# 5 NoSQL-Datenbanksysteme

## 5.1 Lernziele dieses Kapitels

- Sie verstehen die **Motivation** für die Entwicklung und den Einsatz von **NoSQL-Datenbanksystemen**.
- Sie kennen die wichtigsten **Kategorien von NoSQL-Datenbanksystemen** sowie deren wichtigste Eigenschaften und Einsatzgebiete.
- Sie können einschätzen, welche Kategorie von NoSQL-Datenbanksystemen für einen **Anwendungsfall** am besten geeignet ist.

## 5.2 Einführung

Am Anfang dieses Moduls wurde der Einsatz von Datenbanksystemen motiviert [Elm09, Kap. 1]. Dabei wurden Aspekte wie Universalität, Datenunabhängigkeit, Standardisierung, Mehrbenutzersynchronisation und Robustheit hervorgehoben. Die Datenbanksysteme, von denen dabei gesprochen wird, sind hauptsächlich **relationale Systeme mit SQL als Datendefinitions- und Anfragesprache**. Offenbar sind diese relationalen Datenbanksysteme für viele Anwendungsfälle gut geeignet, stellen sie doch seit mehreren Jahrzehnten den De-Facto-Standard insbesondere bei Geschäftsapplikationen dar.

Dennoch gibt es Situationen, in denen relationale Systeme an ihre Grenzen stoßen. Sadalage und Fowler [Sad13, Kap. 1] machen dafür mehrere Gründe aus:

- **Impedanzfehler:** Wie in Kurseinheit DBS 8 besprochen, sind relationale Datenbanksysteme nicht unmittelbar für die Speicherung von Objektgraphen geeignet, wie sie in objektorientierten Applikationen vorherrschen. Es muss daher immer eine Abbildungsschicht zwischen objektorientierter und relationaler Welt geben.
- **Applikationsspezifische Datenbanken:** Relationale Datenbanken können als zentrale Datenhaltung für viele Applikationen dienen. Standardisierung sowie SQL als einheitliche Sprache trugen dazu bei. Relationale Datenbanken dienen so der Applikationsintegration.
  Ein neuerer Trend geht allerdings in Richtung applikationsspezifischer Datenbanken, da große zentrale Datenbanken, die viele Applikationen

bedienen, über lange Zeit sehr schwer zu warten sind. Jede Applikation bringt dabei ihre eigene Datenhaltung mit, die nach den Erfordernissen der jeweiligen Anwendung gestaltet sein kann. Dabei geht der Trend zu sogenannten **Microservices**, d.h. fachlich eng begrenzten Applikationen, die untereinander lose gekoppelt sind. Die Integration der Services geschieht nicht mehr über eine gemeinsame Datenbank, sondern über Service-Schnittstellen oder über den Austausch asynchroner Nachrichten.

- **Skalierbarkeit:** Seit dem Aufkommen des World Wide Web Mitte der 1990er Jahre sind viele Anwendungen in „Web Scale" zu denken – es kann also sein, dass ein erfolgreiches Angebot plötzlich Millionen von Nutzern hat. Die globale Verbreitung mobiler Endgeräte verstärkt diesen Trend. Relationale DBS sind in diese Größenordnungen nicht gut über mehrere Rechner zu skalieren, da das relationale Modell aufgrund von Fremdschlüsseln und Joins die Verteilung erschwert. Es gibt Ansätze, auch solche Operationen in relationalen Systemen zu verteilen, diese sind aber für „Web Scale" nicht leistungsfähig genug.

**Big Data**

In diesem Zusammenhang wird von **Big Data** gesprochen. Eine gängige Definition[15] sieht dafür folgende Kriterien, die sogenannten **„3 V"**:

- **Volume:** Es sind **große** Datenmengen zu verarbeiten, also z. B. die Daten von Millionen Nutzer*innen einer Applikation.
- **Velocity:** Die Daten müssen **schnell** verarbeitet werden. Das kann sowohl eine schnelle Reaktionszeit als auch einen hohen Durchsatz bedeuten.
- **Variety:** Datenbestände sind heute sehr **vielfältig**. Strukturierte Daten, z. B. aus relationalen Systemen, müssen ebenso wie semistrukturierte und unstrukturierte Daten verarbeitet werden.

Um diesen Herausforderungen zu begegnen, sind vielfach Datenbanksysteme entstanden, die sich vom relationalen Modell und den darauf gängigen Operationen der relationalen Algebra bzw. SQL ganz oder teilweise lösen und auf andere Datenstrukturen setzen. Das kann sowohl der Skalierbarkeit als auch der Flexibilität der Datenrepräsentation dienen.

