

Teil 1: Dokumentenbasierte Datenbanken im Überblick: Konzepte und Funktionsweise

Unstrukturiert mit Struktur: NoSQL

Relationale Datenbanken haben lange Zeit den Bereich der Datenverarbeitung dominiert. Die immer weiter steigenden Mengen digital gespeicherter Informationen haben jedoch dazu geführt, dass eine relationale Datenbank für einige Bereiche nicht mehr die optimale Lösung ist. Insbesondere für unstrukturierte Daten sind NoSQL-Datenbanken besser geeignet. Wir geben einen Überblick über Datenmodelle jenseits der Tabellenstruktur.

von Elena Bochkor und Dr. Veikko Krypczyk

Denkt man an Daten, assoziiert man damit auch sehr oft eine Speicherung in Tabellenform. Daten gleichen Typs, lediglich mit unterschiedlichen Werten (Inhalten), werden als Datensätze gespeichert und in Tabellen abgelegt. Stehen die Tabellen untereinander über ein Schlüsselkonzept in Beziehung und sind die Tabellenstrukturen normalisiert, haben wir es mit einer relationalen Datenbank zu tun. Tabellen wiederum bestehen aus Spalten und Zeilen für die Datenspeicherung. Diese Form der Datenspeicherung ist ideal, wenn die zu speichernden Daten sich in eine genaue Struktur und Form bringen lassen. Typische Beispiele sind Kundendaten (Name, Vorname, Anschrift, Geburtsdatum, ...), Produktdaten (Artikelnummer, Bezeichnung, Preis, ...) oder Bestelldaten (Produktnummer, Kundennummer, Anzahl, ...). Ähnliche Beispiele lassen sich massenhaft finden. Relationale Datenbanken sind damit für die Speicherung strukturierter Daten ideal. Es

wird wenig Speicherplatz benötigt (Vermeidung von Redundanz), und der Zugriff auf einen einzelnen Datensatz ist über einen Primärschlüssel sehr schnell möglich (Kasten: „Relationale Datenbank“).

Die Anforderungen an die Datenhaltung unterliegen jedoch einem Wandel. Im Zeitalter von Mobile Computing, Big Data (Kasten: „Big Data“) und Cloud-Computing haben wir es mit einem immer größer werdenden Aufkommen an Daten zu tun (Abb. 1).

Eine solche Speicherung von Massendaten muss bewältigt werden. Hinzu kommt der Umstand, dass nur ein Bruchteil der vorliegenden Daten in strukturierter Form (Tabellen) vorliegt oder in eine solche Form gebracht werden kann. Die meisten Daten werden vielmehr in unstrukturierter oder semistrukturierter Form vorliegen (Abb. 2).

Auch für diese Datenformate muss es einen Weg geben, mit ihnen zu arbeiten, d. h. sie zu speichern, zu lesen, zu ändern und schnell auf sie zuzugreifen. Beispiele für unstrukturierte Daten sind zum Beispiel Texte.

Da die Datenbanksprache SQL (Structured Query Language) eng mit der Arbeitsweise von relationalen Datenbanken verbunden ist, werden andere Datenbankentypen zunächst einmal unter dem Begriff NoSQL-Datenbanken zusammengefasst. Letztere benötigen kein festgelegtes Tabellenschema, vermeiden Joins (Zusammenhänge zwischen Tabellen) und skalieren horizontal.

Artikelserie

Teil 1: Dokumentenbasierte Datenbanken im Überblick: Konzepte und Funktionsweise

Teil 2: Praxiseinsatz: eine NoSQL-Datenbank in der eigenen Applikation

Wann ist es besser eine NoSQL-Datenbank zu verwenden? Gibt es unterschiedliche Ansätze für diese Form der Datenbank? In unserer zweiteiligen Artikelserie gehen wir diesem Thema in Theorie und Praxis auf den Grund. Dieser erste Teil widmet sich der notwendigen Theorie von NoSQL-Datenbanken. In Teil 2 blicken wir dann in die Praxis. Wie kann man mit einer solchen Datenbank arbeiten? Wie funktioniert der Datenzugriff aus PHP?

NoSQL und Datenmodell im Überblick

Der Begriff NoSQL ist ein wenig irreführend, denn das „No“ in NoSQL steht für „Not only SQL“ und nicht für „Not SQL“. Das bedeutet also nicht das Ende der Abfragesprache SQL. Gemeint sind dabei alle Arten von Datenspeicherung, die nicht den relationalen Datenbanken zuzuordnen sind. Durchaus kann im Einzelfall mit der Abfragesprache SQL oder

einem ähnlichen Dialekt gearbeitet werden. Ursprünglich stammt der Begriff aus dem Jahr 1998 und wurde durch Carlo Strozzi (IBM) geprägt. Damals bezeichnete der Begriff der NoSQL-Datenbank zwar eine relationale Datenbank, aber eine ohne SQL API. Erst seit dem Jahr 2009 steht NoSQL für „Not only SQL“ und wird

