Im Vergleich: NoSQL vs. relationale Datenbanken

Kevin Klöckner
Universität Siegen
kevin.kloeckner@Student.Uni-Siegen.de

ABSTRACT

Mit dem stetigen Anstieg von digital gespeicherten Informationen, ist eine relationale Datenbank für einige Bereiche nicht mehr die optimale Lösung. Für diese Bereiche wurde NoSQL entwickelt. NoSQL-Datenbanken benötigen kein festgelegtes Tabellenschema, vermeiden Joins und skalieren horizontal. Damit ist NoSQL für große Datenmengen bestens geeignet.

Dieser Artikel setzt sich mit den Vor- und Nachteilen von NoSQL und relationalen Datenbanken auseinander und vergleicht ihre Funktionalität in den jeweiligen Bereichen.

Des weiteren bringt dieser Artikel jene Vergleiche anhand von Beispielen näher und verdeutlicht damit die genauen Vor- und Nachteile.

1. EINLEITUNG

Eine relationale Datenbank ist nicht in jedem Fall die beste Lösung. In einigen Bereichen haben relationale Datenbanken ziemliche Probleme. Ein klassisches Beispiel an dieser Stelle sind graphenbasierte Probleme, wie Netzwerke und Topologien jeglicher Art. Die Menge an digital gespeicherten Informationen wird immer größer, schon jetzt kämpfen relationale Datenbanken mit erheblichen Leistungsproblemen. Da es mittlerweile darum geht, Hunderttausende oder Millionen von Nutzer zu bedienen, kann die gewünschte Performance nicht mehr über einzelne Server erreicht werden, sondern nur mit Rechnernetzen und genau damit haben relationale Datenbanken große Probleme.[17]

Aus diesem Grund wurden NoSQL Datenbanken entwickelt. Eine NoSQL Datenbank verfolgt einen nicht relationalen Ansatz und versucht die Probleme, die eine relationale Datenbank in manchen Bereichen hat, zu lösen. Um nun zu veranschaulichen, wie gut NoSQL Datenbanken das realisieren, werden viele Arten von NoSQL Datenbanken jeweils mit einer relationalen Datenbank verglichen.

2. RELATIONALE DATENBANKEN

Eine relationale Datenbank ist eine auf Tabellen basierende Datenbank, die auf dem relationalen Datenbankmodell beruht. Das bedeutet dass die Daten mit einem eindeutigem Schlüssel ausgestattet sind und in einer Beziehung miteinander stehen. Die Abfragesprache der Datenbank ist SQL, eine weit verbreitete und größtenteils standardisierte Abfragesprache, so dass Anwendungsprogramme beinahe vom verwendeten Datenbanksystem unabhängig sind. Eine relationale Datenbank ist die am weitesten verbreitete Datenbank und bis heute ein etablierter Standard.[18]

Jedoch ist sie nicht für jede Art von Problem die Optimale Lösung. Aufgrund der Beschränkungen der relationalen Algebra bietet das relationale Datenbankmodell keine Unterstützung zur Berechnung von rekursiven Anfragen. Es ist beispielsweise nicht möglich, in einer Abfrage alle Vorfahren einer Person zu berechnen, wenn diese über eine Relation „VorfahreVon“ mit dem jeweiligen Vorfahren gespeichert sind.[18]

Außerdem sind Replikation bei verteilten Systemen kann nur sehr umständlich umgesetzt werden, relationale Datenbanksysteme sind für Ein-Server-Hardware konzipiert. Die bisherigen Versuche, relationale DB-Systeme und ihre Daten auf Rechner-Cluster zu verteilen, funktionieren zwar, aber die Performanz ist noch sehr gering. Ein weiterer Kritikpunkt sind die künstlichen Schlüssel die manchmal zur eindeutigen Identifizierung von Tupeln eingesetzt werden müssen, da diese mit der abstrakten Beschreibung eines Anwendungsobjektes nichts zu tun haben, sondern „nur“ Verwaltungsinformationen sind. [18]

3. DOKUMENTENORIENTIERTE DATENBANKEN

Eine dokumentenorientierte Datenbank ist eine Datenbank, bei der die Daten in Form von Dokumenten abgespeichert werden. Dabei hat jeder Datensatz ein eigenes Dokument, was mit einem eindeutigen Identifikator angesprochen werden kann.[2, 12]

