Open Source Graphdatenbanken – Konzepte und Klassifikation

Olaf Herden, Kevin Redenz

Fakultät Technik
Studiengang Informatik
Duale Hochschule Baden-Württemberg Campus Horb
Florianstr. 15
72160 Horb
o.herden@hb.dhbw-stuttgart.de
k.redenz@hb.dhbw-stuttgart.de

Abstract: In der jüngsten Vergangenheit hat es im Datenbankmarkt einen Paradigmenwechsel gegeben. Haben über viele Jahre relationale Datenbanken den Markt klar beherrscht und jede Art von Daten abgespeichert („One Size Fits All"), so sind unter dem Begriff NoSQL (Not only SQL) eine ganze Reihe von neuartigen Systemen entstanden. Eine besondere Kategorie von NoSQL-Datenbanken sind Graphdatenbanken. Eine Vielzahl von, meistens unter einer Open Source Lizenz laufenden, Systemen mit zum Teil stark unterschiedlichen Eigenschaften ordnet sich dieser Gruppe zu. Wesentliche Konzepte dieser Systeme werden in diesem Beitrag beschrieben. Weiterhin haben wir als Orientierungshilfe in der unübersichtlichen und sich rasch ändernden Welt der Graphdatenbanken ein Klassifikationsschema erstellt und auf einige Vertreter angewandt.

1 Einleitung

Seit Beginn der professionellen Datenverarbeitung finden intensive Untersuchungen statt, wie Daten am praktikabelsten persistent gehalten werden können, d.h. dauerhaft über die Laufzeit des aktuellen Programms hinaus abgespeichert werden können. Nach ersten Ansätzen, die Daten in Dateien zu speichern, kam man schnell zur Erkenntnis mit Datenbanksystemen eigene Software hierfür zur Verfügung zu stellen. Entscheidend dabei ist das Datenmodell, welches festlegt, wie die Daten gespeichert werden und welche Operationen auf ihnen zulässig sind. Zunächst dominierten in den 60er Jahren das hierarchische und Netzwerk-Modell (synonym: CODASYL-Modell), bevor in [Co70] das relationale Modell vorgeschlagen wurde. Wenige Jahre später standen die ersten kommerziellen Systeme zur Verfügung und in den 80er Jahren durchdrangen diese den Markt.

In den 90er Jahren kamen weitere Datenbanktypen an den Markt: Mit dem Aufkommen objektorientierter Sprachen und Modellierungsmethoden objektorientierte Datenbanken, durch zunehmenden Datenaustausch und das XML-Format bedingt XML-Datenbanken und aufgrund von Data Warehouses und analytischen Umgebungen Datenbanken mit multidimensionalem Datenmodell. Die jungen Produkte in allen drei Sparten konnten jedoch am Markt nicht gegen die mittlerweile sehr etablierten relationalen Datenbanken bestehen und verschwanden mehr oder weniger bald wieder. Vielmehr wurden in den relationalen Systemen Erweiterungen zu den genannten Aspekten vorgenommen. Damit galt das „One Size Fits All"-Paradigma, mit relationalen Systemen ein Typ für alle Zwecke. Dadurch ergab sich über eine Dekade ein sehr stabiler Markt, in dem die existierenden Systeme weitere Reife erlangten.

Das Abdecken aller Eigenschaften durch relationale Systeme stößt langsam an Grenzen und in jüngerer Vergangenheit ist ein Paradigmenwechsel [St10, St11] zu beobachten, der zur Bewegung NoSQL (Not Only SQL) geführt hat.