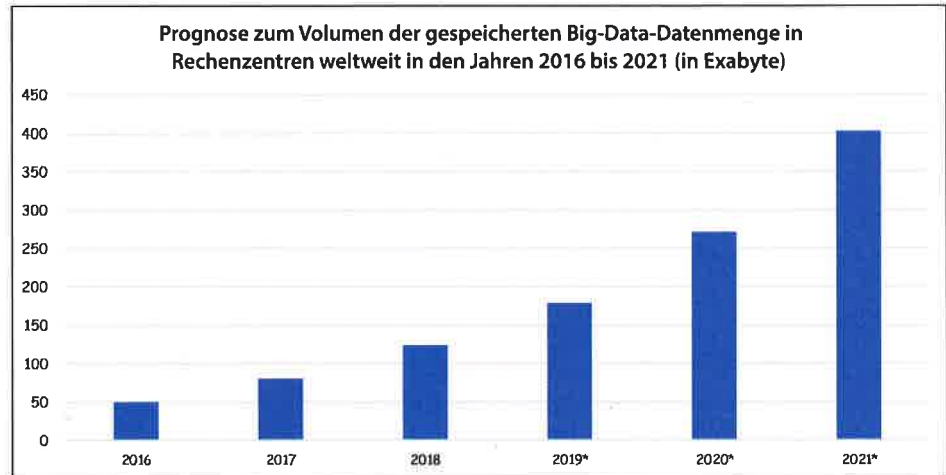


Abb. 1: Explosion der Datenmengen als Folge der Digitalisierung [1]

Relationale Datenbank

Relationale Datenbanken basieren auf Tabellen und sind dem relationalen Datenmodell zuzuordnen. Das relationale Datenmodell ist das am weitesten verbreitete und im praktischen Einsatz bewährteste Modell. Es stammt aus dem Jahr 1970 und wurde zum ersten Mal von Edgar F. Codd bei IBM vorgestellt. Bis heute bleibt es ein etablierter Standard.

Das Grundprinzip relationaler Datenbanken ist, dass sie konsistent und redundanzfrei sein müssen. Die grundlegende Datenstruktur ist die Relation, d. h. die Tabelle mit Spalten und Zeilen. Dabei werden die Zeilen als Tupel und die Spalten als Attribute der Relation bezeichnet. Die Daten in einer relationalen Datenbank sind mit einem eindeutigen Schlüssel ausgestattet und stehen stets in einer Beziehung zueinander. Ein sogenanntes Relationenschema legt die Anzahl und die Typen der Attribute einer Tabelle fest. Relationale Datenbanken

setzen SQL als Abfragesprache ein. Diese Sprache basiert auf der relationalen Algebra und besitzt eine relativ einfache Syntax mit wenigen Grundbefehlen. Das zugehörige Datenmanagementsystem nennt sich RDBMS (Relational Database Management System). Probleme zeigen relationale Datenbanken im Big-Data-Umfeld. Bei sehr großen Datenmengen, jenseits des Terabytebereichs, stoßen sie heute an ihre Grenzen oder verursachen sehr hohe Kosten aufgrund der dann notwendigen Hardwareanforderungen. Außerdem ist es schwierig bzw. kaum möglich, unstrukturierte Daten, wie zum Beispiel Bilder, Dokumente usw., in diesen Datenbanken zu speichern bzw. Datensätze mit unterschiedlichem Aufbau abzulegen, denn die Tabellenstruktur macht es erforderlich, dass die Datensätze jeweils über die gleichen Attribute (Eigenschaften) verfügen müssen.

Big Data


Unter dem Begriff Big Data versteht man Datenmengen sowohl strukturierter als auch unstrukturierter Art, die zu umfassend, zu komplex oder zu schnelllebig sind, um sie mit herkömmlichen Methoden der Datenverarbeitung auszuwerten. Ursprünglich stammt der Begriff aus den frühen 2000er Jahren. Branchenanalyst Doug Laney hat den Begriff Big Data verwendet, um sein 3-V-Modell zu formulieren: Volumen, Velocity und Variety sind dabei die drei entscheidenden Faktoren.

- **Volume:** Gemeint sind große Mengen Daten, die aus einer Vielzahl unterschiedlicher Quellen stammen.
- **Velocity:** Die Datenströme bewegen sich sehr schnell und müssen zeitnah verarbeitet werden.

- **Variety:** Die Daten liegen in unterschiedlichsten Formaten vor. Später wurde das Modell um weitere Faktoren erweitert, d. h. um:
- **Veracity:** Damit ist die Echtheit der vorliegenden Daten gemeint.
- **Value und Validity:** Diese Begriffe stehen für einen unternehmerischen Mehrwert und die Sicherstellung der Datenqualität.

Anwendung findet Big Data in fast allen Lebensbereichen. Große, oft unstrukturierte Datenmengen werden analysiert und somit Erkenntnisse gewonnen, die als Grundlage für bessere Entscheidungen dienen. Diese Form der Massendatenverarbeitung benötigt hinsichtlich der Datenbank flexible Speicherkonzepte.

Abb. 2:
Strukturierte, semistrukturierte und unstrukturierte Daten

<u>Strukturierte Daten</u>	<u>Semistrukturierte Daten</u>	<u>Unstrukturierte Daten</u>						
<ul style="list-style-type: none"> • Feste Struktur • Klare Bezeichnung der Spalten 	<ul style="list-style-type: none"> • Tragen einen Teil der Strukturinformation mit sich 	<ul style="list-style-type: none"> • Keine feste Struktur 						
<table border="1"> <thead> <tr> <th>ID</th><th>Name</th><th>Vorname</th></tr> </thead> <tbody> <tr> <td>1</td><td>Müller</td><td>Klaus</td></tr> </tbody> </table>	ID	Name	Vorname	1	Müller	Klaus	<pre><Nm> Müller Klaus </Nm> <Strasse> Marktgasse </Strasse></pre>	
ID	Name	Vorname						
1	Müller	Klaus						

für Datenbankkonzepte verwendet, die eine Alternative zum relationalen Modell bilden.

