Abstrakt zum Vortrag im Oberseminar

Graphdatenbanken

Gero Kraus
HTWK Leipzig

14. Juli 2015

1 Motivation

Zur Darstellung komplexer Beziehungen bzw. Graphen sind sowohl relationale als auch NoSQL-Datenbanken weniger geeignet. Vor allem relationale Datenbanken wurden zudem nie für solch einen Anwendungsfall entwickelt, diese haben ihren Ursprung in der Abbildung von Formularen und Tabellenstrukturen.

Bei der Darstellung von Graphen treten unter anderem folgende Probleme auf:

Komplexität Geschäftslogik und Fremdschlüssellogik wird vermischt.

Foreign Key Constraints führen zu Entwicklungsoverhead Das Überprüfen, dass diese innerhalb der Anwendung auch eingehalten werden, ist aufwändig.

Schema führt zu vielen NULL-Werten Das starre Schema der relationalen Datenbank führt zu vielen NULL-Werten, welche innerhalb der Anwendung abgefangen werden müssen.

Viele Joins nötig Um Beziehungen wiederherzustellen, sind viele aufwändige Joins nötig.

Anwendung hat großen Einfluss auf Schemadesign Je nachdem, welcher Anwendungsfall abgedeckt werden soll, fällt das Schema, unter anderem aus Performancegründen, anders aus.

Beziehungen offenbaren keine Semantik Die Semantik von Beziehungen ist im Schema nicht ersichtlich.

Auch alle Typen von NoSQL-Datenbanken, also Dokumentenorientierte, Spaltenorientierte und Key-Value-Datenbanken haben dasselbe Grundproblem: Diese müssen ebenso eine Form von Fremdschlüssellogik einführen, um Beziehungen darzustellen.

1.1 Lösung: Graphdatenbank

Das Problem aller Datenbanken ist also, dass Semantische Zusammenhänge nur implizit vorhanden sind und für die Datenbank nicht erkennbar sind. Diese müssen deshalb bei Abfragen durch Joins aufwändig rekonstruiert werden.

Die Lösung für dieses Problem ist die Verwendung einer Datenbank, welche explizit für die Darstellung von Beziehungen entworfen wurde, eine Graphdatenbank. Graphdatenbanken verwenden ein Format, welches Informationen über Beziehungen explizit speichert. Meist ist dies ein sog. Labeled Property Graph (s. Abb. 1). Dieser ist ein gerichteter Graph mit benannten Knoten, welche Attribute besitzen können. Kanten sind ebenso benannt und können auch Attribute beinhalten. Die Struktur ist im allgemeinen variabel, das heißt, schemafrei.

Da innerhalb eines solchen Graphen Kanten traversiert werden können, ist die Suche von Beziehungen unter den Knoten ungleich schneller als bei Relationalen/NoSQL-Datenbanken. [2][5, S.21]

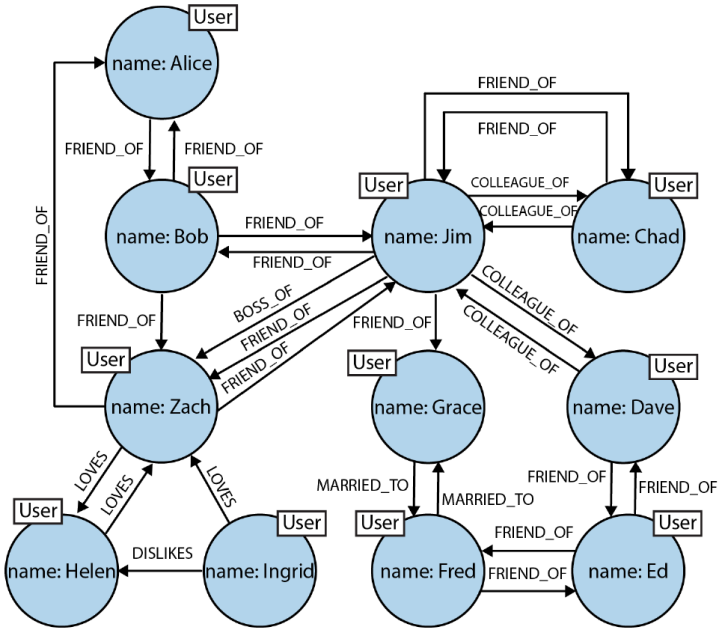

Abbildung 1: Beispiel eines Labeled Property Graph [5]

2 Typen von Graphdatenbankmanagmentsystemen

Graphdatenbankmanagmentsysteme im Allgemeinen stellen dem Nutzer ein Graphenmodell (s. Abschnitt 3 auf der nächsten Seite) zur Verfügung, auf welchem die typischen CRUD (Create, Read, Update, Delete)-Methoden ausgeführt werden können. Die

überwiegende Mehrheit der Graphdatenbanken ist für OLTP optimiert, also ausgelegt auf Transaktionsperformance, Transaktionsintegrität und Operative Verfügbarkeit.

Graphdatenbanken können grob durch die zugrundeliegende Speicher- und Abarbeitungstechnik unterschieden werden:

Die Graphdatenbank kann den Graph nativ, also in einem Graphenmodell speichern oder aber nicht nativ, also zum Beispiel in einer relationalen oder NoSQL-Datenbank.