Zeitgleich trat auch der Begriff Big Data auf die Bildfläche, wobei häufig falscherweise Big Data und NoSQL in einem Atemzug genannt werden. Dies ist aber nicht korrekt wie wir in Abschnitt 2 darlegen werden. Ebenso sei an dieser Stelle ausdrücklich erwähnt, dass Graphdatenbanken keine Erfindung der letzten Jahre oder der NoSQL-Bewegung sind, sondern es schon seit einigen Jahrzehnten Ansätze in diesem Umfeld gibt. Als erster publizierter Ansatz ist hier das System R&M [RM75] zu nennen. Hierin wurde ein semantisches Netzwerk zum Speichern von Metadaten vorgeschlagen, weil damalige Systeme typischerweise die Semantik der Datenbank nicht berücksichtigten. Den Startschuss einer lebhaften Entwicklungen diverser Vorschläge für Konzepte und Prototypen gab das Logische Datenmodell [KV84], in dem ein explizites Graphdatenmodell relationales, hierarchisches und Netzwerkmodell generalisieren sollte. Die hierauf aufbauenden Ansätze sollen hier nicht alle einzeln erwähnt werden, einen sehr guten Überblick gibt [AG08]. Mitte der 90er Jahre ebbte die Welle dann ab, parallel dem Ende der diversen Vorschläge objektorientierter Datenbanken. In den letzten Jahren entstand dann eine neue Welle von Systemen, wobei verschiedene Hersteller oder Communities sehr verschiedene Konzepte zur Realisierung einer Graphdatenbank verfolgen. Um hier den Überblick zu behalten und realistisch die Fähigkeiten und Grenzen von Systemen einschätzen zu können, wird in diesem Beitrag eine Klassifikationsschema für Graphdatenbanken vorgeschlagen und auf wichtige, aktuelle Systeme angewendet.
Der Rest des Papers ist folgendermaßen gegliedert: Im folgenden Abschnitt wird auf die beiden Begriffe Big Data und NoSQL eingegangen. Abschnitt 3 stellt unser Klassifikationsschema für Graphdatenbanken vor und wendet es auf existierende Systeme an. Der Beitrag schließt mit einer Zusammenfassung und einem Ausblick.

2 Big Data, NoSQL und Graphdatenbanken

Dieser Abschnitt soll die Begriffe Big Data und NoSQL definieren, den Bezug zueinander herstellen und die Einordnung von Graphdatenbanken in diesem Kontext vornehmen.

2.1 Big Data

Unter dem Begriff Big Data werden Daten verstanden, die in sehr großer Menge anfallen und dabei in den Aspekten Erfassung, Speicherung, Suche, Verteilung und Analyse existierende DB-Systeme an ihre Grenzen bringen.

Big Data werden häufig maschinell erzeugt, z.B.

- Applikations- und Web-Logs
- Netzwerk Monitoring
- Sensoren (z.B. RFID)
- Barcodeerfassung
- Intelligente Stromzähler (SmartMeter)

Ein Beispiel aus dem letztgenannten Anwendungsszenario soll den Begriff „Big“ verdeutlichen [Kö11]. Bei einem kalifornischen Energieversorger sind bisher 1 bis 2 Ablesungen pro Jahr notwendig. Um die Daten intelligenter Stromzähler für eine bedarfsgerechte Energienutzung verwenden zu können, sind Ablesungen im 15-Minuten-Takt notwendig, in der Endausbaustufe werden gar Ablesungen im Minutentakt angestrebt. Abbildung 1 stellt das Datenwachstum graphisch dar.

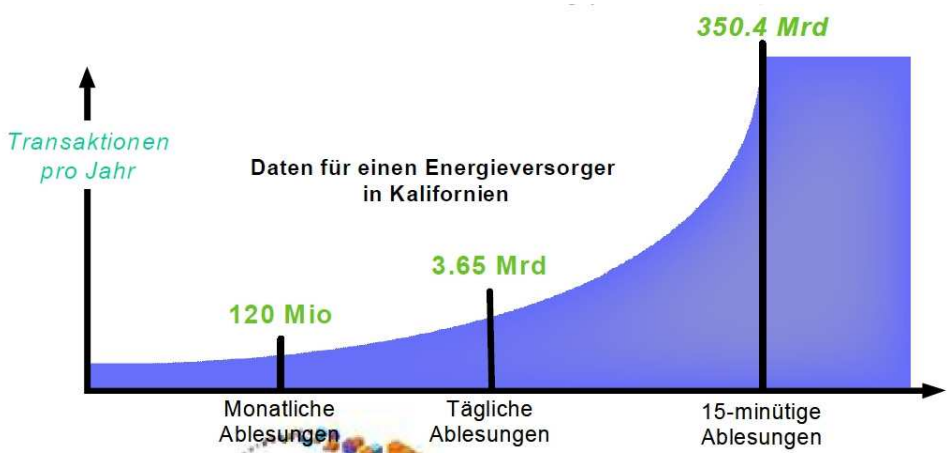

Abbildung 1: Zeilen- und spaltenorientierte Speicherung

Big Data können aber auch durch Personen erfasst werden, z.B. in Blogs, digitalen sozialen Netzwerken oder Online-Foren.