NoSQL-Datenbanken nutzen beispielsweise Objekte, Dokumente, Wertepaare oder Listen und Reihen für die Organisation der Daten. Es existieren keine festgelegten Tabellenschemata. Im Vergleich zu den relationalen Datenbanken sind NoSQL-Datenbanken sehr flexibel einsetzbar und für große Datenmengen gut geeignet. Oft werden die NoSQL-Datenbanken auch als „strukturierte Datenspeicher“ (Structured Storage) bezeichnet. Da NoSQL für mehrere Arten von Datenbanktechnologien steht, kann man diese Art von Datenbanksystemen nach unterschiedlichsten Kriterien klassifizieren. Eine sinnvolle Einteilung ist die Unterscheidung nach dem Datenbankmodell. Die wichtigsten Datenbankmodelle von NoSQL-Datenbanken sind Key-Value-Datenbanksysteme, dokumentenorientierte Datenbanksysteme, Column-Family-Datenbanksysteme und Graphdatenbanken. Wir sehen uns diese Varianten jetzt genauer an.

Key-Value-Datenbanksysteme

In Key-Value-Datenbanksystemen werden die Daten in Paaren gespeichert, die sich aus einem eindeutigen Schlüssel (Key) und einem zugeordneten Wert (Value) zusammensetzen. Die Eindeutigkeit des Schlüssels kann sich auf eine ganze Datenbank oder einen bestimmten Namensraum (Key Space) beziehen. Sowohl beliebige strukturierte als auch semistrukturierte und unstrukturierte Daten können im Value gespeichert werden. Das Datenformat kann ebenso beliebig sein, beispielsweise XML, Textdaten usw. Key-Value-Paare werden durch die Angabe eines Keys oder eines Key-Bereichs aus der Datenbank gelesen. Ein Zugriff innerhalb des Values ist in der Regel nicht möglich, da das Datenbanksystem über keine expliziten Informationen über den Inhalt des Values verfügt. Geeignet sind solche Datenbanksysteme für Anwendungen, bei denen Daten über einen Key aus einer Datenmenge selektiert oder gespeichert werden müssen. Nicht geeignet sind diese Systeme für alle diejenigen Anwendungen, bei denen der Zugriff auf den Inhalt des Values unterstützt werden muss. Kommen wir zu einem Beispiel [2]:

```
[007, "titel:'Abloeserekord';datum:'2013-09-01';
      'text:'Mega-Transfer perfekt ...';datum:'2013-09-01';
      'text:'Endlich - er wird ...';datum:'2013-09-01';
```

```
'text':'Unglaublich ... " ]
```

```
[008, "titel:'Traum-Transfer';datum:'2013-09-0';
      'text':'Die Fans applaudieren ... " ]
```

Wir haben es hier mit einem Datenschema der Form [Key, Value] zu tun. Die beiden Beispielobjekte sind über die Keys 007 und 008 zu erreichen. Es fällt auch sofort auf, dass die Objekte vollständig unterschiedliche Daten enthalten können. Welche Daten in welchem Format in den Values abgelegt werden, spielt keine Rolle. Zwei Varianten kann man bei den Key-Value-Datenbanksystemen unterscheiden. Die In-Memory-Variante behält die Daten im Speicher und sorgt damit für eine hohe Performance. Eine typische Anwendung sind Cachespeichersysteme, d. h. die Zwischenspeicherung von umfassenden Datensätzen im Arbeitsspeicher. Daneben existiert die On-Disk-Variante. Hier werden die Daten direkt auf der Festplatte abgelegt. Key-Value-Datenbanksysteme haben folgende Vorteile:

- eine sehr gute Skalierbarkeit,
- eine schnelle und effiziente Datenverwaltung,
- einfache Abfragen werden sehr schnell verarbeitet, d. h. der Zugriff über den Schlüssel ist sehr effizient.

Als Nachteil ist hauptsächlich zu nennen, dass komplexe Abfragen kaum möglich und umfassende Objektbeziehungsmodelle schwer darzustellen sind. Da innerhalb der Value-Einträge nicht gesucht werden kann, gibt es keine Möglichkeit, von einem Value auf einen Key zu schließen. Um solche Suchvorgänge dennoch zu erledigen, müssten diese Zusammenhänge als eigene Einträge in der Datenbank abgelegt werden. Benötigt man solche umgekehrten Abfragen häufig, wird das System schnell unübersichtlich und der Verwaltungsaufwand steigt stark an.

Key-Value-Datenbanksysteme kann man gut dort anwenden, wo große und unterschiedliche Datenmengen eindeutig über einen Key identifiziert werden müssen. Ein Beispiel ist die Datenspeicherung von Informationen in einem Warenkorb, wobei die User-ID den Key repräsentiert. Beispiele für Key-Value-Datenbanken sind: Oracle Berkeley DB, Amazon Dynamo und MemcachedDB.

Gut geeignet sind dokumentenorientierte Datenbanksysteme für sehr große Mengen strukturierter und semistrukturierter Daten.