Das die Datensätze in einzelne Dokumente gespeichert werden hat den Vorteil, dass jedes Dokument , wie in Abbildung 1 zu sehen ist, gänzlich andere Variablen enthalten kann.[12]

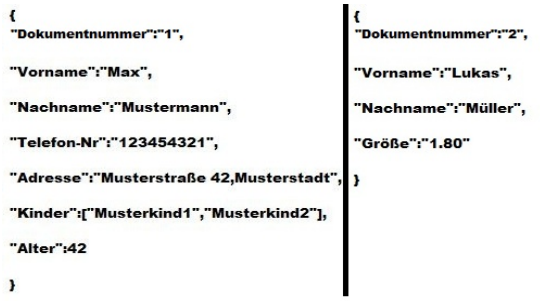


Figure 1: Zwei Beispieldokumente im JSON Format

4. DOKUMENTENORIENTIERTE DB VS. RELATIONALE DATENBANKEN

Um einen Vergleich zur relationalen Datenbank zu ziehen, schauen wir uns zuerst einmal die Vorteile des Dokumentenformats an. Wie wir bereits wissen, ist es in einer dokumentenorientierten Datenbank möglich, dass jedes Dokument sich von den anderen gänzlich unterscheidet.[12]

Möchte man zum Beispiel einem Dokument eine zusätzliche Information hinzufügen, so kann man dies einfach tun, weil die Dokumente keinem vorgefertigtem Dateitypen folgen. In einer relationalen Datenbank wäre dies undenkbar. Jeder Datensatz folgt in einer Tabelle einem festen Muster. So müsste für eine weitere Information jeder Datensatz der ganzen Tabelle ebenfalls diese Information hinzugefügt bekommen oder Null-Werte erlaubt werden.[2]

Eine dokumentenorientierte Datenbank wäre also dann vorzuziehen, wenn für die zu speichernden Daten kein gemeinsamer Typ festgelegt werden kann und sich die gespeicherten Informationen in vielen Datensätzen unterscheiden. Die Individualität der Dokumente bringt allerdings auch eine Schwäche mit sich: Referenzen. Um eine Referenz in dokumentenorientierten Datenbanken darzustellen, gibt es mehrere Möglichkeiten. Eine dieser Möglichkeiten wäre ein, wie in Abbildung 2 dargestelltes, eingebettetes Dokument. In dieser Methode wird ein Dokument einfach an das Aktuelle als neuer Unterpunkt angehängt.[7]

Figure 2: Beispiel eingebettetes Dokument

Diese Methode bietet sich nur an, wenn die Daten beim Auslesen des Eintrags auch benötigt werden, da bei jedem Zugriff auf ein Dokument dieses vollständig ausgelesen wird und deshalb keine Informationen gespeichert werden sollten, die viel Platz benötigen.[7]

Eine weitere Möglichkeit wäre eine Referenz auf die Objekt ID(Abbildung 3). Hier wird anstatt des kompletten Eintrags einfach auf die jeweilige Dokument ID verwiesen. Diese Methode sollte verwendet werden, wenn die Daten beim Auslesen nicht immer benötigt werden. Der Nachteil hier ist, dass zusätzliche Abfragen nötig sind, um an die referenzierten Daten zu kommen.[7]

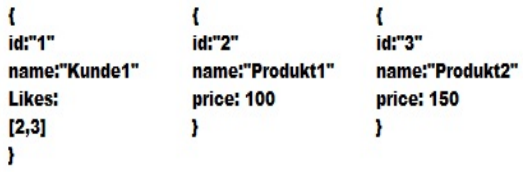


Figure 3: Beispiel Referenz Objekt ID

Dokumentenorientierte Datenbanken eignen sich besonders gut, um nicht stark vernetzte Daten mit unterschiedlichen Eigenschaften zu speichern. Bei stark vernetzten Daten sollte man lieber auf eine andere Datenbank zurückgreifen. Aus diesem Grund werden sie vor allem im Bereich der Web-Applikationen eingesetzt. Das Dokumentenorientierte Speicherformat eignet sich insbesondere für die Speicherung durch HTML-Formulare übertragener Daten.[7, 12]

