UNIVERSITY NAME

DOCTORAL THESIS

Thesis Title

Author:
Test123

Supervisor:
Dr. James SMITH

A thesis submitted in fulfillment of the requirements
for the degree of Doctor of Philosophy

in the

Research Group Name
Department or School Name

June 9, 2019

Declaration of Authorship

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Luebeck, June 9, 2019

_____

UNIVERSITY NAME

Abstract

Faculty Name
Department or School Name

Doctor of Philosophy

Thesis Title

by Test123

The Thesis Abstract is written here (and usually kept to just this page). The page is kept centered vertically so can expand into the blank space above the title too...

Contents

List of Figures

List of Tables

For/Dedicated to/To my…

Chapter 1

Hintergrund

Um Daten in einer Datenbank abzulegen ist ein Datenmodell nötig, welches das Allgemeine Verhalten der Datenbank definiert. Dieses ist durch folgende Eigenschaften definiert ist: ein Liste von Datenstruktur Typen, ein Liste von Operatoren die auf die Daten angewandt werden können und eine Liste der Regeln zur Vollständigkeit der Datenbank (Codd, 1981). Zwei dieser Datenmodelle sind der Relationale-Ansatz und die NoSQL-Bewegung. Die meisten Datenbanken werden heutzutage nach dem relationalen Datenmodell verwaltet und mittels SQL bearbeitet (Miller, 2013). Dieses Schema wurde über viele Jahre lang optimiert und gilt für viele Daten als performanteste Implementierung (Miller, 2013). Die verwendete Datenstruktur bildet eine Tabelle, eine Reihe bildet ein Objekt und die Spalten bilden die dazugehörigen Attribute (Miller, 2013). Die Möglichkeiten der Operationen, mit denen die Daten verändert oder angefragt werden,sind durch den SQL-Standard bzw. die Implementierung des Standards limitiert. Die Regeln zur Vollständigkeit der Datenbank sind ebenfalls im Standard festgehalten. Als eine Alternative zu diesem relationalen Datenmodell gibt es die “NoSQL” Bewegung, welche erstmal 1998 erwähnt wurde (Strauch, Sites, and Kriha, 2011). Diese Bewegung versuchte zunächst den Gebrauch von SQL als Anfragesprache zu vermeiden und brachte in den folgenden Jahren weitere Datenmodelle hervor. Eines dieser Konzepte ist das Darstellen von Daten in der Datenstruktur eines Graphen (Miller, 2013). Die sogenannten Graphdatenbanken(GDB) werden besonders in dem Darstellen von Netzwerken verwendet (Han et al., 2011), da das relationalen Datenmodell bei großen Datenmengen und vielen komplex verbunden Informationen durch eine hohe Anzahl von notwendigen Joins nicht performant verwendet werden kann (Miller, 2013). Für GDBs gibt es keine standardisierte Anfragesprache und die Menge an möglichen Operatoren ist sehr variable. Neo4j ist ein populäres Beispiel für Graphdatenbanken und wird im folgenden genauer beschrieben.

Chapter 2

Grundlagen

2.1 Graph

Ein Graph ist eine abstrakte Datenstruktur, welche für die GDBs verwendet werden (Vicknair et al., 2010). Ein Graph besteht aus einer endlichen nicht leeren Summe von Knoten, auch Vertex oder Punkte genannt und Kanten. Die Kanten des Graphen bildet die Verbindung zwischen zwei Knoten http://wwwmayr.in.tum.de/lehre/2008WS/ea/EA-7.pdf(7.06.19), wenn die Kanten aus einem geordneten Paar bestehen und somit eine Richtung besitzen wird der Graph als gerichtet bezeichnet, bei einem ungeordneten Paar wird von einem ungerichteten Graph gesprochen(Bondy and Murty, 1976). Wenn die Kanten Attribute, wie zum Beispiel Kosten besitzen wird der Graph als gewichtet bezeichnet und ohne Attribute als ungewichtet http://wwwmayr.in.tum.de/lehre/2008WS/ea/EA-7.pdf(7.06.19).