Dokumentenorientierte Datenbanksysteme

Bei diesen Systemen werden die Daten ebenso in Key-Value-Paaren gespeichert, der Inhalt des Values ist jedoch nicht „undurchlässig“. Gespeichert werden Dokumente eines bestimmten Datenformats, und das Dokument selbst wird in diesem Fall nicht als unstrukturiertes Textformat verstanden. Die typischerweise unterstützten Datenformate sind JSON, seine binäre Repräsentation BSON oder XML. Die Daten werden als Folge von Property-Value-Paaren dargestellt. Unter Properties werden dabei die identifizierenden Bezeichner verstanden, die bei einigen Formaten eindeutig innerhalb eines Dokuments, bei anderen auch mehrwertig sein können. Interessanterweise sind sogar hierarchische Strukturen möglich: Es ist möglich, dass Values wiederum Properties enthalten. Anders als Key-Value-Datenbanksysteme, unterstützen dokumentenorientierte Datenbanksysteme auch die Auswahl von Dokumenten anhand der Werte der Properties. Die dafür notwendige Indexierung der Properties erfolgt bei manchen Systemen automatisch, andere Systeme überlassen die Auswahl der zu indexierenden Properties dem Anwender. Gut geeignet sind dokumentenorientierte Datenbanksysteme für sehr große Mengen strukturierter oder semistrukturierter Daten, vor allem in den Bereichen Contentmanagement, interaktive Webanwendungen und E-Commerce-Anwendungen. Ein Beispiel hierzu zeigt Listing [2].

Auch hier erfolgt der Zugriff auf die Dokumente über die Keys (`_id`: "007" und `_id`: "008"). Die Dokumente umfassen jedoch strukturierte Daten. Gewissermaßen könnte man ein Dokument mit einem Datensatz aus einer Tabelle einer relationalen Datenbank vergleichen. Der Unterschied ist jedoch, dass sich die Dokumente in der Struktur unterscheiden können. Ein Schema, das die gesamte Datenbank umfasst, gibt es nicht. Man spricht daher auch von Schemafreiheit. Die Struktur der zu speichernden Daten muss nicht vorab angegeben werden und kann sich von Dokument zu Dokument ändern. Um dem System eine effiziente Abfrageunterstützung zu ermöglichen, ist es allerdings sinnvoll, strukturell sehr unterschiedliche Dokumente getrennt zu speichern bzw. Dokumente mit einer ähnlichen Struktur zu Collections (Sammlungen) zusammenzufassen. Dokumentenbasierte Datenbanken unterstützen im Unterschied zu Key-Value-Systemen, die gezielte Auswahl eines Dokuments anhand der Werte seiner Properties. Das ist möglich, da die Datenbank in die Dokumente „hineinsehen“ kann. Dazu müssen die Properties vom System indexiert werden. Ebenso ist es möglich, einzelne Properties innerhalb eines Dokumentes zu ändern. Die Vorteile der dokumentenorientierten Datenbanksysteme sind:

- Existierende Dokumente können als Ganzes abgebildet werden.
- Die Datenbank kann sehr gut skalieren.

Ihre Nachteile sind:

- Durch die Schemafreiheit müssen die Anwendungsprogramme selbst für die korrekten Formate der Dokumente und ihrer Properties sorgen.
- Eine Abfragesprache wie SQL existiert nicht.

Dokumentenorientierte Datenbanksysteme werden dort eingesetzt, wo sehr große Mengen strukturierter oder semistrukturierter Daten verarbeitet werden sollen und ggf. eine Flexibilität des Schemas, d. h. der Struktur der Dokumente benötigt wird. Anwendungen gibt es im Bereich von Contentmanagementsystemen und Blogging-Plattformen. Ein Beispiel für eine dokumentenbasierte Datenbank ist MongoDB.

Column-Family-Datenbanksysteme

In diesem Datenbanksystem erfolgt die Datenspeicherung in einer oder mehreren Tabellen (Table). Dabei werden die Datensätze in Zeilen modelliert, die dann durch einen eindeutigen Schlüssel (Key) identifiziert werden können. Ein solcher Datensatz kann mehrere Attribute besitzen, die durch entsprechende Spaltenbezeichner (Column Qualifiers) ausgezeichnet werden. Die Spalten werden in Spaltenfamilien gruppiert (Column

Listing 1

```
// Kollektion blogposts
{
  "_id": "007",
  "titel": "Abloeserekord",
  "datum": "2013-09-01",
  "text": "Mega-Transfer perfekt. Fuer unglaubliche ...",
  "kommentare": [
    { "kom-datum": "2013-09-01",
      "kom-text": "Endlich - er wird ..." },
    { "kom-datum": "2013-09-01",
      "kom-text": "Unglaublich! Kein Spieler ..." } ]
  }
{
  "_id": "008",
  "titel": "Traum-Transfer",
  "datum": "2013-09-02",
  "text": "Die Fans applaudieren ...",
  "kommentare": [ ]
}
```