5. GRAPHDATENBANKEN

Eine Graphdatenbank ist eine Datenbank, in der die Daten anhand eines Graphen dargestellt und abgespeichert werden. Ein Graph besteht jeweils aus Knoten und Kanten, welche die Beziehungen zwischen den Knoten darstellen. Die Knoten und Kanten können zusätzliche Attribute besitzen. Außerdem besitzt jede Kante eine Eigenschaft, welche die Beziehung definiert. Man könnte eine Graphdatenbank also eher als eine Beziehungsdatenbank verstehen. Der Graph kann sowohl gerichtet(Abbildung 4) als auch ungerichtet sein.[4, 15]

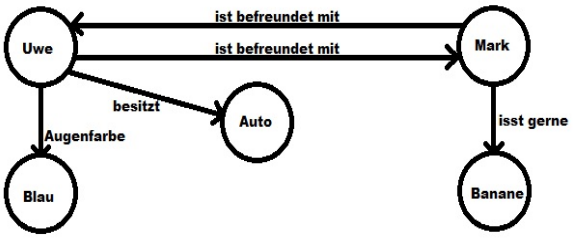


Figure 4: Beispiel eines gerichteten Graphes

Das die Beziehungen direkte Verbindungen sind, hat den Vorteil, dass Operationen zwischen den Einträgen viel effizienter ausgeführt werden können.[4]

6. GRAPHDATENBANKEN VS. RELATIONALE DATENBANKEN

Die Vorteile einer Graphdatenbank liegen in den meisten Fällen klar auf der Hand. Die Daten werden direkt vernetzt, gespeichert und verarbeitet. Durch diese Struktur kann man auf komplexe Anfragen wie rekursiv geschachtelte Joins verzichten und dies führt zu einer effizienten Traversierung[9, 10]

Traversierung eines Graphen bedeutet, dass der Graph beginnend von einem Startknoten durchlaufen wird. Meistens wird ein weiterer Knoten gesucht, wie z.B. der Weg von einer Stadt zur anderen. Durch die effiziente Traversierung ist die Performance deutlich höher als die einer relationalen Datenbank. Vor allem bei Systemen die auf das Suchen von Routen spezialisiert sind, da die Struktur einer Graphdatenbank genau für solche Fälle aufgebaut ist.[9, 10] Eine relationale Datenbank müsste hier auf komplexe Anfragen wie rekursiv geschachtelte Joins zurückgreifen, die sehr Performancelastig sind.[9, 10]

Die Skalierung einer Graphdatenbank hingegen ist nicht so einfach wie bei anderen NoSQL Datenbanken. Wird die Anzahl der Knoten und Kanten im Graphen für einen einzelnen Server zu groß, wird hier mit der Partitionierung versucht, den Graph aufzuteilen, so dass dieser auf mehrere Systeme verteilt liegt. Allerdings ist es nicht immer einfach eine entsprechende Stelle für eine Teilung im Graphen zu finden. Selbst bei einer gleich verteilten Relevanz aller Knoten im Graph, existiert keine mathematisch exakte Methode, um die Anzahl der durchschnittenen Kanten zu minimieren.[9, 10, 15]

Da eine relationale Datenbank allerdings ebenfalls Schwierigkeiten mit der Skalierung besitzt, ist dies kein direkter Nachteil gegenüber einer relationalen Datenbank, jedoch gegenüber anderen NoSQL Datenbanken. Ein sehr großer Nachteil einer Graph-Datenbank ist die Abfragesprache. Bisher existiert noch keine einheitliche Abfragesprache aufgrund der fehlenden Standardisierung zwischen den Graphdatenbanken. Es gibt viele Abfragesprachen sowie Graphmodelle. Eine Graphdatenbank wird beispielsweise für Systeme wie Routen-Planer oder connections in sozialen Netzwerken genutzt.[16]

7. KEY-VALUE-DATENBANKEN

Eine Key-Value-Datenbank ist eine Datenbank, in der jeder Datensatz einem einzigartigen Key zugeteilt wird(Abbildung 5). Der Key dient zum schnellen Zugriff auf den Datensatz und ignoriert den Inhalt des Datensatzes vollständig.[1, 11]

Das der Key den Inhalt des Datensatzes vollständig ignoriert hat den Vorteil, dass jede Operation nur einen einzigen Festplattenzugriff benötigt, um einen Datensatz zu finden.[1]

Die Key-Value-Datenbank weist eine gewisse Ähnlichkeit mit der dokumentenorientierten Datenbank auf, welche jedoch den Key mit in dem Dokument speichert, während in der Key-Value-Datenbank der Key nicht im Datensatz selber vorhanden ist, sondern lediglich zu diesem führt(Abbildung 6). Die Keys werden als assoziativer Array gespeichert in dem jeder Key zu genau einem Wert führt. Der Key zeigt also wie ein Pointer auf den Datensatz.