2.2 Neo4j

Neo4j ist eine Graphdatenbank, welche in Java implementiert wurde (Vukotic et al., 2015). Als Grundlegende Datenstruktur wird ein gerichteter und gewichteter Graph verwendet.Knoten stellen die Entitäten,beispielsweise eine Person oder ein Produkt, da und Kanten stellen die Relationen zwischen den Entitäten da, beispielsweise “isAngestellt” und können optional ein Gewicht und eine Richtung besitzen. Attribute können als zusätzliche Informationen in den Knoten gespeichert werden wie zum Beispiel: Name bei einem Knoten “Person”. Knoten können mit Bezeichnern versehen werden, um so leichter in Anfragen verwendet werden zu können. Die Operationen sind entweder durch durch die Anfragesprache Cypher,welche eine standardisierten Syntax mit mehreren vordefinierten Funktionen besitzt, limitiert oder durch die jeweilige verwendete Programmiersprache.Es wird das Einbetten weiterer Bibliotheken, welche weitere Funktionen oder Algorithmen stellen, unterstützt. Durch weitere Funktionen wie zum Beispiel die Bibliothek “APOC” [1] ist es möglich, Daten aus verschiedenen Formaten wie JSON,XSL oder XML in die Datenbank zu laden oder Daten aus anderen Web-APIs zu nutzen. Neo4j lässt sich in einem eingebetteten Modus oder einem Server Modus nutzen. Der eingebettete Modus dient der direkten Nutzung durch die Java Core API(Application programming interface) von Neo4j. Der Server Modus ermöglicht eine getrennte Ausführung von dem Code und dem bestehenden Neo4j System.

[1] https://neo4j-contrib.github.io/neo4j-apoc-procedures/ (17.05.19)

2.3 CAP und ACID unter Neo4j

Das CAP-Theorem charakterisiert das Verhalten einer Datenbank anhand von folgenden 3 Eigenschaften: Consistency(Konsistenz), Availability(Verfügbarkeit), Partitionstoleranz (Simon, 2000). Konsistenz beschreibt die Eigenschaft, dass die Daten in allen Partitionen zur selben Zeit dieselben Werte besitzen und das gleiche Verhalten aufweisen. Verfügbarkeit beschreibt die Möglichkeit zu jeder Zeit eine Anfrage an das System stellen zu können und auch zu jeder Zeit eine Antwort auf die gestellte Anfragen bekommen zu können. Partitionstoleranz gewährleistet, dass sich das Verhalten des System nicht verändert, wenn mehrere Partitionen von diesem System erstellt werden und alle Partitionen müssen das gleiche Verhalten aufweisen (Simon, 2000). Neo4j erfüllt die Bedingung der Verfügbarkeit und der Partitionstoleranz[16] und wird so als "AP-Database" bezeichnet.
Die ACID Eigenschaft setzt sich aus 4 Eigenschaften zusammen, die das Verhalten der Transaktionen einer Datenbank beschreiben (Haerder and Reuter, 1983). Atomicy(Atomisch) beschreibt, dass jede Transaktion einzeln betrachtet wird und nur fehlschlagen oder erfolgreich sein kann. Durch Consistency(Konsistenz) kann jede Transaktion nur valide Daten verwenden und den validen Zustand einer Datenbank nicht in einen nicht-validen Zustand überführen. Isolation erwartet, dass jede Transaktion unabhängig von einer parallel-laufenden Transaktion abläuft und keine dieser Transaktionen beeinflusst. Durability(Haltbarkeit) ist gegeben, wenn der Effekt einer Transaktion auf den Speicher ausgeübt wurde und auch bei einem Absturz des Systems beibehalten wird (Haerder and Reuter, 1983). Solange Neo4j auf einem einzelnen System läuft und nicht das High Aviability Feature der Enterprise Edition nutzt, ist es ACID konform (Holzschuher and Peinl, 2013). Das atomische und haltbare Verhalten wird durch das sogenannte write-ahead log(wat) versichert. Bei diesem Mechanismus werden alle Operationen einer Transaktionen nach dem Beenden der Transaktion in einem Log-File festgehalten, bevor diese den Speicher beeinflussen, so kann auch bei einem Absturz des System das Log-File genutzt werden um ein vorherige Transaktion zu wiederholen. Dieses Log-File wird auch für die High-Availability genutzt, welche es erlaubt die Datenbank in einem Netzwerk auf mehrere Systeme zu verwenden, dennoch ist es nicht mehr möglich ein ACID Verhalten zu gewährleisten, da keine absolute Garantie für ein atomisches und konsistente Verhalten gibt (Vukotic et al., 2015). Eine weitere versicherung für das atomische Verhalten bildet das Verhindern von Deadlocks innerhalb der Transaktion. Zur Verhinderung von Deadlocks wird "RWLock" verwendet, was eine Implementierung des Java "ReentrantReadWriteLock" für Neo4j ist. Dieser verwaltet alle Schreibsperren, die während einer Transaktion erstellt werden und versucht potentielle Deadlocks zu erkennen (Raj, 2015).