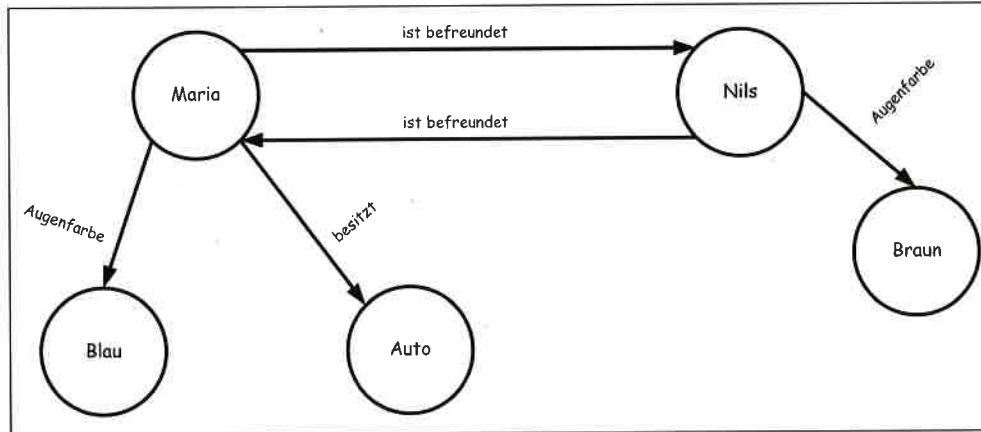


Abb. 3: Beispiel einer graphenorientierten Datenbank

Family). Solche Spaltenfamilien sind nicht nur eine logische Gruppierung der Spalten, sondern können auch zur physischen Aufteilung der Speicherung verwendet werden. Column-Family-Datenbanksysteme eignen sich für alle diejenigen Fälle, in denen sehr große Mengen strukturierter Daten verarbeitet werden sollen und Schemaflexibilität notwendig ist. Die Modellierung der Daten könnte – vereinfacht – wie in Tabelle 1 dargestellt aussehen.

In diesem Beispiel wurde pro Tabelle nur eine Spaltenfamilie angewendet. Natürlich können auch mehrere Spaltenfamilien pro Tabelle definiert werden (Tabelle 2).

Ist die Form der Datenmodellierung in einem Column-Family-Datenbanksystem nicht dem einer relationalen Datenbank ähnlich? Basiert die Modellierung nicht auch dort auf mehreren Tabellen, die über Primär- und Fremdschlüssel untereinander in Beziehung stehen? Es gibt einige entscheidende Unterschiede:

blogpost (Table)

blogpost_daten (Column Family)

id (Key)	titel	datum	text
007	Ablöserekord	2013-09-01	Großartige Aktion
008	Traum-Transfer	2013-09-02	Alle sind begeistert

kommentare (Table)

kommentar_daten (Column Family)

id (Key)	datum	text	id
123456	2013-09-01	Das kann nicht ...	007
123457	2013-09-02	Unglaublich, warum?	007

Tabelle 1: Modellierung eines Blogs in einem Column-Family-System [2]

Table

Column Family 1			Column Family 1			
id	titel	...	name	text
...
...

Tabelle 2: Prinzip einer Tabelle mit mehreren Spaltenfamilien

- Im Gegensatz zu den Spaltenfamilien müssen Spalten zum Zeitpunkt der Erstellung der Tabelle noch nicht festgelegt werden.
- Spalten können auch noch später, d. h. beim Einfügen eines neuen Datensatzes ergänzt werden (Schemafreiheit).
- Die Daten werden in Column-Family-Systemen nicht zeilenweise, sondern spaltenweise (Column-oriented) im Speicher abgelegt.
- Column-Family Datenbanksysteme weisen die folgenden Vorteile auf:

- Sie skalieren sehr gut.
- Es besteht ein schneller Zugriff auf einzelne Spalten und ihre Daten.

Es gibt folgenden Nachteil: Die Schreibzugriffe über mehrere Spalten sind vergleichsweise langsam. Wenn man zum Beispiel die Einträge *Name*, *Wohnort* und *Beruf* einer Person ändern möchte, dann muss man auf drei unterschiedliche Spalten zugreifen, die auch physisch an unterschiedlichen Orten gespeichert sein können. Beispiele für Column-Family-Datenbanksysteme sind Cassandra und HBase. Column-Family-Datenbanksystem eignen sich zum Beispiel für Anwendungen im Web wie auch für Contentmanagementsysteme.

Graphdatenbanken

Eine weitere Art von NoSQL-Datenbanken sind graphenorientierte Datenbanken bzw. Graphdatenbanken. Einige Autoren zählen sie wegen ihrer besonderen Art der Datenspeicherung und -modellierung auch als eigene Kategorie. Es handelt sich dabei um eine Datenbank, die Daten anhand eines Graphen darstellt und abspeichert. Graphen bestehen jeweils aus Knoten und Kanten. Kanten repräsentieren die Beziehung zwischen den Knoten. Zu den Knoten und Kanten können zusätzliche Attribute gehören. Außerdem besitzt jede Kante eine Eigenschaft, die die Beziehung definiert. Eine Graphdatenbank kann als eine Beziehungsdatenbank verstanden werden (Abb. 3).

Graphdatenbanken unterscheiden sich von anderen Arten der NoSQL-Datenbanken, denn sie verfolgen ein anderes Ziel. Key-Value-Datenbanksysteme, dokumentenorientierte Datenbanksysteme und Column-Family-Datenbanksysteme haben das Ziel, Systeme zu schaffen, die leicht auf größeren Clustern skaliert werden können bzw. die mit großen Datenclustern, die nur schwach miteinander verbunden sind, arbeiten können. Bei den Graphdatenbanken ist das anders: Es geht um den Umgang mit stark verknüpften Daten. Damit können sehr gut Zusammenhänge zwischen Beziehungsnetzwerken, wie sie bei Facebook, LinkedIn und bei Suchmaschinen

