger, which can be cloned from https://github.com/jexp/batch-import.

The CSV file has to be converted into the format specified in the readme file. The tool is very flexible in terms of the number of properties and the types of each property. The nodes and relationships can be within the same file or within multiple files. The example file format is present in the sample directory. To run the tool, use the following command:

```
$ wget https://dl.dropboxusercontent.com/u/14493611/batch_importer_22.zip
$ unzip batch_importer_22.zip
# Download sample nodes.csv and rels.csv from the github repo under sample
$ import.sh test.db nodes.csv rels.csv
$ cp test.db ${NEO4J_ROOT}/data/graph.db
```

Each parameter in the command has been fully explained in the readme file.

The batch import tool also supports a parallel batch inserter, which can speed up the process of importing data from a large number of nodes and relationships.

Benchmark figures claimed by the batch importer tool are 2 billion nodes and 20 billion relationships in 11 hours (500K elements/second).

This is claimed over the EC2 high I/O instance.
Using custom scripts

Custom scripts can be written in any language to import data from CSV files. Custom scripts give you the advantages of checking various erroneous scenarios, leaving out redundant columns, and other flexibilities. For a smaller number of nodes and relationships, custom scripts can be written in any language of your choice.

The exact format of the script will depend on the CSV file. You can write the script as follows:

```
#Bash Script for importing nodes
NEO4J_ROOT="/var/lib/neo4j"
while read LINE
do
  name=`echo $LINE | awk -F "," '{print $3}'`
  ${NEO4J_ROOT}/bin/neo4j-shell -c mknode --np "{'name':$name}" -v
done
```

```
#Bash Script for creating relationships
#Format of csv should be startnode,endnode,type,direction
NEO4J_ROOT="/var/lib/neo4j"
IFS=""
while read LINE
do
  echo $LINE
  array=($LINE)
  ${NEO4J_ROOT}/bin/neo4j-shell -c cd -a ${array[0]} mkrel -d ${array[3]} -t ${array[2]} ${array[1]}
done
```

This task can also be achieved in Python using the py2neo module, as shown in the following script:

```
#Sample Python code to create nodes from csv file
import csv
from py2neo import neo4j, cypher
from py2neo import node, rel
```
graph_db = neo4j.Graph('http://localhost:7474/db/data/)
ifile = open('nodes.csv', 'rb')
reader = csv.reader(ifile)
rownum = 0
for row in reader:
    nodes = graph_db.create({'name':row[2]})
ifile.close()

A similar Python code can be written for creating relationships, too. The py2neo module can also be used to create a batch request, wherein there's a whole list with parameters as shown in the following code:

records = [(101, "A"), (102, "B"), (103, "C")]
graph_db = neo4j.Graph('http://localhost:7474/db/data/)
batch = neo4j.WriteBatch(graph_db)
for emp_no, name in records:
    batch.get_or_create_indexed_node("Employees", "emp_no", emp_no,
    {
        "emp_no": emp_no, "name": name
    })
nodes = batch.submit()

How it works...

Batch import performance is achieved by skipping all the transactional behavior and losing ACID guarantees. If the batch import fails, the database will be broken, possibly irrecoverably, and lead to the loss of all the information.

See also

Custom scripts can be written for REST as well as for the embedded interfaces of Neo4j. For the full cookbook on py2neo recipes, refer to http://py2neo.org/2.0/cookbook.html.
Importing data from RDBMS to Neo4j

Graph data comes in different formats, and sometimes it’s a combination of two or more formats. It is very important to learn about the various ways to import data having different formats into Neo4j. In this recipe, you will learn how to import data present in an RDBMS database, such as MySQL, SQL Server, into the Neo4j graph database server. Following is a sample figure for a graph:

Getting ready

To get started with this recipe, install Neo4j by using the steps from the earlier recipes of this chapter.

How to do it...

The data from RDBMS can be imported by using the two methods described here.