2.4 Neo4j als Datenbank management system

Ein Datenbank Managment System(DBMS), ist für die Verarbeitung von Anfragen verantwortlich und kann mit folgenden Teilen beschrieben werden: Eine Schnittstelle mit dem Nutzer, eine Anfragesprache, ein Anfragen Optimierer, eine database engine, eine storage engine ,eine transaction engine und operation features(Angles, 2012). Als Schnittstelle mit dem Nutzer stehen die Anwendungen "Neo4j Browser" unter Linux und "Neo4j Desktop" unter Windows zur Verfügung, zusätzlich besteht die Möglichkeit eine, von den Neo4j Treibern unterstützte Programmiersprache zu verwenden und direkt die Neo4j APIs zu nutzen. Die Anfragesprache für Neo4j

Browser und Neo4j Desktop ist "Cypher". Zum Optimieren von Anfragepläne wird der "Volcano Optimizer Generator" (Graefe and McKenna, 1993) verwendet (Francis et al., 2018). Die database engine wird durch die Core API dargestellt. Die storage engine durch die record files und die transaction unit wird durch das tranc-saction management dargestellt. Die operation features zum Wiederherstellen von Daten werden in den Transaktions-Log realisiert. Optional kann zu eventuellen Performanz Steigerung ein Cache verwaltet werden. In der Neo4j Enterprise Version ist es möglich, das DBMS auf mehrere Systeme in einem Netzwerk mittels High Availability Funktion auszulagern (Vukotic et al., 2015). Zusammengefasst wird die Architektur in 2.1

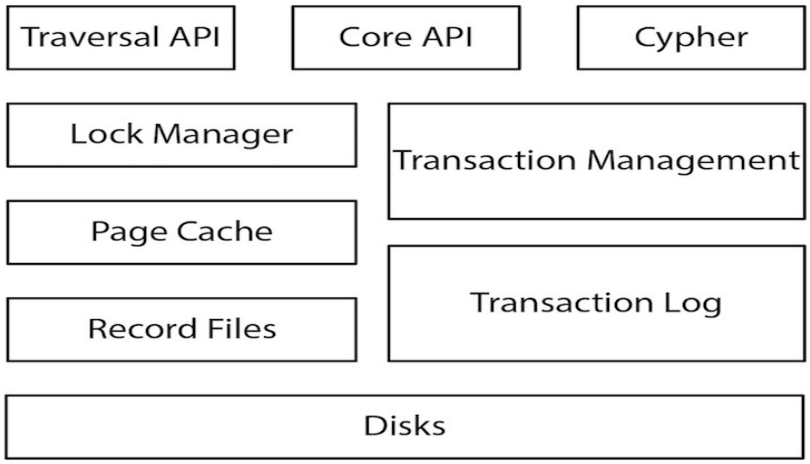

FIGURE 2.1: Allgemeine Architektur von Neo4j.