Außerdem kann die Abarbeitungsengine nativ arbeiten, was bedeutet, dass diese direkt auf dem Graphen traversiert. Dies erfordert, dass direkt in den Knoten Informationen über deren Nachbarn gespeichert ist. Diese Form der Abarbeitung nennt man auch *Indexfreie Adjazenz*. Die Alternative hierzu ist die nicht native Abarbeitung, wobei die Nachbarschaftsbeziehungen dann z.B. über einen Index hergestellt werden. [5, S. 205ff]

3 Grapharten

Neben dem Labeled Property Graph existieren noch weitere Arten von Graphen:

Hypergraph Der Hypergraph ist ähnlich dem Labeled Property Graph, allerdings können hier Kanten eine beliebige Anzahl an Knoten verbinden.

RDF RDF ist mehr eine Beschreibungsform von Graphen als eine eigene Graphart und ist grundlegender Bestandteil des Semantic Webs. Hierbei werden Beziehungen durch eine Subjekt-Prädikat-Objekt-Struktur, sog. Tripel, beschrieben. Durch diese einfache Beschreibungsform ist die Speicherart nicht nativ. [4]

4 Beispiel: Neo4j

Neo4j ist eine der populärsten Graphdatenbanken [1]. Das Datenmodell basiert auf dem Property Graph, ist schemafrei und gut skalierbar. Die Community-Edition der Datenbank ist Open Source.

Neo4j besitzt zur Abfrage sowohl eine REST-API als auch mit *Cypher* eine eigens entwickelte Abfragesprache, auf welche im Weiteren etwas genauer eingegangen wird.

4.1 Die Abfragesprache Cypher

Cypher ist eine einfache Abfragesprache für Neo4j, welche nach dem Prinzip des Pattern-Matching arbeitet. Der Nutzer stellt Anfragen an das System, indem er die abzufragenden Daten via einer Syntax, welche an ASCII-Art erinnert, beschreibt. Zu Abb. 2 wäre die Repräsentation in dieser Syntax:

```
(emil)<-[:KNOWS]-(jim)-[:KNOWS]->(ian)-[:KNOWS]->(emil)
```

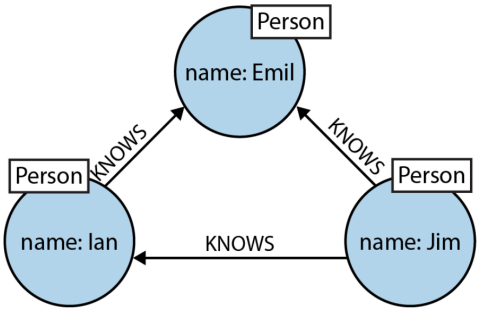

Abbildung 2: Einfacher Graph zur Abbildung von Personenbeziehungen [5]

Eine einfache Abfrage wäre zum Beispiel folgende:

```
MATCH (a:Person {name:'Jim'})-[:KNOWS]->(b)-[:KNOWS]->(c),
(a)-[:KNOWS]->(c)
RETURN b, c
```

5 Interessante Anwendungen

5.1 Häufige Anwendungsfälle

Es existieren unterschiedlichste Fälle, in denen eine Graphdatenbank von Nutzen ist. Beispiele hiervon sind:

Sozial Soziale Beziehungen können sehr gut als Graph dargestellt werden.

Geoinformationen Auch Geoinformationen eignen sich hervorragend zur Darstellung als Graph. Zum einen existieren spezielle Datenstrukturen, wie z.B. der R-Baum, zum anderen sind allein schon Adressdaten hierarchisch aufgebaut.

5.2 Nützliche Algorithmen

innerhalb einer Graphdatenbank sind natürlich alle auf Graphen arbeitenden Algorithmen besonders interessant. Hierzu zählen z.B.:

Tiefensuche Sinnvoll, wenn einem Pfad gefolgt werden soll, um unterschiedliche Datenfragmente zu finden

Breitensuche Sinnvoll für systematische Suche oder Wegfindung

Dijkstra-Algorithmus Kürzeste-Wege-Algorithmus

A*-Algorithmus Kürzeste-Wege-Algorithmus mit Heuristik, basierend auf Djikstra

6 Weitere Systeme

Es existieren viele verschiedene Graphdatenbanken, meist von relativ unbekannten Unternehmen. Allerdings haben auch Große Datenbankhersteller Systeme zur Graphrepräsentation, so existiert für die Datenbank Oracle Enterprise Edition das separat lizenzierbare System *Spatial and Graph*, welches als Raum- und Graphdatenbank fungiert.

Mit dem darin enthaltenen *Network Data Model* wird dem Nutzer ein Property Graph-Modell zur Verfügung gestellt, welches allerdings intern relational, also nicht nativ gespeichert wird. [3]

Literatur

[1] *DB-Engines Ranking - popularity ranking of graph DBMS*. URL: http://db-engines.com/en/ranking/graph+dbms (besucht am 08. 05. 2015).

[2] Michael Hunger. *Neo4j 2.0 Eine Graphdatenbank für alle*. German. Frankfurt am Main: entwickler.press, 2014. ISBN: 978-3-86802-315-2 3-86802-315-1. URL: http://nbn-resolving.de/urn:nbn:de:101:1-2014032620976.

[3] *Oracle Spatial and Graph*. URL: http://www.oracle.com/technetwork/database/options/spatialandgraph/overview/index.html (besucht am 07. 05. 2015).

[4] *RDF 1.1 Primer*. URL: http://www.w3.org/TR/2014/NOTE-rdf11-primer-20140624/ (besucht am 07. 05. 2015).

[5] Ian Robinson, Jim Webber und Emil Eifrem. *Graph databases*. " O'Reilly Media, Inc.", 2013. URL: info.neo4j.com/rs/neotechnology/images/Graph_Databases_2e_Neo4j.pdf.