Big Data können dabei sowohl strukturiert als auch halbstrukturiert als auch unstrukturiert sein.

Als letzter Aspekt für Big Data sei hier noch die oft notwendige zeitnahe Verarbeitung der Daten erwähnt, um entsprechenden Nutzen aus den Daten ziehen zu können.

2.2 NoSQL

Unter dem Begriff NoSQL (Not only SQL) werden alle Datenbanken zusammengefasst, die nicht dem relationalen Modell folgen [EF+10, Ti11a, RW12]. Dadurch ergibt sich eine Sammlung sehr heterogener Lösungen. In der Literatur [EF+10] werden üblicherweise folgende Kategorien unterschieden:

- Schlüssel-Wert-(Key-Value-)basierte Datenbanken: Bei diesen Datenbanken wird unter einem eindeutigen Schlüssel ein einzelner Wert gespeichert, wobei prinzipiell die Struktur des Wertes von der Datenbank nicht interpretiert wird. Einige Lösungen weichen hiervon ab und bieten z.B. das Speichern von Listen und Mengen (Redis) oder die Gruppierung von Schlüssel-Wert-Paaren zu Domänen (Amazon Simple DB).

  Großer Pluspunkt der Schlüssel-Wert-basierten Systeme ist die einfache Verteilung großer Datenmengen auf mehrere Rechner. Demgegenüber wird die Strukturierung von Daten jedoch vernachlässigt.

  Bekannte Vertreter dieser Kategorie sind Amazon Simple DB [Am12], Chordless [Ch12], Riak [Ri12] und Redis [Re12].

- Wide Column Stores basieren auf Tabellen und bieten die Möglichkeit in einer Zeile mehrere Attribute zu speichern. Attributname und –wert bilden ein Schlüssel-Wert-Paar, das als Spalte bezeichnet wird. Wesentlicher Unterschied zu relationalen Tabellen ist die Nicht-Existenz eines Schemas, d.h. zu einer Zeile können beliebige Daten hinzugefügt bzw. weggelassen werden. Daraus ergibt sich auch das primäre Anwendungsgebiet von dünn besetzten Daten. In diese Kategorie fallen u.a. die Systeme Google BigTable [CD+06], Apache Cassandra [He10, Ca12], Apache HBase [Ge11, Hb12] und Hypertable [Hy12].

- In dokumentenorientierten DB-Systemen werden die Daten in Form semi-strukturierter Dokumente gespeichert, wobei Dokumente mithilfe eines Schlüssels gespeichert und geladen werden. Abfragen nach Dokumenten mit bestimmten Bestandteilen sind möglich. Anwendungsgebiet sind (semi-)strukturierte Inhalte, die Skalierbarkeit hingegen ist gegenüber den beiden ersten vorgestellten Ansätzen geringer. Die beiden Hauptvertreter dieser Kategorie sind CouchDB [ALS10, WK11, Co12] und MongoDB [CD10, Ba11, Mo12].

- Graphdatenbanken sind auf das Speichern und effiziente Traversieren von Graphen ausgelegt.

3 Klassifikation von Graphdatenbanken

In diesem Abschnitt wird ein Schema zur Klassifikation von Graphdatenbanken vorgestellt. Dieses teilt sich in mehrere Kategorien auf, die in den folgenden Teilabschnitten beschrieben werden. Angewendet wird dies Schema auf die Open Source Graphdatenbanken Neo4J, Sones GraphDB, Infogrid, HyperGraphDB und VertexDB. Zum ergänzenden Vergleich werden die beiden kommerziellen Vertreter InfiniteGraph und DEX hinzugezogen.

3.1 Grundlegende Informationen

In dieser Kategorie werden die Basisinformationen über das jeweilige Produkt dargestellt. Dieses sind Informationen über den Hersteller und seine Webseite, seit wann es das Produkt gibt, die untersuchte (aktuelle) Version, angebotenes Open Source Lizenzmodell, unterstützte Plattformen und Implementierungssprache.

Tabelle 1 fasst die Informationen für die untersuchten Datenbanken zusammen.