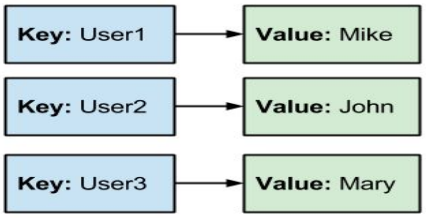


Figure 5: Beispiel des Key-Value Schemas

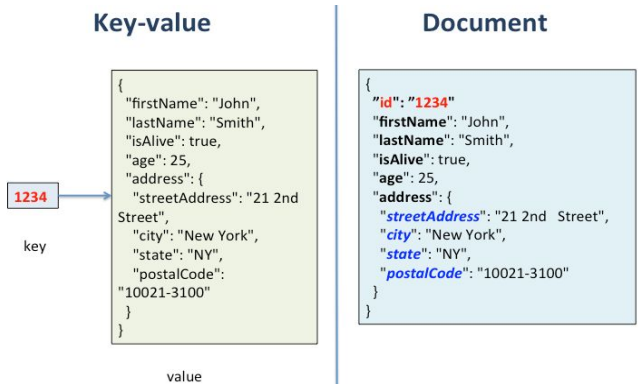


Figure 6: Beispiel Unterschied Key-Value und Dokumentenorientiert

8. KEY-VALUE-DATENBANKEN VS. RELATIONALE DATENBANKEN

Ein Vorteil des simplen ID -> Wert Formats, ist der schnelle und einfache Zugriff auf einen Datensatz. Ein weiterer Vorteil des Formats ist, die einfache Skalierbarkeit auf mehrere Server. Dadurch kann die Last auf mehrere Systeme verteilt werden, was in der heutigen Zeit bei den riesigen Datenmengen sehr nützlich ist. Dies nennt man horizontale Skalierung. Aus dem Grund, dass immer weitere Server hinzugefügt werden können, sind der horizontalen Skalierung keine hardware-technischen Grenzen gesetzt.[1, 11]

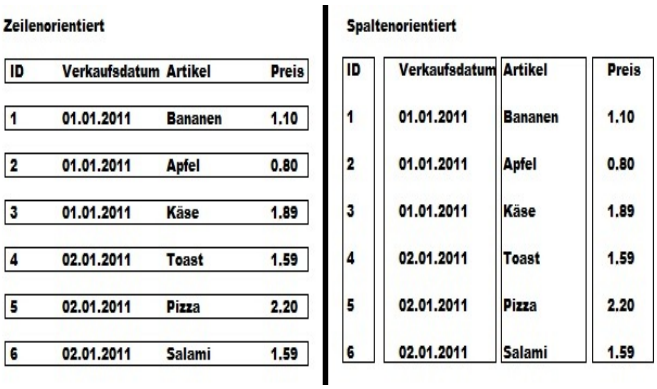
Eine relationale Datenbank skaliert vertikal. Das bedeutet, dass die Leistung nur durch Erweiterung und Verbesserung der Hardware erhöht werden kann. Vorteil dieser Skalierbarkeit ist, dass ein System, unabhängig der Software, mit Hilfe von besserer Hardware effizienter gemacht werden kann. Der Nachteil daran ist, dass das Limit durch die aktuell stärkste Hardware gesetzt ist. Weil der Zugriff einer Key-Value-Datenbank lediglich über den Schlüssel erfolgt, bieten solche Systeme oft nur eingeschränkte Abfragemöglichkeiten, um schnelle Lese- und Schreibeoperationen zu garantieren.[11]

Da es in Key-Value-Datenbanken außerdem keine Möglichkeit gibt, nach einem Wert zu suchen und die zugehörige ID auszulesen, muss für jede umgekehrte Beziehung ebenfalls ein Eintrag angelegt werden(z.B. ID => Username, Username => ID). Wenn ein System nun viele gegenseitige Beziehungen benötigt, wird es schnell unübersichtlich und man muss sich ständig um die Verwaltung kümmern.[6]