Using the Neo4j SQL importer tool

Peter Neubauer, the man behind the Neo technology, has developed an excellent tool for this purpose, called the Neo4j SQL importer tool, which takes SQL dumps. The tool can be cloned from his repository over GitHub, which is available at https://github.com/peterneubauer/sql-import.
Using custom scripts

Custom scripts can be written for a particular RDBMS schema, which is more useful as it is designed by keeping the schema in mind. Take an example of the following schema:

```
Orders
orderID
customerId
dateOrdered
dateRequired
status
......
```

```
Products
productID
name
description
quantity
unitPrice
......
```

```
OrderDetails
orderID
productID
quantity
lineNumber
......
```

The Orders and Products tables will represent nodes in Neo4j, while OrderDetails will represent the relationships between them. Relationships can be in both the directions. So, starting from the Products node, we can easily find out how many different Orders have been made for that product and vice versa.

How it works...

In the SQL import tool, most of the things revolve around the primary key. Each of the columns can be made a node, and it will have a relationship with the node that is storing the primary key. In the case of relationships with other tables, the relationship will be made on the foreign key.

There's more...

One of the best use cases of Neo4j is to build a recommendation engine on top of it. Since most of the data now resides in traditional RDBMS, the very first step will involve importing the data into Neo4j.
Importing data from the Geoff format to Neo4j

Graph data comes in different formats, and sometimes it's a combination of two or more formats. It is very important to learn about the various ways to import data having different formats into Neo4j. In this recipe, you will learn how to import data that is present in the Geoff format, into the Neo4j graph database server.

Getting ready

To get started with this recipe, install Neo4j by using the steps from the earlier recipes of this chapter.

How to do it...

The data in the Geoff format can be easily imported using the load2neo tool available at http://nigelsmall.com/load2neo.

The following is the code for building the tool:

```
wget http://nigelsmall.com/d/load2neo-0.6.0.zip
```

This ZIP archive contains three files: two JAR files which need to be copied to your Neo4j plugin directory and a neo4j-server.properties file that contains has content to be added to the identically named file within the Neo4j conf directory. This is a single line that mounts the plugin at the correct URI offset.

How it works...

Geoff is a text-based interchange format for Neo4j graph data that should be instantly readable to anyone familiar with Cypher, on which its syntax is based.

This is the syntax of Geoff:

```
(alice {"name":"Alice"})
(bob {"name":"Bob"})
(carol {"name":"Carol"})
(alice)<-[[:KNOWS]]->(bob)<-[[:KNOWS]]->(carol)<-[[:KNOWS]]->(alice)
```
Bulk load

Bulk loads can be executed by running the following `curl` command from the command line:

```
curl -X POST http://localhost:7474/load2neo/load/geoff -d '(a)<-[[:KNOWS]]->(b)
```

See also

To know more about the Geoff format, go to http://nigelsmall.com/geoff.

Importing data from OrientDB to Neo4j

There are tons of options available when it comes to graph databases, such as FlockDB, AllegroGraph, InfiniteGraph, OrientDB, and so on. It is important to learn how to migrate data from any one of these to Neo4j if you are thinking of migrating to Neo4j.

In this recipe, you will learn how to migrate data from OrientDB to the Neo4j server.

Getting ready

To get started with this recipe, install Neo4j using the steps from the earlier recipes of this chapter.

How to do it...

OrientDB is an open source GraphDB, with a mix of features taken from document databases and object orientation.

Exporting in the JSON format

OrientDB has given us the utility to export data in the JSON format. We can access this utility by typing the following commands in a terminal:

```
$ ./console.sh
orientdb> export database graph.json
```

The JSON format is as follows:

```
"records": [{
  "@type": "d", "@rid": "#12:476", "@version": 0, "@class": "Whiz",
  "id": 476,
  "date": "2011-12-09 00:00:00:000",
}]
```
Getting Started with Neo4j

Now, this data can be parsed using a custom script, which can insert data into Neo4j.