Grundlegende Eigenschaften							
	Neo4J	**Sones GraphDB**	**Infogrid**	**HyperGraphDB**	**Vertex DB**	**InfiniteGraph**	**DEX**
Anbieter	Neo Technology	Sones GmbH	NetMesh	Kobrix Software Inc.	Steve Dekorte et.al.	Objectivity Inc.	Sparsity Technologies
Erste Version	2007	2010	???	2010	2009	2010	2007
Untersuchte Version	1.6	2.0	2.9.5	1.1	k.A.	2.0	4.2
Lizenz, Open Source	GPLv3 (Community Edition) AGPLv3 (Advanced und Edition)	AGPLv3	AGPLv3	LGPL	Revised BSD	Nein	Nein
Lizenz, kommerziell	Java	Als SaaS	Ja	Nein	Nein	Ja (EULA und kostenpflichtig)	PersonalEvaluation und kommerziell
Plattform							
Windows	Ja	Plattformunabhängig	Ja	Ja	k.A.	Ja	Ja
Linux	Ja		Ja	Ja	k.A.	Ja	Ja
Unix	Ja		Ja	Nein	k.A.	Nein	Ja
MacOS	Ja		Ja	Ja	k.A.	Ja	Nein
Andere	Ja (JVM-basiert)		Ja (JVM-basiert)	Nein	k.A.	Nein	Nein
Geschrieben in	Java	C#	Java EE	Java	C	Java, Kern in C++	Java, .NET, C++ im Kern

Tabelle 1: Kategorie Grundlegende Informationen

Datenmodell							
	Neo4J	**Sones GraphDB**	**Infogrid**	**HyperGraphDB**	**InfiniteGraph**	**DEX**	**Vertex DB**
Gerichtete Kanten	Ja	Ja	Ja	Ja	Ja	Ja	Ja
Ungerichtete Kanten	Nein	?	Ja	Ja		Ja	Ja
Knoten-Property	Ja	Ja	Ja	Ja	Ja	Ja	Ja
Kanten-Property	Ja	Ja	Nein	Teilweise ([0..1])	Ja	Ja	Nein
Schema	Schemalos	Schemalos	Wählbar	Schemalos		Schemalos	Schemalos
Knotentypisierung	Nein	Ja	Ja	Ja	Ja	Ja	Ja
Kantentypisierung	Ja	Ja	Ja	Ja	Ja	Ja	Ja
Sonstige Eigenschaften	Schlingen	Hyperkanten	Schlingen	Schlingen, Hypergraphen	---	Schlingen	---

Tabelle 2: Kategorie Datenmodell

3.2 Datenmodell

In dieser Kategorie werden die von den Produkten unterstützten Grapheigenschaften zusammengefasst.

Als elementare/inhärente Eigenschaften müssen alle Graphdatenbanken Knoten und Kanten erzeugen können.

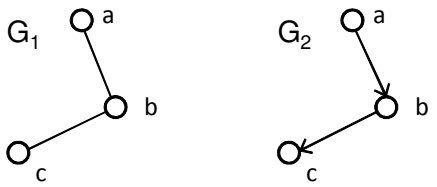

Abbildung 2: Beispiel ungerichteter und gerichteter Graph

Die Knoten des Graphen aus Abbildung 2 werden durch den Pseudocode in Listing 1erzeugt.[1]

```
Node a = new Node();
Node b = new Node();
Node c = new Node();
```

Listing 1: Pseudocode zur Erzeugung von Knoten

Zum Anlegen der Kanten gibt es verschiedene Möglichkeiten: Manche Systeme unterstützen gerichtete und ungerichtete Kanten, andere nur einen Typ. Auch beim Definieren der Beziehungen gibt es Unterschiede. In der Regel werden durch eine Methode `setEdgeTo` implizit gerichtete Kanten erzeugt, der ungerichtete Fall muss durch Definition der Hin-und Rückrichtung simuliert werden, wie es Listing 2 zeigt.

```
// Für Graph G2
a.setEdgeTo(b);
b.setEdgeTo(a);
c.setEdgeTo(b);
// Für Graph G1
a.setEdgeTo(b);
b.setEdgeTo(a);
b.setEdgeTo(c);
c.setEdgeTo(b);
```

Listing 2: Pseudocode zum Erzeugung von Kanten

[1] In diesem Pseudocodebeispiel wurde eine Notation mit Konstruktor/Aufrufen gewählt. Viele Systeme realisieren das Erzeugen neuer Knoten nach dem Factory-Pattern.