**NoSQL**

Für diese neuartigen Datenbanksysteme hat sich der Begriff **NoSQL** etabliert. Anfangs als „kein SQL" gelesen, geht es allerdings weniger um SQL als Anfragesprache als um eine Abkehr vom relationalen Modell und seinen Operationen. Tatsächlich wird NoSQL heute als **„Not only SQL"** gelesen; manche NoSQL-Systeme bieten Anfragesprachen für nichtrelationale Datenmodelle an, die sehr ähnlich zu SQL sind.

---

15 https://www.gartner.com/en/information-technology/glossary/big-data

## 5.3 Kategorien von NoSQL-Datenbanksystemen

Während die verschiedenen RDBMS einander recht ähnlich in ihrem Aufbau und ihren Möglichkeiten sind, sind NoSQL-Systeme naturgemäß viel heterogener. Es sind Dutzende verschiedener NoSQL-DBMS auf dem Markt, die jeweils unterschiedlichen Fokus auf die 3 V haben. Eine Standardisierung wie bei den RDBMS gibt es auf diesem Markt (noch) nicht, so dass es derzeit recht aufwändig wäre, ein bereits etabliertes NoSQL-DBMS in einem Projekt durch ein anderes zu ersetzen. Andererseits haben sich Kategorien von NoSQL-Systemen herausgebildet, in denen sich die Angebote jeweils ähneln.

### 5.3.1 Key-Value-Stores

In diesen DBMS ist die zentrale Datenstruktur eine Abbildung von Namen auf Werte. Als Beispiel kann man eine Abbildung von Kundennummern (Name) auf Kundendatensätze (Wert) heranziehen:

**Tabelle 9: Beispiel Key-Value-Store**

| Name (Kundennummer) | Wert (Kundendaten) |
|---|---|
| 1233 | Peter Meier, Hauptstraße 1, Trier |
| 1543 | Frauke Müller, Marktplatz 7, Karlsruhe |
| 3243 | Ludwig Schmitt, Wilhelmstraße 17, Bottrop |

Solche Datenstrukturen sind als Abbildungstabellen, Caches oder für temporäre Sitzungsdaten sehr häufig anzutreffen. In einer relationalen Datenbank würde man diese Daten als Tabelle mit dem Namen als Primärschlüssel und den Werten als weitere Attribute darstellen. Datenbanksysteme jedoch, die nur solche Name-Wert-Paare verwalten müssen, auf die ausschließlich über den Namen zugegriffen wird, können viel effizienter damit umgehen als ein universelles relationales System.

Die sogenannten Key-Value-Stores gehören damit zu den Systemen, die sowohl in Bezug auf die Antwortzeiten als auch auf die Skalierbarkeit auf viele Rechner besonders leistungsfähig sind. Bekannte Beispiele für solche Systeme sind **Redis**, **Riak** und **Aerospike**.

### 5.3.2 Dokumentenorientierte DBMS

Diese Kategorie von DBMS verwendet als Datenstruktur Sammlungen von **Dokumenten**. Damit sind allerdings nicht Textdokumente gemeint, sondern in der Regel Daten in JSON wie in Abschnitt 3.3 eingeführt, z. B.:

```
{ _id: 17,
  name: "Sabine Meyer",
  adresse: { strasse: "Hauptmarkt 1", plz: "54290",
             ort: "Trier" },
  bestellungen: [
    { renr: 17-1,
      datum: 2022-08-10,
      positionen: [...],
    }, { ... }
}
```

Ein Vorteil dieser Dokumente ist, dass sie in ihrer Struktur flexibel sind. In diesem Beispiel könnte es etwa Kund*innen mit Bestellungen geben oder solche ohne. Es könnten auch völlig unterschiedliche Dokumente gespeichert werden, etwa solche über Kund*innen und solche über Lieferanten.

Zunächst kann eine Dokumentendatenbank wie ein Key-Value-Store genutzt werden, dessen Werte Dokumente sind: gegeben ein Schlüssel (im Beispiel `_id: 17`), gebe das Dokument mit diesem Schlüssel zurück. Es ist aber auch möglich, Dokumente nach ihren Inhalten anzufragen. Dazu bedient man sich wieder der **Pfadausdrücke** wie in Kapitel 3 und 4 gezeigt.

Bekannte dokumentenorientierte DBMS sind **MongoDB**, **CouchDB** und **Elasticsearch**.

### 5.3.3 Wide-Column-Stores

Die sogenannten Wide-Column-Stores speichern Daten in einer Struktur, die relationalen Tabellen ähnelt, wobei allerdings jede Zeile **unterschiedliche** Spalten haben kann. Aus einer relationalen Sicht kann man sich das vorstellen als eine Tabelle mit sehr vielen Spalten und entsprechend vielen Nullwerten:

**Tabelle 10: Beispiel Wide-Column-Store**

| KdNr | Name | Alter | Ort | Hose? | Jacke? | … |
|---|---|---|---|---|---|---|
| 1 | Meier | 32 | - | - | ja | … |
| 2 | Müller | - | Trier | Ja | - | … |

Hier wird also repräsentiert, dass Kundin 1 Meier heißt, 32 Jahre alt ist und eine Jacke gekauft hat; ihr Wohnort ist nicht erfasst. Kunde 2 heißt Müller, kommt aus Trier und hat eine Hose gekauft; sein Alter ist unbekannt.