Using Gremlin

Gremlin can be used to export data in the XML format from OrientDB and to import data into Neo4j, as shown here:

```java
gremlin> graph = new OrientGraph("local:<path_of_db> ");
gremlin> graph.saveGraphML('graph.xml');
gremlin> graph = new Neo4jGraph('data/graph.db');
gremlin> graph.loadGraphML('graph.xml');
```

Gremlin can also be used to get all the nodes and relationships from OrientDB, which can be inserted into Neo4j, as follows:

```java
gremlin> graph = new OrientGraph("local: <path_of_db> ");
gremlin> graph.V # Get All Vertices
gremlin> graph.E # Get All Edges
```

How it works...

Gremlin is a graph traversal language. Gremlin works over those graph databases/frameworks that implement the Blueprints property graph data model. Fortunately, OrientDB and Neo4j are among them.

See also

To find out more about Gremlin, go to http://www.tinkerpop.com/.
Importing data from InfiniteGraph to Neo4j

There are tons of options available when it comes to graph databases, such as FlockDB, AllegroGraph, InfiniteGraph, OrientDB, and so on. It is important to learn how to migrate data from any one of these to Neo4j if you are thinking of migrating to Neo4j.

In this recipe, you will learn how to migrate data from InfiniteGraph to the Neo4j server.

Getting ready

To get started with this recipe, install Neo4j by using the steps from the earlier recipes of this chapter.

How to do it...

InfiniteGraph, a product of Objectivity, Inc., is an enterprise-proven, distributed graph database that can handle the needs of big data.

The best way to import data from InfiniteGraph to Neo4j is via Gremlin, as shown here:

```
gremlin> import com.tinkerpop.blueprints.impls.ig.*
gremlin> graph = new IGGraph("neo_data.boot")
gremlin> graph.V # Gives all the nodes
gremlin> graph.E # Gives all the edges
gremlin> graph.loadGraphML('graph.xml');
gremlin> graph = new Neo4jGraph('neo/graph.db');
gremlin> graph.loadGraphML('graph.xml');
```

Infinite supports Blueprints, so it works with Gremlin, which means that all the methods also work with InfiniteGraph.

How it works...

Gremlin is a graph traversal language. Gremlin works over those graph databases/frameworks that implement the Blueprints property graph data model. Fortunately, OrientDB and Neo4j are among them.
There's more...

To know more about Gremlin, go to http://www.tinkerpop.com/.

Importing data from the DEX graph database to Neo4j

There are tons of options available when it comes to graph databases, such as FlockDB, AllegroGraph, InfiniteGraph, OrientDB, and so on. It is important to learn how to migrate data from any one of these to Neo4j, if you are thinking of migrating to Neo4j.

In this recipe, you will learn how to migrate data from the DEX graph database to the Neo4j server.

Getting ready

To get started with this recipe, install Neo4j by using the steps from the earlier recipes of this chapter.

How to do it...

DEX is a highly scalable graph database solution, which is mostly written in Java and C++. The key feature of DEX is that its query performance has been optimized for large graph databases. Also, it's very lightweight, which allows the storage of billions of nodes and relationships at a very low metadata storage cost.

The default exporter can be used to export the DEX graph database to GraphML, which can be easily loaded into Neo4j. This is done by using the following lines of code:

```java
DefaultExport graph = new DefaultExport();
g.export("dex_export.graphml", ExportType.YGraphML, graph);
```

Gremlin can also be used to solve the problem, as shown here:

```java
gremlin> graph = new DexGraph("neo/data.dex");
gremlin> graph.saveGraphML('graph.xml');
gremlin> graph = new Neo4jGraph('neo/graph.db');
gremlin> graph.loadGraphML('graph.xml');
```
How it works...