Alternativ zur impliziten Definition von Kanten kann es auch Methoden `setDirectedEdgeTo` und `setUndirectedEdgeTo` geben, die das Definieren gerichteter wie ungerichteter Kanten gestatten. Für den Graphen G_2 sehe die Kantendefinition dann wie in Listing 2 aus (mit dem Unterschied, das die Methode nun `setDirectedEdgeTo` heißt), für G_1 kann die in Listing 3 beschriebene kürzere Form verwendet werden.

```
a.setUndirectedEdgeTo(b);
b.setUndirectedEdgeTo(c);
```

Listing 3: Pseudocode zum Erzeugung ungerichteter Kanten

In den meisten Anwendungen tragen sowohl die Knoten als auch die Kanten Informationen, in der Graphentheorie spricht man hier von knoten- und/oder kantenmarkierten Graphen [Ti11b]. Im Umfeld von Graphdatenbanken hat sich hier das sog. Property-Graph-Modell [EF+10] etabliert. Neben den Knoten und Kanten gibt es als dritten Grundtyp Properties. Dies sind Schlüssel-Wert-Paare (Key-Value-Pairs) und sie werden in die Knoten bzw. Kanten eingebettet. Im einfachsten Falle dürfen dabei den Knoten und Kanten beliebige Properties zugeordnet werden. In manchen Systemen muss oder kann a priori definiert werden, welche Properties zulässig sind. Weiterhin kann für Kantenproperties einschränkend festgelegt werden, zwischen welchen Knoten sie verwendet werden dürfen.

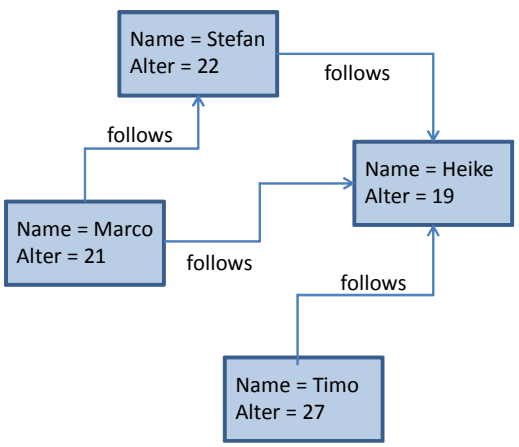

Abbildung 4: Beispiel Digitales soziales Netzwerk

Dazu zeigt Abbildung 4 einen Graphen aus einem digitalen sozialen Netzwerk und Listing 4 die freie und die vorher definierte Property-Zuordnung.

```
Node myFirstNode = new Node();
Node mySecondNode = new Node();
Node myThirdNode = new Node();
Node myFourthNode = new Node();
```

```
// Freie Property-Zuordnung
myFirstNode.setProperty("Name","Stefan");
myFirstNode.setProperty("Alter","22");
. . .
// Vorher definierte Properties
PropertyType name = new PropertyType("Name");
PropertyType alter = new PropertyType("Alter");
myFirstNode.setPropertyType(name);
myFirstNode.setPropertyType(alter);
myFirstNode.setProperty("Name","Stefan");
myFirstNode.setProperty("Alter","22");
. . .
```

Listing 4: Definieren und Zuordnen von Properties

Neben den Properties ist es möglich Knoten und/oder Kanten zu typisieren, d.h. sie mit einem Identifier bzw. Label zu versehen. Während Identifier typischerweise automatisch vergeben werden, werden Label explizit gesetzt. Beispielsweise ist in Abbildung 4 „follows“ ein Label für die Kanten.

Alle weiteren statischen Grapheigenschaften werden in einem Punkt zusammengefasst. Denkbar sind hier beispielsweise Schlingen oder Hyperkanten [Ti11b].

Tabelle 2 fasst die Informationen für die untersuchten Datenbanken zusammen.

3.3 DB-Eigenschaften

In diesem Abschnitt werden die elementaren DB-Eigenschaften von Graphdatenbanken untersucht. Zentraler Aspekt ist hierbei die Transaktionsfähigkeit, wobei wir zwischen ACID-Transaktionen und weiter gehenden Konzepten unterscheiden. Zu letzteren gehören insbesondere das Konzept des MVCC (Multi Version Concurrency Control) [Re78, BG81] und Transaktionaler Speicher [HM93, HLR10].