Repräsentiert man nur die Spalten, die nicht leer sind, so hat jedes Tupel ein anderes Schema:

**Tabelle 11: Beispiel Wide-Column-Store**

| KdNr | Name | Alter | Jacke? | … |
|---|---|---|---|---|
| 1 | Meier | 32 | ja | … |

| KdNr | Name | Ort | Hose? | … |
|---|---|---|---|---|
| 2 | Müller | Trier | Ja | … |

Die Besonderheit bei Wide Column Stores ist, dass man **sehr viele Spalten** verwenden kann, z. B. für jeden Artikel im Katalog eine, um anzuzeigen, dass Kund*innen den jeweiligen Artikel gekauft haben. Intern werden nur diejenigen Spalten repräsentiert, die für ein Tupel jeweils befüllt sind.

Dadurch ergeben sich neue Möglichkeiten, Daten zu **modellieren**. Anstatt für dieses Beispiel eine M:N-Beziehung zwischen Kunden und Artikeln über eine separate Tabelle zu realisieren, führt man einfach bei jedem Kunden Spalten für alle seine Artikel auf. Insofern kann man Wide-Column-Stores wie Key-Value-Systeme betrachten: der Schlüssel eines Tupels ist der Key, der Wert die jeweils belegten Spalten und deren Inhalte.

Manche Systeme wie z. B. Cassandra bieten eine SQL-ähnliche Sicht auf diese Datenstrukturen. Die unterschiedlichen Spalten, die zunächst nicht ins Konzept der SQL-nahen Anfragesprache passen, werden dabei z. B. als Attribute mit einem strukturierten Datentyp `Map<Name, Inhalt>` aufgefasst.

Bekannte Vertreter der Wide Column Stores sind **Cassandra**, **HBase**, **Accumulo** und **Google BigTable**.

### 5.3.4 Spaltenorientierte und HTAP-DBMS

Ein bekanntes Problem relationaler DBMS ist, dass sie je nach Ausgestaltung des Schemas **entweder** besonders gut für einzelne Operationen **oder** für Analysen geeignet sind. Wie in der Kurseinheit DBS 9 diskutiert, werden daher oft unterschiedliche Datenmodelle für OLTP- und OLAP-Anwendungen verwendet.

**Spaltenorientierte DBMS** sind für OLAP-Anwendungen optimiert. Anstatt die Daten tupelweise zu speichern, speichern sie sie spaltenweise. Tabelle 12 und Tabelle 13 verdeutlichen den Unterschied in einem Anwendungsfall, in dem Auswertungen über Kundendaten erstellt werden sollen.

Zunächst zeigt Tabelle 12, dass die Daten nicht normalisiert sind. Die Stadt bestimmt das Bundesland und das Bundesland das Land. In 3NF oder BCNF müsste man diese Tabelle entsprechend zerlegen, was aber bei Auswertungen zwei zusätzliche Joins bedingen würde.

Weiterhin sind Tabellen für OLAP-Anwendungsfälle oft sehr breit, haben also Dutzende oder Hunderte von Spalten. Für eine Anfrage der Art „Wie viele Kunden gibt es pro Bundesland“ müssten daher sehr große Datenmengen eingelesen werden, von denen dann nur eine Spalte benötigt wird.

**Tabelle 12: Zeilenweise Speicherung**

| **Kunden** | | | | |
|---|---|---|---|---|
| **kdnr** | **stadt** | **bland** | **land** | **…** |
| 1 | Hannover | NDS | D | … |
| 2 | Bochum | NRW | D | … |
| 3 | Dallas | TX | USA | … |
| … | … | … | … | … |

**Tabelle 13: Spaltenweise Speicherung**

| **stadt** |
|---|
| Hannover |
| Bochum |
| Dallas |

| **bland** |
|---|
| NDS |
| NRW |
| TX |

| **land** |
|---|
| D |
| D |
| USA |

Bei einer spaltenweisen Speicherung wie in Tabelle 13 gezeigt müsste man für diese Art der Anfrage nur die Spalten einlesen, die tatsächlich benötigt werden; hier wäre dies die Spalte `bland`. Zudem lassen sich Spalten, die wenige verschiedene Werte enthalten, sehr gut komprimieren, so dass jede Ausprägung der Spalte mit wenigen Bit dargestellt werden kann.