Gremlin is a graph traversal language. Gremlin works over those graph databases/frameworks that implement the Blueprints property graph data model. Fortunately, OrientDB and Neo4j are among them.

See also

To know more about Gremlin, go to http://www.tinkerpop.com/.

Common configurations of Neo4j

Neo4j is very flexible in terms of configuration, and it can be changed to achieve performance, security, and flexibility.

In this recipe, you will learn about the common configuration files of Neo4j.

Getting ready

To get started with this recipe, install Neo4j using the steps from the earlier recipes of this chapter.

Before getting into the recipe, here are some important points that you need to consider:

- The main configuration file can be found at conf/neo4j-server.properties
- The default server logging configuration file is at conf/log4j.properties
- Tuning parameters can be tuned in the conf/neo4j.properties file
- The configuration file for a wrapper used in daemonizing can be found at conf/neo4j-wrapper.properties
- The logging configuration for the HTTP protocol is found in the conf/neo4j-http-logging.xml file

How to do it...

The Neo4j shell can also be used to access a remote graph database. To do so, perform the following steps:

1. Change the following settings:
   - In the server primary configuration file, add this line:
     ```
     enable_remote_shell = true
     ```
The default port for remote shell access can be changed by adding the following line:

```
enable_remote_shell_port=1234
```

Now, you can access the remote database.

2. The Neo4j web console, by default, can be accessed only from the localhost. If you want to access the web console from any machine (which you should never do as anyone can then play with your graph database), then perform the following tasks:

   1. In the server primary configuration file, uncomment this line:
      ```
      org.neo4j.server.webserver.address=0.0.0.0
      ```
      The default port for remote shell access can be changed by editing the following line:
      ```
      org.neo4j.server.webserver.port=7473
      ```

   2. Now, restart the Neo4j server using the following command:
      ```
      ./neo4j restart
      ```

**How it works...**

Neo4j comes with lots of configuration options, and by changing the parameters in different configuration files, you can configure each part of it.

**There's more...**


**Running multiple instances of Neo4j over a single machine**

Many times, there will be a need to run multiple instances of Neo4j over a single machine.

In this recipe, you will learn about how to run multiple instances of Neo4j.

**Getting ready**

To get started with this recipe, install Neo4j by using the steps from the earlier recipes of this chapter.
How to do it...

Neo4j can handle only a single graph instance. To run multiple graph instances, you have to run multiple Neo4j servers over the same machine, as follows:

1. Replicate the configuration file for each instance and change the following parameters:
   
   `org.neo4j.server.database.location=data/graph.db`

   Change this path for each instance by setting different database paths for different instances. Also, for each instance, set different ports for the web console, which is shown in the following parameter:

   `org.neo4j.server.webserver.port=5678`

2. Now, restart the Neo4j server by using the following command:

   `./neo4j restart`

How it works...

Neo4j can handle only one instance at a time. In order to run multiple instances of Neo4j, we have to replicate the files and change the graph database directory of each instance.

See also


Building Neo4j from the source

Neo4j is an open source software that is readily available to other open source developers to change the source code according to their requirements.

In this recipe, you will learn how to build Neo4j from the source.

Getting ready

To get started with this recipe, install JDK and Maven before building Neo4j.
**How to do it...**

Run the following commands to build Neo4j from the source:

```bash
git clone https://github.com/neo4j/neo4j.git
cd neo4j
mvn clean install
```

A good approach for this recipe will be to go through the *readme* file, which is present in the top level directory, and follow the steps given in that file. For more information, please refer to [https://github.com/neo4j/neo4j/](https://github.com/neo4j/neo4j/).

**How it works...**

Neo4j is open source and Java based. It is built using Maven.

**There's more...**

To know more about how to build Neo4j from the source, go to [https://github.com/neo4j/neo4j/](https://github.com/neo4j/neo4j/).
Where to buy this book

You can buy Neo4j Cookbook from the Packt Publishing website.
Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.
Click here for ordering and shipping details.