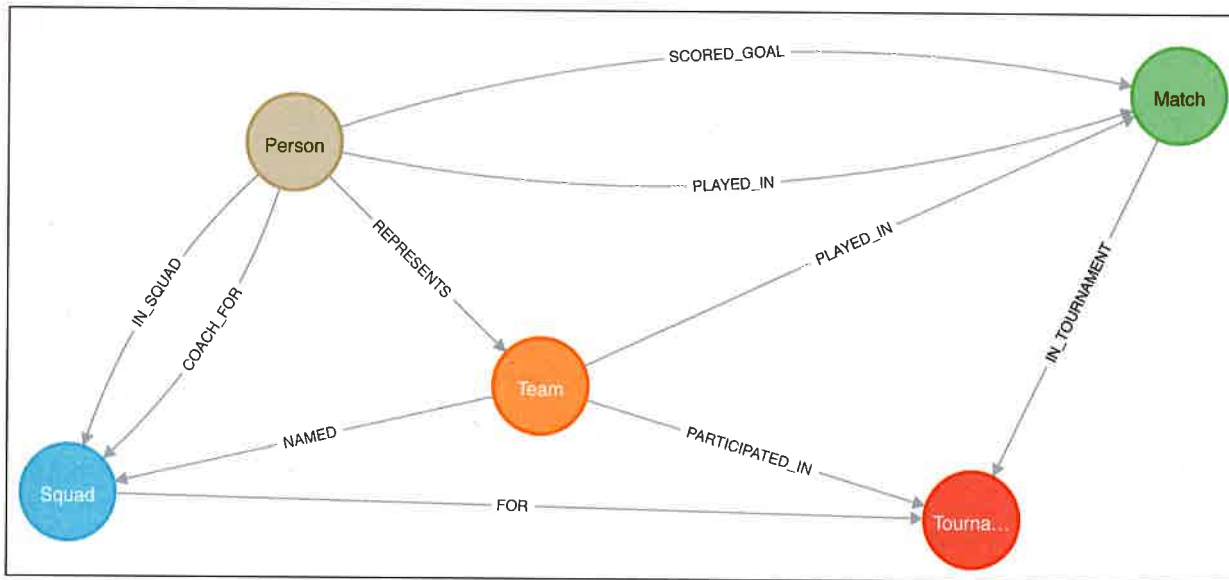
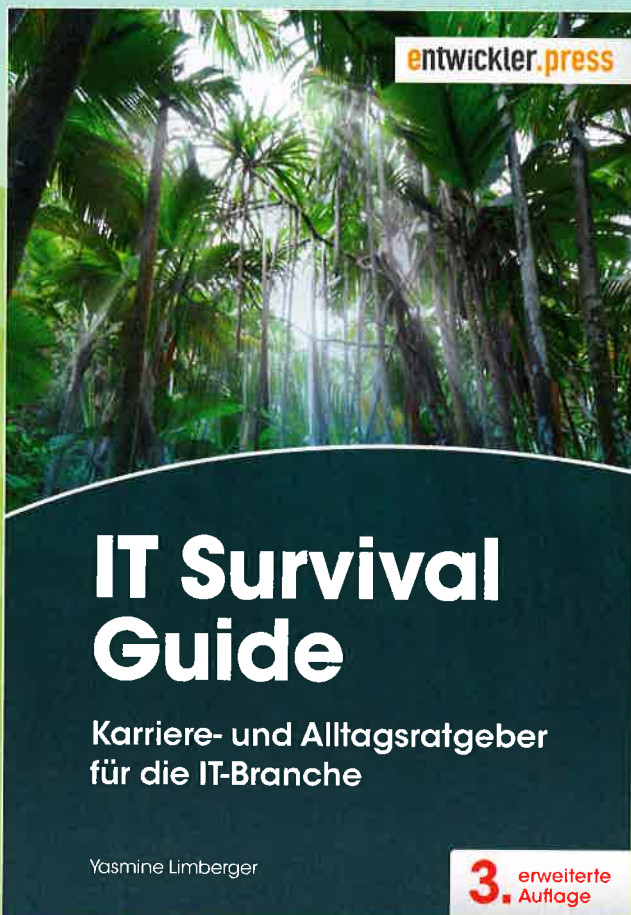


Abb. 4:
Schema der
Beispielda-
tenbank als
Graph

vorkommen, abgebildet werden. Der wichtigste Vorteil von Graphdatenbanken ist die Möglichkeit, vernetzte Informationen zu speichern und zu verarbeiten. Durch die Art der Speicherung kann man sehr gut durch den Graphen traversieren und damit eine hohe Performance erreichen, die für solche Anwendungszwecke besser ist als bei relationalen Datenbanksystemen. Als Nachteil ergibt sich, dass es noch keine einheitliche Abfragesprache gibt, d. h. jede Graphdatenbank verwendet eine andere Abfragesprache. Eine bekannte Graphdatenbank ist Neo4j.

Neo4j kann man direkt ausprobieren. Man kann sich entweder eine lokale Datenbank inklusive eines Desktops installieren oder direkt online Experimente durchführen [3]. Es stehen für Onlineexperimente einige interessante Musterdaten zur Auswahl. Dazu starten wir auf der Webseite die Neo4j Sandbox zu den Daten der Fußballweltmeisterschaft („Women’s World Cup 2019“). In der Sandbox können wir Befehle direkt an die Datenbank senden. Mit `CALL db.schema()` reproduzieren wir das Schema der Datenbank und sehen das Ergebnis als Graphen (Abb. 4).