2.4.1 Cypher

Im Gegensatz zu den relationalen Datenbanken gibt es bei Graphdatenbanken keine standardisierte Anfragesprache, welche bei den meisten Graphdatenbanken Verwendung findet (Han et al., 2011). In Neo4j besteht seit 2000 die Möglichkeit die deklarative Anfragesprache "Cypher" zu verwendenden (Francis et al., 2018). Cypher wird von Neo4j entwickelt und wurde ursprünglich ausschließlich für die Neo4j

Datenbank verwendet. Seit Oktober 2015 findet Cypher als "openCypher" Gebrauch in anderen Datenbanken (Francis et al., 2018). Da für die Traversal API und die Java Core API Kenntnisse in Java bzw. einer durch die Neo4j-Treiber unterstützten Programmiersprachen benötigt werden, bildet Cypher eine Möglichkeit ohne diese Kenntnisse mit der Datenbank zu interagieren[16]. Die Syntax ist an SQL und Gremlin (Vukotic et al., 2015) orientiert. In Cypher wird ein Muster von dem Nutzer angegeben und alle Objekte, die dieses Muster erfüllen, können zurückgegeben werden. Die wichtigsten Schlüsselwörter sind:
Where: Dies hat die gleiche Funktion wie in SQL und spezifiziert Objekte durch einen Ausdruck.
Match: Dadurch wird das Muster spezifiziert indem Neo4j suchen soll. Beispielweise:

```
MATCH (p1:Person)-[:Friends]->(p2:Person)
WHERE p1.name= 'Peter'
RETURN p.name
```

, hier werden alle Personen-Knoten durchsucht, die mit einer "Friends"-Relation mit dem Personen-Knoten von "Peter" verbunden sind. p1 stellt ein Label für den Knoten vom Typ "Person" da, [:Friends] ist eine Relation vom Typ "Friends" und durch "->" wird angegeben, dass es sich um eine ausgehende Kante vom Knoten p1 handelt.
Return: Es wird angegeben, welche Objekte bzw. welche Attribute der Objekte, die das Muster erfüllen zurückgegeben werden sollen.
Delete: Wie in SQL wird ein Objekt oder mehrere Objekte aus der Datenbank entfernt.
With: Dadurch lassen sich in einer Anfrage Objekte manipulieren bevor sie zu einer weiteren Anfrage gegeben werden. Beispielsweise:

```
MATCH (p:Person{name: 'Peter'})
WITH COUNT(p) as count
RETURN count
```

Create: Erzeugt ein Objekt in der Datenbank Beispielsweise

```
Create (p:Person)
```

, hier wird ein Knoten vom Typ Person erzeugt. Es ist auch möglich mehrere Knoten mit den dazugehörigen Relationen zu erzeugen. Indizes für die Objekte oder Attribute von Objekten können ebenfalls über Create erzeugt werden.
Limit: Beschränkt die Menge, welche durch das Return-Statement zurückgegeben wird Beispielsweise

```
MATCH (p:person) RETURN p.name LIMIT 10
```

, hier werden nur die Namen von 10 Personen zurückgegeben
SUM/COUNT/AVG: Werden wie in SQL verwendet.
Der Nutzer beeinflusst so welche Daten durch das Musters gesucht werden. Der Nutzer besitzt keine Möglichkeit die Art der Berechnung zu beeinflussen. Cypher wird dadurch als Nutzerfreundlichere, aber auch weniger performante Alternative zu den APIs empfohlen (Vukotic et al., 2015). In den folgenden Experimenten wurde sowohl Cypher als als auch die Travese API für einen direkten Vergleich verwendet.

2.4.2 Java Core API

Als low-level Schnittstelle zu den Kernfunktionen von Neo4j ist dies die Möglichkeit mit der besten Laufzeit (Vukotic et al., 2015). Zur Verwendung dieser imperativen API sind weitreichende Programmierkenntnisse und Wissen über die Neo4j-Bibliotheken, sowie einer genauen Vorstellung über die Daten in dem Graph erforderlich. Wenn diese Kenntnisse gegeben sind, ist die API sehr flexible verwendbar und der Nutzer hat hohen Einfluss darauf, wie die Anfragen bearbeitet werden sollen und kann so eine optimale Berechnungsstrategie angeben (Vukotic et al., 2015). Die so produzierten Queries sind in meisten sehr lang. Eine Beispielhafte Traversion wäre:

```
private GraphDatabaseService graphdb;
public Node getSomeNode( Long ID){
        return graphDb.getNodeById(ID);
}
```

Die Funktion getSomeNode(Long ID) gibt den Knoten mit der gegeben ID zurück.

