Anfragesprachen für Big Data

Zusmmenfassung zum Vortrag im Oberseminar: Datenbanksysteme - Aktuelle Trends

Felix Haller (Mat.-Nr.: 68426)

03. Mai 2017

Inhaltsverzeichnis

1 Geschichte und Einordnung SQL/NoSQL (Motivation/Demo)

Die Geschichte von *NoSQL*[1] ist vergleichbar mit der von *SQL* in den 70er Jahren des vergangenen Jahrhunderts. Damals gab es sehr viele Anbieter, die sehr viele unterschiedliche Datenbanksysteme anboten. Diese waren in ihrem Aufbau meist grundverschieden und untereinander nicht kompatibel. Es bildeten sich heterogene Strukturen, die schwer oder gar nicht kombinierbar waren. Zudem war der Umgang mit einer Datenbank bis dahin meist geprägt durch Arbeiten, die nah an der Hardware und somit eher auf physischer Ebene stattfanden.
Das sollte sich um 1975 ändern. Man wollte dem Entwickler ermöglichen mit einer *vereinheitlichten* und strukturierten Abfragesprache auf logischer Ebene mit der Datenbank zu kommunizieren. Die Entwicklung von *SEQUEL* (später *SQL*) begann. Die Entwicklung und Etablierung dieser Sprache als Standard hat das Arbeiten mit Datenbanken seitdem stark vereinfacht und neue Anwendungsgebiete ermöglicht.
Bei NoSQL Datenbanken sieht es heute ähnlich aus. In den letzten 10 Jahren haben sich mehr als 225 NoSQL Datenbanken in mehr als 10 verschiedenen Kategorien entwickelt[2] und jede verfolgt ihre eigenen Ziele auf ganz unterschiedliche Art und Weise. Auch die

[1] Not Only SQL
[2] Quelle: nosql-database.org

Abfragesprachen unterscheiden sich sehr stark voneinander und erschweren somit die Kombination verschiedener Systeme.
Im Vortrag wurden zu Beginn zwei weit verbreitete NoSQL-Systeme vorgestellt: *CouchDB* und *MongoDB*, die beide einen unterschiedlichen Ansatz verfolgen. CouchDB [COUCH] setzt auf die Ansteuerung und Bedienung über eine HTTP-API mittels der HTTP Methoden *GET*, *PUT*, *POST* und *DELETE*, welche hier die CRUD[3]-Funktionen repräsentieren. Es existieren mit *Fauxton* und *Futon* auch zwei graphische Benutzerschnittstellen, die als Webanwendung implementiert sind und die ebenfalls ausschließlich die o.g. HTTP-API verwenden. Das zweite demonstrierte NoSQL-Datenbanksystem *MongoDB* verfolgt einen eher lokalen Ansatz und ist über ein eigenes Kommandozeilenprogramm bedienbar oder mittels einer Programmbibliothek aus vielen Programmiersprachen heraus ansprechbar. Die Datenrepräsentation erfolgt hier wie auch bei CouchDB standardmäßig im JSON[4]-Format. Zur Interaktion mit der Datenbank stellt MongoDB eigene Methoden bereit, die auf den jeweiligen Datenbeständen arbeiten und ebenfalls JSON als Datenformat für die Eingabedaten akzeptieren. Abbildung 1 zeigt das beispielhafte Einfügen von Daten mittels der *insert* Methode in einen Datenbestand namens *restaurants*. Ziel der Demonstration war es zu zeigen, wie unterschiedlich die bestehenden Abfragesprachen der Systeme sind, obwohl es sich im Grunde um Technologien für einen ähnlichen Anwendungsbereich handelt.
NoSQL Datenbanken im Allgemeinen arbeiten meist auf einem Datenmodell, welches mit Schlüssel-Wert Paaren arbeitet und für diese Art der Datenhaltung optimiert ist. Es gibt verschiedene Arten von NoSQL-Datenbanken z.B. reine Key-Value Stores, Dokumentendatenbanken, Graphendatenbanken, spaltenorientierte (s.g. Column Stores) und viele andere, die aber eher eine untergeordnete Rolle spielen.
Die Anwendungsgebiete und die Ziele sind aber bei den meisten Arten ähnlich. Es geht meist um große Datenmengen, die möglichst verteilt und über Webschnittstellen erreichbar abgelegt werden soll. Die meisten NoSQL Datenbanken sind für viele gleichzeitige Lesezugriffe optimiert und folgen dem *BASE*[5]-Prinzip, welches sich gewissermaßen im Gegensatz zu dem in der Welt der relationalen Datenbanken anerkannten ACID befindet. BASE besagt, dass es keine traditionellen Transaktionen wie bei SQL gibt und die Konsistenz zu einem bestimmten Zeitpunkt nicht garantiert werden kann. Das System wird zwar in Zukunft konsistent sein, doch nicht unbedingt zum Zeitpunkt der Abfrage. Das ermöglicht ein flexibleres Arbeiten mit den Datenbeständen, muss aber durch verschiedene Synchronisationsmechanismen gesteuert und überwacht werden [PRITCH].
Aus o.g. Gründen der Heterogenität hat sich nun auch in der Welt der NoSQL Datenbanken eine Bewegung gebildet, die versucht eine einheitliche Abfragesprache für die verschiedenen Systeme zu entwickeln, die im Idealfall zusätzlich das Abfragen von herkömmlichen relationalen Datenbanken ermöglicht. Das Ziel ist die Verbesserung der Interoperabilität zwischen den Produkten um einen gesunden Wettbewerb und damit