Wenn man also die Nachteile der Komplexität und der Abfragemöglichkeiten in Kauf nehmen kann, wird man mit einer enormen Geschwindigkeit und einfacher Skalierbarkeit belohnt. Sind die Daten jedoch zu stark vernetzt oder werden komplexe Abfragen benötigt, sollte man eine andere Datenbank benutzen.[6]

9. SPALTENORIENTIERTE DATENBANKEN

Eine spaltenorientierte Datenbank ist eine Datenbank, die ihre Informationen, im Gegensatz zu anderen Datenbanken, in einer Tabelle spaltenweise statt zeilenweise abspeichert und ausließt(Abbildung 7). Der Zusammenhang der Attribut-Werte ist durch deren Position in der jeweiligen Spalte gegeben. Dadurch wird das sequentielle Lesen kompletter Spalten begünstigt. [3, 13]



Zeilenorientiert

ID	Verkaufsdatum	Artikel	Preis
1	01.01.2011	Bananen	1.10
2	01.01.2011	Apfel	0.80
3	01.01.2011	Käse	1.89
4	02.01.2011	Toast	1.59
5	02.01.2011	Pizza	2.20
6	02.01.2011	Salami	1.59

Spaltenorientiert

ID	Verkaufsdatum	Artikel	Preis
1	01.01.2011	Bananen	1.10
2	01.01.2011	Apfel	0.80
3	01.01.2011	Käse	1.89
4	02.01.2011	Toast	1.59
5	02.01.2011	Pizza	2.20
6	02.01.2011	Salami	1.59

Figure 7: Beispieltabelle 1 Unterschied zeilenorientiert und spaltenorientiert

```
Speicher Reihenfolge Zeilenorientiert :
1, 01.01.2011, Bananen, 1.10,2, 01.01.2011
Apfel ....

Speicher Reihenfolge Spaltenorientiert:
1, 2, 3, 4, 5, 6, 01.01.2011, 01.01.2011
01.01.2011....
```

10. SPALTENORIENTIERTE DATENBANKEN VS. RELATIONALE DATENBANKEN

Der Vorteil einer spaltenorientierten Datenbank besteht also in dem Fall, in dem man für eine oder mehrere Spalten alle Zeilen auslesen will. Dabei muss man nicht in der kompletten Datei nach den richtigen Spalten suchen, sondern kann einfach den kompletten Block auslesen, wie beispielsweise alle Gehälter zum zusammenrechnen des Gesamtgehaltes.[3, 13, 8]

Vorname	Nachname	Gehalt	.....
Uwe	Bach	2212	
Udo	Nö	1845	
Hans	Beste	2487	
Mark	Neu	2245	

Figure 8: Beispieltabelle 2 Unterschied zeilenorientiert und spaltenorientiert

Im Abbildung 8 gezeigten Beispiel des Gesamtgehaltes würde eine zeilenorientierte Datenbank alle Zeilen komplett auslesen, während eine spaltenorientierte Datenbank nur eine einzige Spalte auslesen muss. Man sieht in diesem Beispiel gut, wie effizient eine spaltenorientierte Datenbank in den richtigen Einsatzgebieten sein kann. [8]

Eine zeilenorientierte Speicherung ist hingegen besser, wenn man die Informationen vieler Spalten benötigt. Ein Beispiel wäre das Suchen nach Einzelpersonen und deren Informationen. Ergebnis wäre nur eine Zeile und viele Spalten, eine zeilenorientierte Datenbank wäre hier also klar im Vorteil.[3, 8]

Das die Daten in Spalten gespeichert werden, hat außerdem noch den Vorteil einer einfachen Kompression, da in einer Spalte oft gleichartige Daten vorkommen und diese physisch nah beieinander liegen. Das macht eine Dekompression sehr einfach und im besten Fall können Operationen direkt auf die komprimierten Daten ausgeführt werden. [3] Im Endeffekt kommt es darauf an, für welche Abfragen die Datenbank genutzt werden soll. Möchte man seine Datenbank beispielsweise für Buchhaltungssysteme nutzen, die über wenig Zeilen aber viele Spalten gehen, ist eine zeilenorientierte Datenbank die bessere Wahl. Nutzt man die Datenbank allerdings oft für analytische Zwecke, die wenige Spalten aber viele Zeilen umfassen, dann ist hier die spaltenorientierte Datenbank ganz klar überlegen. Aus diesem Grund werden sie für analytische Informationssysteme, bei denen die direkte Verarbeitung von Informationen benötigt ist genutzt wie beispielsweise Google. [3, 8]