2.4.3 Traversal API

Diese deklarative API ist schneller als Cypher und langsamer als die Core API. Die Traversal API erlaubt einen high-level Zugriff auf Neo4j, welcher weniger abstrakt als die Core API ist und dennoch Programmierkenntnisse erfordert. Der Nutzer muss keine genaue Vorstellung von den Daten im Graphen haben.Es wird eine Beschreibung angegeben, wie eine Suche genau ausgeübt werden soll und kann diese dann auf einen Knoten anwenden.

```
private Traverser getFriends(final Node person)
{
TraversalDescription td = graphDb.traversalDescription()
        .breadthFirst()
        .relationships( RelTypes.RELATIONSHIP4, OUTGOING )
        .evaluator( Evaluators.atDepth(2) );
return td.traverse( person );
}
```

Zusammengefasst wäre diese Traversion: Suche alle Verbindungen von dem Anfangsnode über die Relation RELATIONSHIP4 bis zu Tiefe 2. Beim Traversieren kann der der Nutzer zwischen 2 grundsätzlichen Vorgehen wählen: Breadth-first und Depth-first, diese haben abhängig von der Struktur des Graphen eine unterschiedliche Laufzeit (Vukotic et al., 2015).
Breadth-First(Breiten-Suche) : Zuerst werden alle Knoten mit derselben Distanz betrachtet, danach werden alle Knoten mit der nächst höheren Distanz betrachtet, dies wird solange ausgeübt bis alle Knoten betrachtet wurden.
Depth-First(Tiefen-Suche) : Zuerst wird ein Nachbarknoten mit der geringsten Distanz betrachtet, ausgehend von diesem Nachbarknoten werden alle Nachbarn betrachtet. Wenn alle Nachbarn betrachtet wurden, wird ausgehend von Anfangsknoten ein weiterer Nachbar mit der geringsten oder nächst geringsten Distanz betrachtet und ausgehend von diesem werden erneut alle Nachbarn betrachtet. Dieses Vorgehen wird als default angenommen, wenn keine Vorgehen von dem Nutzer angegeben wird, wird dieses Vorgehen verwendet.

Wenn der Nutzer mit der Struktur der Daten in dem Graph vertraut ist, kann das ausgewählte eine erheblichen Performance Unterschied hervorbrigen[16]. Die Traversion kann auch bidirektional aufgeführt werden, dabei werden 2 Anfangsknoten angeben und die Suche wird so lange ausgeführt, bis es zu einer Kollision der beiden Knoten kommt. Wenn eine Kollision zwischen den beiden Knoten erkannt wird, ist damit zum Beispiel eine Verbindung der beiden Knoten bewiesen. Für die folgenden Experimente wurde bei den meisten Anfragen die Traversal API verwendet.

Chapter 3

Chapter Title Here

3.1 Main Section 1

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam ultricies lacinia euismod. Nam tempus risus in dolor rhoncus in interdum enim tincidunt. Donec vel nunc neque. In condimentum ullamcorper quam non consequat. Fusce sagittis tempor feugiat. Fusce magna erat, molestie eu convallis ut, tempus sed arcu. Quisque molestie, ante a tincidunt ullamcorper, sapien enim dignissim lacus, in semper nibh erat lobortis purus. Integer dapibus ligula ac risus convallis pellentesque.

3.1.1 Subsection 1

Nunc posuere quam at lectus tristique eu ultrices augue venenatis. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Aliquam erat volutpat. Vivamus sodales tortor eget quam adipiscing in vulputate ante ullamcorper. Sed eros ante, lacinia et sollicitudin et, aliquam sit amet augue. In hac habitasse platea dictumst.

3.1.2 Subsection 2

Morbi rutrum odio eget arcu adipiscing sodales. Aenean et purus a est pulvinar pellentesque. Cras in elit neque, quis varius elit. Phasellus fringilla, nibh eu tempus venenatis, dolor elit posuere quam, quis adipiscing urna leo nec orci. Sed nec nulla auctor odio aliquet consequat. Ut nec nulla in ante ullamcorper aliquam at sed dolor. Phasellus fermentum magna in augue gravida cursus. Cras sed pretium lorem. Pellentesque eget ornare odio. Proin accumsan, massa viverra cursus pharetra, ipsum nisi lobortis velit, a malesuada dolor lorem eu neque.