Zweiter wichtiger Punkt ist die Form der Persistenzgestaltung. Hierbei kann zwischen einer proprietären Lösung, der Verwendung existierender Lösungen anderer Anbieter oder einem generischen Ansatz unterschieden werden. Bei letztgenanntem kann nochmals unterschieden werden zwischen einer Low-Level-Realisierung (z.B. Speicher-API von MySQL) oder einer High-Level-Realisierung (z.B. über SQL-Schnittstelle einer relationalen Datenbank).

Als Hauptspeicherkonzepte lassen sich die Aspekte der Existenz eines Caches (inkl. der Verwaltung mit Ein-/Auslagerungsstrategien) und die In-Memory-Fähigkeit betrachten. Letztere bedeutet die Möglichkeit, den gesamten DB-Inhalt im Hauptspeicher verwalten zu können.

Zur Unterstützung des effizienten Zugriffs sind Indexstrukturen notwendig, so dass die hier angebotenen Konzepte zu betrachten sind.

DB-Eigenschaften								
	Neo4J	**Sones GraphDB**	**Infogrid**	**HyperGraphDB**	**InfiniteGraph**	**DEX**	**Vertex DB**
Replikation	Master-Slave mit Master Failover	Nein	Peer-To-Peer	Peer-To-Peer	Objectivity/DB	Nein	Nein
Transaktionen							
ACID	Ja	Ja	Ja, Partiell	ACI	Ja	CI	Eingeschränkt
Weitere Konzepte	Nein	MVCC	Nein	MVCC SW-Transaktionaler Speicher	Nein	Nein	Nein
Persistenz (Storage Engine)	Proprietäres Format	Flexibel	Flexibel	Berkeley DB	Objectivity/DB	Proprietär	Tokyo Cabinet
In-Memory	Nein	Ja	Ja	Nein	k.A.	Ja	Nein
Cache	Ja	Ja	Ja	Ja	Nein	Ja	Nein
Embeddable	Ja	Nein	Ja	Ja	k.A.	Ja	Nein
Versionen	Nein	Nein	Erstellungs-/letztes Änderungsdatum	Nein	k.A.	k.A.	Nein
Verteilung	Domänenspezifisch	Nein	Partitioning-Algorithmen	Ja	Nein	Nein	Nein
Indizierung	Lucene	Ja	Abhängig von Storage Engine	Ja	Ja	Ja	Nein

Tabelle 3: Kategorie DB-Eigenschaften

Schnittstellen							
	Neo4J	**Sones GraphDB**	**Infogrid**	**HyperGraphDB**	**InfiniteGraph**	**DEX**	**Vertex DB**
APIs							
Java	Ja	Ja	Ja	Ja	Ja	Ja	Nein
C#	Ja	Ja	Nein	Nein	Nein	Ja	Nein
C++	Nein	Nein	Nein	Nein	Nein	Ja	Nein
Andere	Ruby, Python, Scala, Clojure, PHP, Javascript	Javascript, PHP, alle .NET-Sprachen	Nein	Nein	Nein	Alle .NET-Sprachen	Nein
Anfragesprache	Lucene, Cypher, Gremlin	GQL (Graph Query Language)	Nein	HGQuery API	Nein	Nein	Proprietär
Traversals	Ja	Ja	Ja	Ja	Ja	Ja	Nein
Protokolle							
HTTP	Ja	Ja	Ja	Nein	Nein	k.A.	Ja
REST	Ja	Ja	Ja	Nein	Nein	k.A.	Nein
WebDAV	Nein	Ja	Nein	Nein	Nein	k.A.	Nein

Tabelle 5: Kategorie Schnittstellen

Ebenfalls interessant ist die Möglichkeit der Einbettung der gesamten Datenbank in die Applikation und die Fragestellung, ob Versionen verwaltet werden können oder nicht.

Die beiden Konzepte der Replikation und Verteilung runden diese Kategorie ab, deren Übersichtsdarstellung in Tabelle 3 zu finden ist.