Anzeige



**BEREIT FÜR DEN
IT-DSCHUNGEL?**

Yasmine Limberger
IT Survival Guide
Karriere- und Alltagsratgeber
für die IT-Branche

3. erweiterte Auflage, 292 Seiten, Mai 2017

ISBN: 978-3-86802-170-7

Preis: 29,90 € (D) / 30,80 € (A)

entwickler.press

www.entwickler.press.de

\$ MATCH (p:Person)-[:SCORED_GOAL]->(match)-[:IN_TOURNAMENT]->(tourn), (p)-[:REPRESENTS]->(team) WITH p, team, count(*) AS goals, apoc.coll.sort(collect(DISTINCT_

p.name	team	goals
"Marta"	"Brazil"	18
"Cristiane"	"Brazil"	11
"Carli Lloyd"	"USA"	10
"Christine Sinclair"	"Canada"	10
"Megan Rapinoe"	"USA"	9
"Alex Morgan"	"USA"	9
"Lisa De Vanna"	"Australia"	7
"Ellen White"	"England"	7
"Isabell Herlovsen"	"Norway"	6
"Eugenie Le Sommer"	"France"	5

Started streaming 10 records after 32 ms and completed after 32 ms

Abb. 5: Antwort der Datenbank in Form einer Tabellenstruktur

Andere Fragen an die Datenbank sind komplexer, wie das Beispiel in Listing 2 zeigt.

Die Abfrage „Wer sind die besten Spielerinnen (Anzahl der Tore), die an der diesjährigen Weltmeisterschaft teilnehmen?“ produziert eine Datentabelle. Das Schema der Antwort generiert die Datenbank eigenständig (Abb. 5).

Das vorgestellte Beispiel zeigt sehr gut, wo die Anwendungsoptionen für graphenorientierte Datenbanken liegen. Daten mit einem hohen Vernetzungsgrad können nach vielfältigen Aspekten ausgewertet werden. Weitere Beispiele auf der Webseite zeigen das ebenfalls, zum Beispiel die Auswertung von Tweets der Social-Media-Kanäle.

Relational vs. NoSQL

NoSQL-Datenbanken unterscheiden sich recht deutlich von relationalen Datenbanken. Daher lohnt es sich, nach der Betrachtung der einzelnen Ansätze der NoSQL-Datenbanken diese nochmalig herauszuarbeiten. Der größte konzeptuelle Unterschied ist die Flexibilität. NoSQL-Datenbanken sind nicht an eine fixe Tabellenstruktur gebunden. Ihre Schemafreiheit erlaubt einen flexibleren Einsatz. Man kann Änderungen der Datendefinitionen schneller durchführen. Die wesentlichen Unterschiede zwischen NoSQL- und SQL-Datenbanken sind:

1. **Tabellen vs. Sammlungen:** SQL-Datenbanken speichern Daten in Tabellenform und diese Tabellen werden dabei in Relation gesetzt. NoSQL-Datenbanken

speichern die Daten in anderer Form, zum Beispiel als flexible Schlüssel-Wert-Paare. Diese sind nicht statisch festgelegt, d. h. ein neues Schlüssel-Wert-Paar kann verwendet werden, ohne die eigentliche Datenstruktur zu verändern.

2. **Unterschiedlicher Normalisierungsgrad:** Die Normalisierung macht das SQL-Datenmodell statisch. Das Aufspannen der modellierten Struktur über verschiedene Tabellen erzeugt Last und Logik im Datenbanksystem. Die zusammenhanglose Datensammlung von NoSQL-Daten ist dagegen wesentlich einfacher.
3. **Anforderungen an die Skalierung:** Die Skalierung eines SQL-basierten Systems ist deutlich komplexer als bei NoSQL-Systemen. Das hängt damit zusammen, dass die Transaktionen erst abgeschlossen werden müssen, bevor sie auf andere Bereiche repliziert werden können. Bei lesenden Zugriffen ist es so, dass sie nicht nur auf den Transaktionsabschluss warten, sondern auch auf die Replikation der Datensätze. Bei NoSQL-Systemen werden Kopien der Datensätze erzeugt, die über verschiedene Knoten verteilt werden.
4. **ACID vs. BASE:** SQL-basierte Datenbanken sind stets konsistent in ihren Strukturen und Transaktionen. Da die lesenden Abfragen auf die schreibenden Vorgänge warten, sind die Daten bei jeder Abfrage des Systems in der Regel nachvollziehbar und gleich. NoSQL-Systeme können, je nach Status der Synchronisation, unterschiedliche Ergebnisse liefern. SQL-Systeme setzen die ACID-Grundprinzipien um, NoSQL-Systeme beschränken sich auf die BASE-Prinzipien (Kasten: „ACID vs. BASE“)

Listing 2: Abfrage an eine Graphdatenbank

```
MATCH (p:Person)-[:SCORED_GOAL]->(match)-[:IN_TOURNAMENT]->(tourn),
      (p)-[:REPRESENTS]->(team)
WITH p, team, count(*) AS goals,
      apoc.coll.sort(collect(DISTINCT tourn.year)) AS years
WHERE (p)-[:IN_SQUAD]->()-[:FOR]->(:Tournament {year: 2019})
RETURN p.name, team.name AS team, goals
ORDER BY goals DESC
LIMIT 10
```

Datenmodellierung in NoSQL-Datenbanksystemen

In relationalen Systemen ist die Datenmodellierung ein über viele Jahre erforschtes und erprobtes Thema. Eine Haupteigenschaft ist dabei das Erreichen einer normalisierten Datenbasis (Normalisierung). In NoSQL-Datenbanken sieht das anders aus. Die Datenmodellierung soll auch eine horizontale Skalierbarkeit unterstützen. Dazu müssen zusammengehörige Daten auf einen Rechenknoten gespeichert werden. Das bedeutet, dass die Operationen nicht über mehrere Knoten hinweg ausgeführt werden. Der Begriff der „aggregatororientierten Modellierung“ wurde aus dem Bereich des Domain-driven

Designs übernommen und steht für die Strukturierung der Daten in zusammengehörige Einheiten. Ein Datenmodell wird in geeignete Aggregate zerlegt. Dabei muss überlegt werden, welche Daten eingebettet und welche getrennt, d. h., referenziert gespeichert werden. Der Vorteil der eingebetteten Speicherung ist, dass häufig zusammen benötigte Daten auch zusammen gespeichert werden und damit mit einer einzigen Operation gelesen werden können (bessere Performance).