[3] Create, Remove, Update und Delete
[4] JavaScript Object Notation
[5] basically available, soft state, eventually consistent

```
db.restaurants.insert(
 {
    "address" : {
       "street" : "2 Avenue",
       "zipcode" : "10075",
       "building" : "1480",
       "coord" : [ -73.9557413, 40.7720266 ]
    },
    "borough" : "Manhattan",
    "cuisine" : "Italian",
    "grades" : [
       {
          "date" : ISODate("2014-10-01T00:00:00Z"),
          "grade" : "A",
          "score" : 11
       },
       {
          "date" : ISODate("2014-01-16T00:00:00Z"),
          "grade" : "B",
          "score" : 17
       }
    ],
    "name" : "Vella",
    "restaurant_id" : "41704620"
 }
)
```

Abbildung 1: Eine Einfügeoperation in MongoDB (Quelle: [MONGO])

hoffentlich Wachstum in der Branche zu generieren. Im Folgenden werden einige Ansätze vorgestellt.

2 coSQL

Ein interessanter Ansatz ist der von Erik Meijer und Gavin Bierman, welche beide bei Microsoft tätig sind. Sie haben es sich zur Aufgabe gemacht ein mathematisches Datenmodell für Key-/Value-Stores zu entwickeln, sozusagen ergänzend zum relationalem Modell für SQL-Datenbanken. Das Ziel ist die Erschaffung einer einheitlichen Abfragesprache für NoSQL.

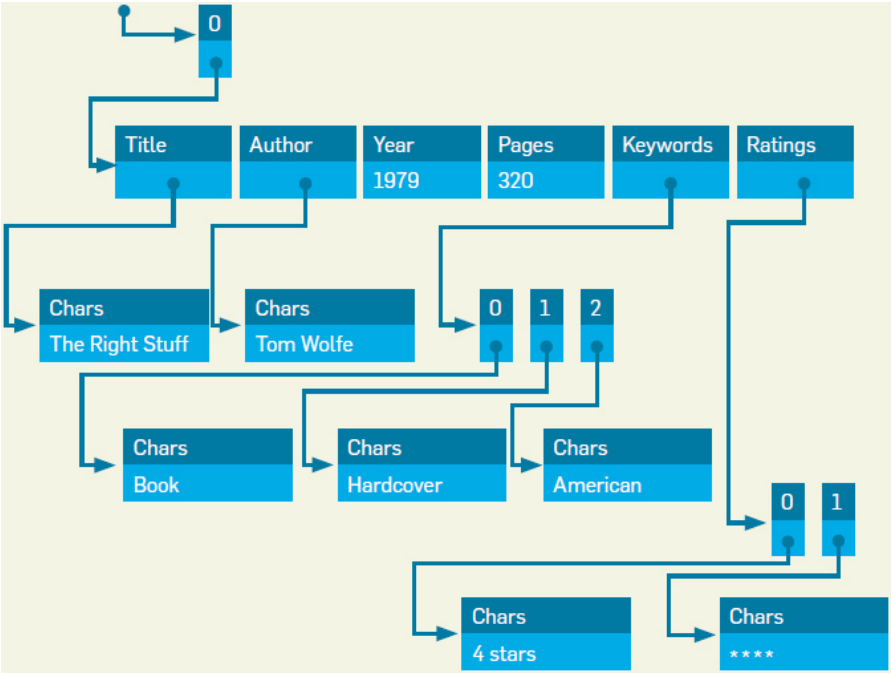

Abbildung 2: Der Beispiel-Objektgraph.