11. OBJEKTDATENBANKEN

Eine Objektdatenbank ist eine Datenbank, die ihre Informationen objektorientiert abspeichert. Die Daten werden also als komplexe Objekte mit Attributen gespeichert, daher kann jedes Attribut selbst wieder ein Objekt sein. Jedes Objekt hat eine eindeutige Identität und ein Zugriff auf dessen Attribute ist nur über Methoden möglich. Die Objekttypen/Klassen besitzen hier die gleiche Rolle, wie die Klassen in der objektorientierten Programmierung. Sie geben an, welche Art von Objekte erstellt werden können. Außerdem gibt es, genau wie in der objektorientierten Programmierung, Vererbungen, welche auch hier Attribute und Methoden vererben(Abbildung 9).[5, 14]

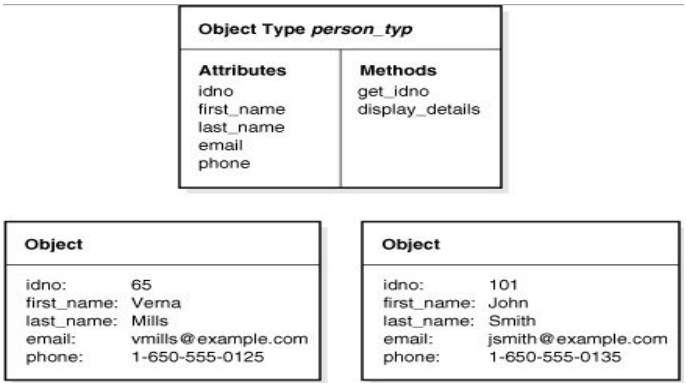


Figure 9: Beispiel Objektdatenbank

Objektdatenbanken nutzen die standardisierte Abfragesprache OQL, die sehr stark an SQL angelehnt ist. OQL wurde extra für die Interaktionen zwischen der objektorientierten Programmierung und einer Objektdatenbank entwickelt.[14]

12. OBJEKTDATENBANKEN VS. RELATIONALE DATENBANKEN

Damit wir eine Objektdatenbank mit einer relationalen Datenbank vergleichen können, schauen wir uns zunächst an, warum objektorientierte Datenbanken entwickelt wurden. Relationale Datenbanken haben eine sogenannte objektrelationale Unverträglichkeit. Das bedeutet, dass Daten, um gespeichert zu werden, in eine andere Form umgewandelt werden müssen.[14]

In einer relationalen Datenbank werden die Objekte als Relationen abgespeichert. Dies kann jedoch viele Probleme mit sich bringen, beispielsweise gibt es für Vererbungen im Relationalen keinen vergleichbaren Ansatz. Außerdem haben Objekte eine Objektidentität unabhängig von ihren Attributen. Ein Primärschlüssel im relationalen Bereich hingegen besteht aus Attributen eines Datensatzes. Deshalb muss hier ein künstlicher Primärschlüssel(Surrogatschlüssel) angelegt werden, der die Objektidentität widerspiegelt. Ein weiteres Beispiel wären die Methoden, für die es ebenfalls keinen Vergleich im relationalen System gibt, da man die Attribute in einer relationalen Datenbank direkt ändern kann. Aus diesem Grund kann eine Datenkapselung nicht umgesetzt werden.[5, 14]

Die objektorientierte Datenbank wurde entwickelt, um genau diese Probleme zu lösen. Eine objektorientierte Datenbank kommuniziert direkt mit der Anwendung. Das heißt, die Objekte werden direkt in der Datenbank angesprochen, da eine Umwandlung nicht mehr notwendig ist. Da Objekte eine komplexe Struktur haben können, sind Zusammenhänge zwischen Objekten dem Datenbanksystem bekannt.