3.4 Utilities

Diese Kategorie fasst alle Werkzeuge zusammen, die rund um die Datenbank die Administration und den Betrieb erleichtern. Im Einzelnen sind dies die Existenz einer Shell bzw. eines Administrationswerkzeugs, die Integration in eine IDE (Integrierte Entwicklungsumgebung) sowie Werkzeuge zum Importieren, Exportieren und Laden von Daten. Die Resultate für die untersuchten Datenbanken befinden sich in Tabelle 4.

Utilities							
	Neo4J	**Sones GraphDB**	**Infogrid**	**HyperGraphDB**	**InfiniteGraph**	**DEX**	**Vertex DB**
Shell	Ja	Ja (Web-basiert)	Ja (HTTP)	Nein	k.A.	Ja	Nein
Administrationswerkzeug	Nein	Ja	Ja (Viewlets)	Nein	k.A.	Ja	Nein
IDE Integration	Ja (Eclipse)	Nein	Ja (Netbeans)	Nein	k.A.	Nein	Nein
Importwerkzeug	Ja	Nein	Ja	Nein	k.A.	Nein	Nein
Exportwerkzeug	Ja	Nein	Ja	Nein	k.A.	Ja	Nein
Ladewerkzeug	Ja	Nein	Ja	Nein	k.A.	Ja	Nein

Tabelle 4: Kategorie Utilities

3.5 Schnittstellen

Als letzte Kategorie sollen die von den Graphdatenbanken angebotenen Schnittstellen betrachtet werden. Dabei stehen an erster Stelle die programmiersprachlichen APIs, wobei den drei am weitesten verbreiteten Sprachen Java, C# und C++ im Schema jeweils eine eigene Zeile gewidmet wird, während andere Sprachen zusammengefasst werden. Neben dem API-Zugriff ist auch der deklarative Zugriff mittels einer deklarativen Anfragesprache wünschenswert. Weiterhin sind die graphspezifischen Traversierungen interessant, wozu sog. Traversals oder Traverser-APIs angeboten werden. Abschließend werden für den Einsatz in webbasierten Umgebungen die unterstützten Protokolle untersucht.

Tabelle 5 fasst die Resultate dieser Kategorie zusammen.

4 Zusammenfassung und Ausblick

In diesem Beitrag haben wir zunächst einen kurzen Überblick über die historische Entwicklung von Datenbanken gegeben, bis hin zu den neuesten Entwicklungen der als Big Data bezeichneten immer stärker anwachsenden Datenvolumen und der NoSQL-Ansätze. Auf Big Data und NoSQL wurde anschließend in Abschnitt 2 vertieft eingegangen, wobei als wesentliches Resultat festzuhalten ist, dass beide Bereiche zwar deutliche Schnittmengen aufweisen, aber keineswegs identisch sind, wie es sehr häufig dargestellt wird.

Eine besondere Ausprägung von NoSQL-Systemen sind Graphdatenbanken. Hierbei gibt es eine Vielzahl von Ideen, Konzepten, Prototypen und schon recht ausgereiften Systemen. Um hier eine gute und nachhaltige Orientierung erlangen zu können, hat sich der Artikel in Abschnitt 3 schwerpunktmäßig mit einem Klassifikationsschema für Graphdatenbanken beschäftigt. Dabei wurden fünf verschiedene Kategorien gebildet und dieses Klassifikationsschema auf aktuelle Graphdatenbanken angewendet.

In der Zukunft sollen weitere Untersuchungen von Graphdatenbanken durchgeführt werden, wobei vor allem ein Performancetest konzipiert werden soll, mit dessen Hilfe die einzelnen Graphdatenbanken untereinander, aber auch Graphdatenbanken mit relationalen Ansätzen verglichen werden können.

Literaturverzeichnis

[AG08] Renzo Angles, Claudio Gutiérrez: Survey of graph database models. ACM Computing Surveys. 40(1): (2008).

[ALS10] Anderson, J.C., Lehnardt, J. und Slater, N.: CouchDB: The Definitive Guide. O'Reilly Media, 2010.

[Am12] http://aws.amazon.com/de/simpledb/.

[Ba11] Banker, K.: MongoDB in Action. Manning-Verlag, 2011.

[BG81] Bernstein, P.A. und Goodman, N.: Concurrency Control in Distributed Database Systems. ACM Comput. Surv. 13(2): 185-221 (1981).