Betrachtet wurde ein Objekt, wie man es aus der objektorientierten Programmierung kennt (Abbildung 2). Die Identitäten der Objekte werden hier als *intensional* beschrieben, was bedeutet, dass Informationen über die Zugehörigkeit der Objekte zueinander nicht Bestandteil des Datensatzes sind. Es handelt sich lediglich um verschachtelte Sammlungen. Modelliert man diesen Objektgraphen nun in einem relationalen Datenbanksystem, entstehen nach der Normalisierung drei Tabellen für Produkte, Schlüsselwörter und Bewertungen (Abbildung 3). Um nun die Zusammenhänge speichern zu können und die referentielle Integrität zu wahren müssen Primär- und Fremdschlüssel eingeführt werden. Diese Darstellung der Identitäten wird als *extensional* bezeichnet.
Durch verschiedene Umformungen und Inlining der Objekte lassen sich beide Speicherformen nun gegenüberstellen (Abbildung 4). Zu erkennen ist, dass sich im Prinzip nur

Abbildung 3: Darstellung in einem RDBMS

die Richtung der Pfeile unterscheidet. Im Objektgraphen zeigen diese zum Unterobjekt und im relationalen Graphen zum Oberobjekt (über Primärschlüssel). In der Mathematik gibt es für diesen Zusammenhang ein eigenes Fachgebiet: die *Kategorientheorie*. Das Kern-Konzept besagt dort:

> Sei C eine Kategorie von Objekten und Pfeilen, dann erhält man die duale Kategorie $co(C)$, indem man alle Pfeile in C umdreht.
>
> Wenn eine Aussage T wahr ist in C, so ist die duale Aussage $co(T)$ wahr in $co(C)$.

Ein Beispiel für diesen Sachverhalt sind die De Morganschen Gesetze:

$$\overline{(a \wedge b)} = \overline{a} \vee \overline{b}$$

$$\overline{(a \vee b)} = \overline{a} \wedge \overline{b}$$

Man bezeichnet diesen Zusammenhang auch als *Dualität*. Diese tritt offensichtlich auch bei unserem Objektgraphen und der Darstellung im relationalen Modell auf. Ein Pfeil zeigt bei SQL vom Kindknoten auf den Vaterknoten, wenn der Fremdschlüssel des Kindes gleich dem Primärschlüssel des Vaters ist. Gleichzeitig zeigt im Objektgraphen-Modell von NoSQL ein Pfeil vom Vaterknoten auf den Kindknoten, wenn ein Zeiger vom Vaterobjekt auf das Kindobjekt existiert. Das bedeutet die Kategorie NoSQL ist dual zur Kategorie NoSQL und da man in der Kategorientheorie dualen Kategorien ein *co* vorne anstellt, handelt es sich bei NoSQL in diesem Sinne eigentlich um *coSQL*.
Meijer und Bierman sehen coSQL aufgrund dieser Dualität nicht als Konkurrenz zu SQL, sondern eher als die zweite Seite der gleichen Medaille an. Jede der Technologien hat aufgrund ihres Aufbaus Stärken und Schwächen und muss für jeden Anwendungszweck gesondert betrachtet werden. Durch das gefundene mathematische Modell gibt es

nun aber eine Grundlage für die Erstellung einer geeigneten Abfragesprache für coSQL Datenbanken. Bei Microsoft hat man diese Sprache in LINQ[6] implementiert.

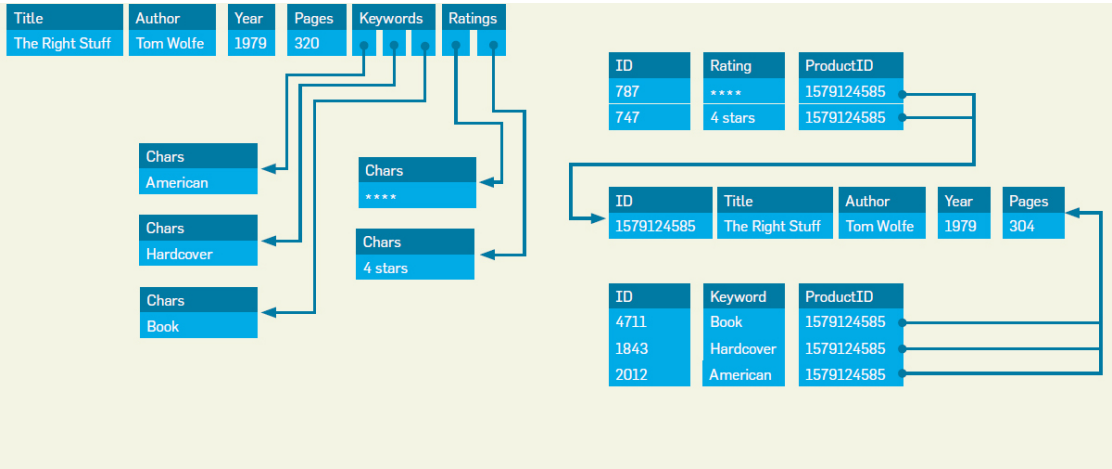

Abbildung 4: Gegenüberstellung von intensionaler (links) und extensionaler (rechts) Speicherung