3.2 Main Section 2

Sed ullamcorper quam eu nisl interdum at interdum enim egestas. Aliquam placerat justo sed lectus lobortis ut porta nisl porttitor. Vestibulum mi dolor, lacinia molestie gravida at, tempus vitae ligula. Donec eget quam sapien, in viverra eros. Donec pellentesque justo a massa fringilla non vestibulum metus vestibulum. Vestibulum in orci quis felis tempor lacinia. Vivamus ornare ultrices facilisis. Ut hendrerit volutpat vulputate. Morbi condimentum venenatis augue, id porta ipsum vulputate in. Curabitur luctus tempus justo. Vestibulum risus lectus, adipiscing nec condimentum quis, condimentum nec nisl. Aliquam dictum sagittis velit sed iaculis. Morbi tristique augue sit amet nulla pulvinar id facilisis ligula mollis. Nam elit libero, tincidunt ut aliquam at, molestie in quam. Aenean rhoncus vehicula hendrerit.

Chapter 4

Die 2 Modi

4.1 Der eingebettete Modus

Mittels Treiber lassen sich verschiedene Programmiersprachen verwenden, um Neo4j anzusteuern. Alle Klassen und Prozesse werden direkt in der Java virtual machine (JVM) ausgeführt und mittels des GraphDataBaseService werden die benötigten Neo4j Funktionen aufgerufen und auf den Speicher angewenden. Dies wird in 4.2 dargestellt.

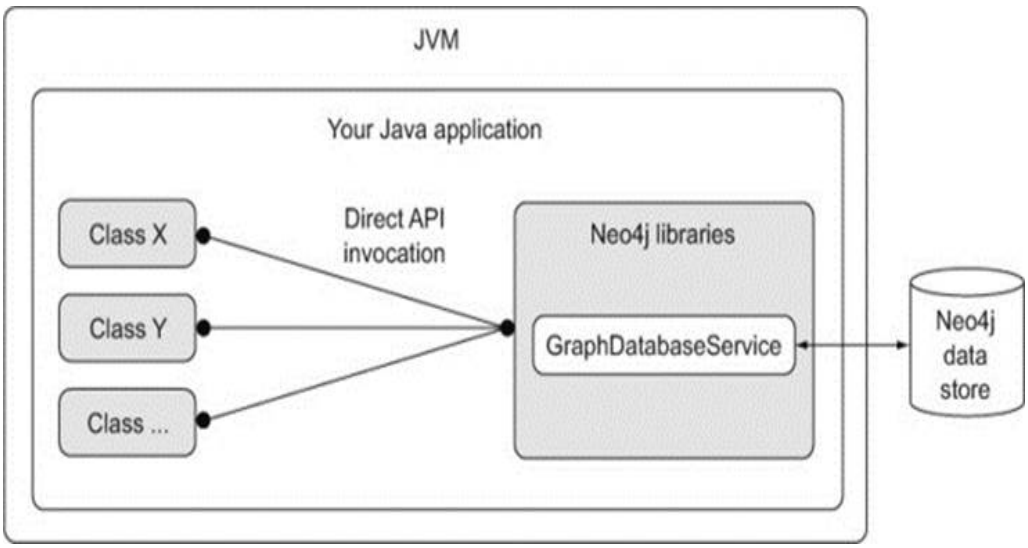

FIGURE 4.1: Allgemeiner eingebetteteter Modus von Neo4j.

4.2 Der Server Modus

In diesem Modus bilden die Klassen und Operationen einen eigenen Prozess und sind isoliert von der Anwendung. Die Kommunikation zwischen der Anwendung des Nutzers und der JVM findet durch die REST API[17] statt[16] und wird mittels HTTP (Hypertext Transfer Protocol) übertragen.Dies wird in 4.2 verdeutlicht.