[Ca12] http://cassandra.apache.org/.

[CBS95] Connolly, T.M., Begg, C.E. und Strachan, A.: Database Systems: A Practical Approach to Design, Implementation and Management. Addison Wesley, 1995.

[CD+06] Chang F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra T., Fikes A., Gruber R.: Bigtable: A Distributed Storage System for Structured Data. OSDI 2006: 205-218.

[CD10] Chodorow, K. und Dirolf, M.: MongoDB: The Definitive Guide. O'Reilly Media, 2010.

[Ch12] http://sourceforge.net/projects/chordless/.

[Co12] http://couchdb.apache.org/.

[Co70] Codd, E.F.: A Relational Model of Data for Large Shared Data Banks. Commun. ACM 13(6): 377-387 (1970).

[EF+10] Edlich, S., Friedland, A., Hampe, J., Brauer, B. und Brückner, M.: NoSQL: Einstieg in die Welt nichtrelationaler Web 2.0 Datenbanken. Erste Auflage, Hanser-Verlag, München, 2010.

[Ge11] George, L.: HBase: The Definitive Guide. O'Reilly, 2011.

[Hb12] http://hbase.apache.org/.

[He10] Hewitt, E.: Cassandra: The Definitive Guide. O'Reilly, 2010.

[HM93] Herlihy, M. und Moss, E.: Transactional Memory: Architectural Support for Lock-Free Data Structures. ISCA 1993: 289-300.
[HLR10] Harris, T., Larus, J. und Rajwar, R.: Transactional Memory. Morgan and Claypool Publishers, Zweite Auflage, 2010.
[Hy12] http://hypertable.org/.
[Kö11] Körner, A.: Geschäftsinnovationen mit Zeitreihendaten: SmartMeter-, Sensor- und Finanz-Daten erschließen. Vortragsfolien, IBM Information Management Forum, Darmstadt, 2011. Online verfügbar unter ftp://ftp.software.ibm.com/software/emea/de/telefonkonferenz/2011-07-15_SWG_PartnerUniversity_Informix_Zeitreihen_v3.pdf
[KV84] Kuper, G.M. und Vardi,M. Y. A new approach to database logic. In Proceedings of the 3th Symposium on Principles of Database Systems (PODS). ACM Press, S. 86-96. Waterloo (Ontario, Kanada), 1984.
[Mo12] http://www.mongodb.org/.
[Re12] http://redis.io/.
[Re78] Reed, D. P.: Naming and Synchronization in a Decentralized Computer System. MIT Dissertation, September 1978. Online verfügbar unter http://publications.csail.mit.edu/lcs/specpub.php?id=773.
[Ri12] http://wiki.basho.com/Riak.html.
[RM75] Roussopoulos, N. und Mylopoulos, J. Using semantic networks for database management. In Proceedings of the 1st International Conference on Very Large Data Bases (VLDB), S. 144-172. Framingham (Massachusetts, USA), 1975.
[RW12] Redmond, E. und Wilson, J.R.: Seven Databases in Seven Weeks: A Guide to Modern Databases and the NoSQL Movement. Pragmatic Programmers Publisher, 2012.
[Sc03] Schneider, M.: Implementierungskonzepte für Datenbanksysteme. Springer, Berlin Heidelberg, 2003.
[SHS05] Saake, G., Heuer, A. und Sattler, K.-U.: Datenbanken: Implementierungstechniken. Verlagsgruppe Hüthig-Jehle-Rehm, 2005.
[St10] Stonebraker, M.: SQL databases v. NoSQL databases. Communications of the ACM (CACM) 53(4):10-11 (2010).
[St11] Stonebraker, M.: Stonebraker on NoSQL and enterprises. Communications of the ACM (CACM) 54(8):10-11 (2011).

[Ti11a] Tiwari, S.: Professional NoSQL (Wrox Programmer to Programmer). John Wiley & Sons, 2011.
[Ti11b] Tittmann, P.: Graphentheorie – Eine anwendungsorientierte Einführung. Hanser-Verlag, 2011.
[WK11] Wenk, A. und Klampäckel, T.: CouchDB: Das Praxisbuch für Entwickler und Administratoren. Galileo Computing, Bonn, 2011.

Für alle genannten Online-Quellen gilt: Letzter Abruf am 7.6.2012.