3 FLWOR-Ausdrücke

Im Vortrag wurden zudem die *FLWOR*-Ausdrücke behandelt. Dieses Akronym steht für die Schlüsselwörter *for*, *let*, *where*, *order by* und *return*. Diese bilden sozusagen das Äquivalent zu den SELECT-FROM-WHERE Ausdrücken in der SQL-Welt und somit die Grundlage für bestimmte Abfragesprachen, wie z.B. XQuery [XQUERY]. FLWOR-Ausdrücke bieten Join-ähnliche Funktionalitäten für strukturierte Daten wie z.B. XML-Dateien. Außerdem erlauben sie Iteration und das Speichern von Werten in Variablen innerhalb einer Abfrage.
Aufbauend auf XQuery existiert noch JSONiq, welches ähnliche Konzepte umsetzt, dabei aber auf das Datenformat JSON setzt. auch hier kommen die FLWOR-Ausdrücke zur Anwendung. Da JSONiq eine Obermenge von JSON selbst darstellt, ist jedes valide JSON Dokument auch eine valide JSONiq Abfrage.
Das dritte vorgestellt Projekt trägt den Namen *SQL++* [SQLPP] und wird entwickelt unter der Aufsicht der Apache Foundation. Es stellt eine zu SQL abwärtskompatible Syntax bereit und bietet somit zumindest in der Theorie eine Sprache für beide Welten. SQL++ arbeitet auf dem Asterix Datenmodell, welches ebenfalls eine Obermenge von JSON ist. Die Abfragen bestehen aus den SELECT-FROM-WHERE Schlüsselworten, wie sie aus SQL bekannt sind, zuzüglich einiger Ergänzungen um komplexe Anfragen in strukturierten hierarchischen Daten (z.B. XML oder JSON) zu ermöglichen.

[6] https://msdn.microsoft.com/de-de/library/aa479865.aspx

4 Fazit

Es hat sich gezeigt, dass es sinnvoll ist, auf der Suche nach einer einheitlichen Sprache, die verschiedenen NoSQL Datenbanksysteme auf ihrer Gemeinsamkeiten hin zu untersuchen und zu versuchen ein mathematisches Modell aus diesen abzuleiten. Die daraus entstehende Sprache wird vermutlich eine höchstmögliche Kompatibilität zu den verschiedenen Systemen bieten können. Der Ansatz von Meijer und Bierman bestätigt das. Generell wäre es wünschenswert, wenn sich aus dem Dschungel von verschiedenen Systemen eine Sprache entwickeln würde, die auf der einen Seite so einfach ist wie SQL und auf der anderen Seite vielleicht sogar offen und frei im Sinne der freien Standards ist. Wenn diese Sprache dann auch noch Möglichkeiten zur gleichzeitigen transparenten Abfrage von NoSQL *und* SQL Datenbanken bietet, dann bleiben für den Datenbankentwickler der Zukunft kaum noch Wünsche offen.

Literatur

[HALLER] Felix Haller: *Anfragesprachen für Big Data*
Vortrag vom 03.05.2017, als Anlage

[MEIJER] Erik Meijer, Gavin Bierman:
A Co-Relational Model of Data for Large Shared Data Banks.
Communications of the ACM, Vol. 54 No. 4, Seiten 49-58

[PRITCH] Dan Pritchett:
Base: An Acid Alternative.
Magazine Queue - Object-Relational Mapping, Vol. 6 Issue 3, Seiten 48-55

[ORACLE] ORACLE WHITE PAPER: *Unified Query for Big Data* Management Systems , März 2016
http://www.oracle.com/technetwork/database/bigdata-appliance/learnmore/bigdatasqloverview21jan2015-2408000.pdf

[SQLPP] http://forward.ucsd.edu/sqlpp.html

[XQUERY] https://docs.microsoft.com/de-de/sql/xquery/xquery-basics

[MONGO] https://docs.mongodb.com/manual/

[COUCH] http://docs.couchdb.org/en/2.0.0/