Die Nachteile entstehen natürlich, wenn diese Daten nicht immer zusammen benötigt werden. Bei einer 1:1-Beziehung ist eine eingebettete Speicherung ohne Redundanz möglich. Bei einer M:N-Beziehung führt die Einbettung der Daten zu Redundanz. Im Fall einer 1:N-Beziehung ist die Redundanz von der Richtung der Einbettung abhängig. In diesem Fall gilt: Es ist sorgfältig zwischen der Vermeidung von Anomalien der Datenstruktur und einem möglichen Gewinn an Performance abzuwägen. Außerdem kann es problematisch werden, wenn die Lebensdauer der Daten unabhängig voneinander ist, d. h., wenn es sich um eine nicht existenzabhängige Beziehung zwischen den Daten handelt. Das Löschen der übergeordneten Daten bedeutet, dass die eingebetteten Daten auch entfernt werden. Dann kann es auch vorkommen, dass noch benötigte Daten versehentlich gelöscht werden [2].

Fazit und Ausblick

NoSQL-Datenbanken sind eine alternative Form für die Speicherung von umfassenden Daten, die sich nicht in eine feste Tabellenstruktur pressen lassen. Ihre Bedeutung wird größer, desto mehr unstrukturierte Daten wir mit Software verarbeiten und analysieren müssen.

Im kommenden zweiten Teil der Artikelserie werfen wir einen Blick in die Praxis und sehen uns an, wie man mit NoSQL-Datenbanken aus PHP arbeitet.



Dr. Veikko Krypczyk ist Entwickler und Fachautor.



Elena Bochkor beschäftigt sich mit dem Design von Webseiten und Apps für mobile Geräte.

Weitere Informationen zu diesen und anderen Themen der IT finden Sie unter <http://larinet.com>.

Links & Literatur

- [1] <https://de.statista.com/statistik/daten/studie/819500/umfrage/prognose-zum-weltweit-gespeicherten-big-data-datenvolumen-in-rechenzentren/>
- [2] Kudraß, Thomas: „Taschenbuch Datenbanken“, Hanser Verlag, 2015
- [3] <https://neo4j.com/>
- [4] <https://www.bigdata-insider.de/was-ist-acid-a-776182/>
- [5] <https://db-engines.com/de/article/BASE>
- [6] <https://www.computerwoche.de/a/nosql-die-neue-datenbankgeneration,1235662>

ACID vs. BASE

Das ACID-Konzept besteht aus vier einzelnen Grundprinzipien [4]:

- **Atomicity (Atomarität):** Ausführung aller oder keiner Teile einer Transaktion. Eine Transaktion besteht in der Regel aus einzelnen Aktionen. Es müssen entweder alle Einzelschritte komplett oder gar nicht ausgeführt werden. Tritt während der Ausführung ein Fehler auf, muss das System dafür sorgen, dass die bereits durchgeführten Änderungen zurückgenommen werden.
- **Consistency (Konsistenz):** Transaktionen erzeugen einen gültigen Zustand oder fallen in den alten Zustand zurück. Führt eine Transaktion zu einem inkonsistenten Zustand, zum Beispiel durch ungültige Verweise zwischen den Tabellen, durch Werte außerhalb der zulässigen Bereiche usw., muss die Transaktion komplett zurückgenommen werden. Nach der Transaktion muss die Datenbank in einem konsistenten Zustand sein. Während der Transaktion kann die Datenbank inkonsistente Zustände aufweisen.
- **Isolation (Abgrenzung):** Transaktionen verschiedener Anwender oder Prozesse bleiben voneinander isoliert. Die Isolation stellt sicher, dass die Nutzung der Datenbank durch mehrere Anwender keine negativen Auswirkungen nach sich zieht. Zum Beispiel ist ein

gegenseitiges Überschreiben von Werten in der Datenbank zu verhindern. Das Datenbankmanagementsystem ist für die Einhaltung dieser Regeln zuständig und erledigt das über Sperrverfahren.

- **Durability (Dauerhaftigkeit):** Nach einer erfolgreichen Transaktion bleiben die Daten dauerhaft gespeichert. Spätere Fehler, wie Systemausfälle usw., dürfen nicht dazu führen, dass Daten gelöscht und nicht mehr hergestellt werden. Über ein Logging können Transaktionen auch nach einem Systemausfall reproduziert werden.

Für NoSQL-Datenbanken gelten dagegen die abgeschwächten BASE-Regeln [5], [6]. BASE steht für:

- **Basically (B) Available (A):** Das System ist grundsätzlich verfügbar. Es wird allerdings toleriert, dass einzelne Teile ausfallen können.
- **Soft State (S), Eventual (E) Consistency:** Das System erhält am Ende konsistente Daten. Im Lauf einer Transaktion können aber inkonsistente Zustände auftreten, die auch an eine Anwendung ausgeliefert werden können.

Das Konzept BASE gibt also die absolute Konsistenz auf. Stattdessen wird die Verfügbarkeit des Systems erhöht und ein zwischenzeitlich unvermeidbarer undefinierter Zustand in Kauf genommen.