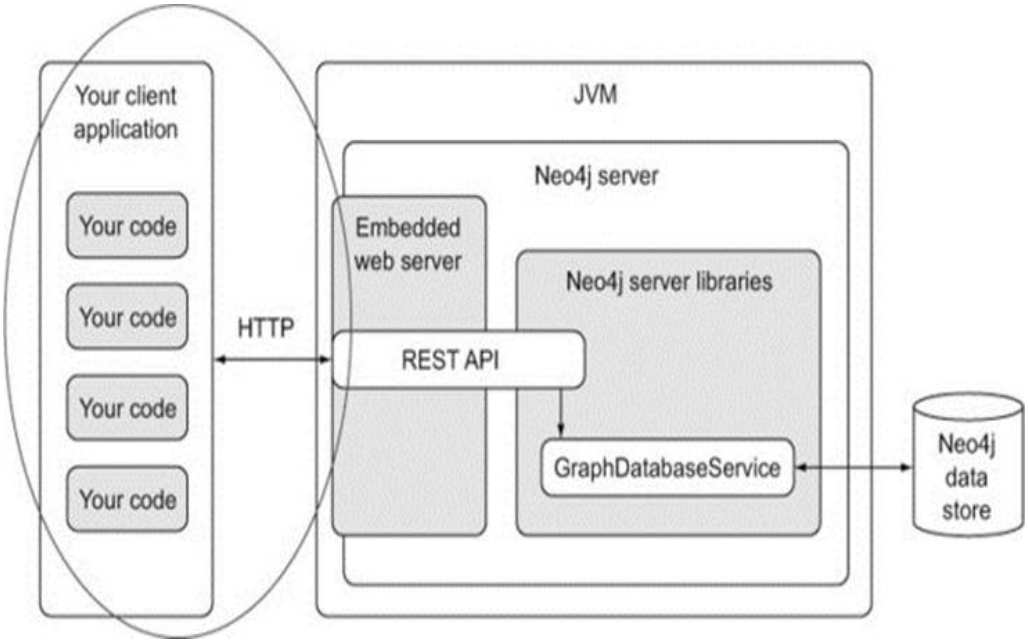

FIGURE 4.2: Allgemeiner Servermodus von Neo4j.

Bibliography

Angles, Renzo (2012). "A comparison of current graph database models". In: *2012 IEEE 28th International Conference on Data Engineering Workshops*. IEEE, pp. 171–177.

Bondy, John Adrian, Uppaluri Siva Ramachandra Murty, et al. (1976). *Graph theory with applications*. Vol. 290. Citeseer.

Codd, Edgar F (1981). "Data models in database management". In: *ACM Sigmod Record* 11.2, pp. 112–114.

Francis, Nadime et al. (2018). "Cypher: An evolving query language for property graphs". In: *Proceedings of the 2018 International Conference on Management of Data*. ACM, pp. 1433–1445.

Graefe, Goetz and William J McKenna (1993). "The volcano optimizer generator: Extensibility and efficient search". In: *ICDE*. Vol. 93, pp. 209–218.

Haerder, Theo and Andreas Reuter (1983). "Principles of transaction-oriented database recovery". In: *ACM computing surveys (CSUR)* 15.4, pp. 287–317.

Han, Jing et al. (2011). "Survey on NoSQL database". In: *2011 6th international conference on pervasive computing and applications*. IEEE, pp. 363–366.

Holzschuher, Florian and René Peinl (2013). "Performance of graph query languages: comparison of cypher, gremlin and native access in Neo4j". In: *Proceedings of the Joint EDBT/ICDT 2013 Workshops*. ACM, pp. 195–204.

Miller, Justin J (2013). "Graph database applications and concepts with Neo4j". In: *Proceedings of the Southern Association for Information Systems Conference, Atlanta, GA, USA*. Vol. 2324. S 36.

Raj, Sonal (2015). *Neo4j high performance*. Packt Publishing Ltd.

Simon, Salomé (2000). "Brewer's cap theorem". In: *CS341 Distributed Information Systems, University of Basel (HS2012)*.

Strauch, Christof, Ultra-Large Scale Sites, and Walter Kriha (2011). "NoSQL databases". In: *Lecture Notes, Stuttgart Media University* 20.

Vicknair, Chad et al. (2010). "A comparison of a graph database and a relational database: a data provenance perspective". In: *Proceedings of the 48th annual Southeast regional conference*. ACM, p. 42.

Vukotic, Aleksa et al. (2015). *Neo4j in action*. Vol. 22. Manning Shelter Island.