Die Daten benötigen außerdem keine Joins für die Objektidentität. Da das Format genau dem der objektorientierten Programmierung entspricht und die Objekt IDs vom System festgelegt werden, können Objekte einfach über die in der Datenbank gespeicherten Beziehungen abgefragt werden.[5, 14]

Objektorientierte Datenbanken können ihre Stärken nur bei sehr komplexen Problemen nutzen. Benötigt man bloß einfache Strukturen und Abfragen, ist sie im Vergleich zu einer relationalen Datenbank langsamer. Ein weiterer Nachteil ist, dass Objektdatenbanken nur wenig verbreitet sind. Dies hat zur Folge, dass die Systeme in der Regel sprachspezifisch sind und der Zugriff nur von wenigen Sprachen aus möglich ist. Außerdem gibt es zahlreiche Tools und Schnittstellen, die in Objektdatenbanken nicht eingesetzt werden können.[5, 14]

13. FAZIT

Ziel dieser Arbeit war es, NoSQL und relationale Datenbanken in vielen verschiedenen Bereichen zu vergleichen. Zu diesem Zweck wurden viele Arten von NoSQL Datenbanken jeweils mit einer relationalen Datenbank verglichen, um so die jeweiligen Vor- und Nachteile dieser Art von Datenbank gegenüber einer relationalen Datenbank zu verdeutlichen.

Dabei hat sich ergeben, dass NoSQL Datenbanken eher auf bestimmte Fälle spezialisiert sind. Die Struktur der Daten spielt hier eine große Rolle, jede Art von NoSQL Datenbank arbeitet in ihrem speziellen Fall hervorragend. Während eine NoSQL Datenbank in ihrem Bereich der relationalen Datenbank meistens weit überlegen ist und dort viele Vorteile bietet, ist sie in den jeweiligen anderen Bereichen der relationalen Datenbank weit unterlegen und für diese Fälle nicht zu empfehlen.

Sollte man also eine Datenbank nur für ganz spezielle Zwecke nutzen, wäre hier eine NoSQL Datenbank genau das Richtige. Wird eine Datenbank jedoch für viele verschiedene Bereiche, Funktionen und Referenzen genutzt, ist eine relationale Datenbank klar im Vorteil.

14. REFERENCES

[1] Stand 10.12.2015:http://nosqlguide.com/key-value-store/nosql-databases-explained-key-value-stores/

[2] Stand 10.12.2015:http://nosqlguide.com/document-store/nosql-databases-explained-document-databases/

[3] Stand 10.12.2015:http://nosqlguide.com/column-store/nosql-databases-explained-wide-column-stores/

[4] Stand 10.12.2015:http://nosqlguide.com/graph-database/nosql-databases-explained-graph-databases/

[5] Stand 10.12.2015:https://docs.oracle.com/cd/B28359_01/appdev.111/b28371/adobjint.htm

[6] Stand 10.12.2015:http://eliteinformatiker.de/2011/06/01/nosql-key-value-datenbanken-memcachedb-project-voldemort-redis/

[7] Stand 10.12.2015:http://eliteinformatiker.de/2011/05/18/nosql-document-store-couchdb-mongodb/

[8] Stand 10.12.2015:http://eliteinformatiker.de/2011/08/07/zeilenorientierte-und-spaltenorientierte-datenbanken/

[9] Stand 10.12.2015:http://eliteinformatiker.de/2011/07/02/nosql-graph-datenbanken-neo4j/

[10] Stand 10.12.2015:http://eliteinformatiker.de/2011/11/29/nosql-graphendatenbanken-theorie/

[11] Stand 10.12.2015:http://lwibs01.gm.fh-koeln.de/wikis/wiki_db/index.php?n=Datenbanken.KeyValueSysteme

[12] Stand 10.12.2015:http://lwibs01.gm.fh-koeln.de/wikis/wiki_db/index.php?n=Datenbanken.Dokumentenorientierte Datenbank

[13] Stand 10.12.2015:http://lwibs01.gm.fh-koeln.de/wikis/wiki_db/index.php?n=Datenbanken.SpaltenorientierteDatenbank

[14] Stand 10.12.2015:http://lwibs01.gm.fh-koeln.de/wikis/wiki_db/index.php?n=Datenbanken.Objektdatenbank

[15] Stand 10.12.2015:http://neo4j.com/developer/graph-database/

[16] Stand 10.12.2015:http://neo4j.com/blog/why-database-query-language-matters/

[17] Stand 10.12.2015:http://www.emc.com/leadership/digital-universe/2014iview/executive-summary.htm

[18] Stand 10.12.2015:http://wikis.gm.fh-koeln.de/wiki_db/Datenbanken/RDBMS