Rein spaltenorientierte DBMS sind eher selten. Vertreter sind **MonetDB**, **Vertica, Druid** und **Kudu**. Häufiger sind sogenannte **HTAP-DBMS**; HTAP steht dabei für „Hybrid Transactional/Analytical Processing“. Hierbei werden die Daten innerhalb des Systems so repräsentiert, dass sie sowohl für Einzeltransaktionen als auch für aggregierte Analysen effizient verwendet werden können. Dabei werden zum Teil spalten- und zeilenorientierte Repräsentationen parallel vorgehalten, müssen dann aber bei Änderungsoperationen abgeglichen werden. Ein Vertreter dieses Ansatzes ist **SAP HANA**.

### 5.3.5 DBMS für Graphen

Ein Graph ist in der Informatik eine Struktur bestehend aus einer Menge von Knoten sowie einer Menge von Kanten, die jeweils zwei Knoten verbinden:

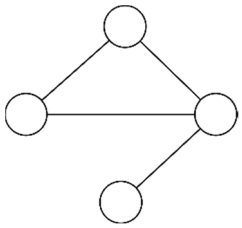

**Abbildung 4: Beispiel eines Graphen**

Abbildung 4 zeigt ein Beispiel für einen (ungerichteten) Graphen mit vier Knoten und vier Kanten.

Graphen eignen sich als Datenmodell, wenn die zu repräsentierenden Daten eine sehr freie Struktur haben oder sich diese Struktur leicht ändern können soll, ohne aufwändige Schemaanpassungen vornehmen zu müssen.

Für Datenbankanwendungen erweitert man die Definition eines Graphen aus der theoretischen Informatik zu einem **Property-Graphen**, in dem man Knoten und Kanten beliebige Eigenschaften, sogenannte **Properties**, zuweisen kann. Dieses Modell ähnelt dann einem ER-Modell: Knoten repräsentieren Entitäten, Kanten Beziehungen zwischen diesen, und Properties entsprechen den Attributen.

Graphbasierte DBMS nutzen Property-Graphen als Datenmodell. Dabei hat man die Möglichkeit, die gültigen Graphen über ein Schema einzuschränken. Man kann also, ähnlich wie im ER-Modell, festlegen, dass z. B. ein bestimmter Typ von Kante (entsprechend einem Relationshiptyp) nur bestimmte Typen von Knoten (entsprechend Entitytypen) verbinden darf. Ein Vorteil von dieser graphbasierten DBMS ist jedoch, dass man ein solches Schema nicht haben **muss**, sondern genau so viele Einschränkungen treffen kann, wie der aktuelle Anwendungsfall benötigt.

Bekannte Vertreter für graphbasierte DBMS sind **Neo4J**, **ArangoDB** und **Amazon Neptune**.

## Übungsaufgaben

5.1 Geben Sie für die folgenden Anwendungsfälle jeweils eine geeignete Art von NoSQL-Datenbank an:

a. In einem Online-Spiel werden für die aktiven Spieler unter sehr hoher Last die aktuellen Sitzungsdaten vorgehalten. Die Daten werden nicht persistiert.
b. In einem Krankenhaus fallen Anästhesieprotokolle an. Wichtig ist, dass alle Daten auf diesen Protokollen repräsentiert werden, auch diejenigen, die nicht in den vorgesehenen Formularfeldern stehen.
c. In einem sozialen Netzwerk werden die Beziehungen zwischen Personen repräsentiert. Darauf werden mit Graphalgorithmen Empfehlungen berechnet.
d. Ein sehr großer Online-Shop analysiert das Kaufverhalten seiner 50 Mio. Kunden. Jeder Kunde und jeder Artikel ist durch ca. 100 Attribute beschrieben.

## Zusammenfassung

NoSQL-Datenbanksysteme wurden für solche Anwendungsfälle entwickelt, in denen relationale DBS an ihre Grenzen stoßen. Dabei geht es hauptsächlich um die Aspekte **Flexibilität** und **Skalierbarkeit**.

Durch ihre nichtrelationalen Datenmodelle wie Graphen oder semistrukturierte Dokumente sind viele NoSQL-Systeme in der Lage, **stark variierende Datenstrukturen** aufzunehmen, ohne dass dafür jeweils Schemata geändert werden müssten.

Andererseits **reduzieren** viele NoSQL-Systeme den Umfang der angebotenen Operationen und bieten z. B. keinen Join an, oder das Datenmodell wird so gestaltet, dass es sich besonders einfach implementieren lässt wie bei den Key-Value-Stores. Auch die Garantien bezüglich Transaktionen werden oft gegenüber den vollen ACID-Eigenschaften eingeschränkt.

Dadurch werden besonders **effiziente und meist auch verteilte Implementierungen** möglich, wodurch sich NoSQL-DBS oft sehr gut **skalieren** lassen. In relationalen DBS mit den vollen Möglichkeiten wie Fremdschlüssel, Joins, Transaktionen ist dies nur schwer möglich.