NoSQL

Im Zuge der rasch wachsenden Informationsflut, hauptsächlich getrieben durch Web 2.0-Medien, der die klassischen Datenbanksysteme nicht mehr gewachsen sind, haben sich unter dem Begriff „NoSQL“ zur effizienteren Verarbeitung großer Mengen an unstrukturierten Daten eine Reihe neuer Datenbankentechnologien entwickelt. Ursprünglich durch Carlo Strozzi geprägt, entstand der Begriff „NoSQL“ im Jahre 1998 als Name für eine relationale Datenbank, die auf UNIX-Dateimanipulationen zur Datenhaltung setzte und eben **kein SQL** als Abfragesprache nutzte (Strozzi, 2007). Im Unterschied dazu steht aber die NoSQL-Bewegung, die **nicht-relationale** Datenbanken meint, die auf verteilten Systemen liegen und nicht die strengen ACID-Konsistenzrichtlinien befolgen (Gull 2012: S. 18). Die Bezeichnung steht hier für „**not only SQL**“ und fand erstmals 2009 als Schlagwort für eine Veranstaltung zum Thema verteilter nichtrelationaler Datenbanken, organisiert durch Johan Oskarsson, in San Francisco Verwendung (Evans, 2009). Die Bezeichnung „NoSQL“ ist ein nicht ganz treffender Begriff, da es nicht unbedingt etwas mit der Sprache SQL zu tun hat und manche NoSQL-Systeme teilweise sogar SQL-Abfragen unterstützen. Vielmehr geht es um die darunterliegende Architektur des Datenmodells und ob dieses einem strikten Schema folgt. Strozzi merkte dazu bereits an, dass „NoREL“ für „not relational“ wohl eine bessere Umschreibung abgegeben hätte (Strozzi, 2007).

Bereits vor der Entwicklung von SQL und dem relationalen Datenbankmodell durch IBM in den 70er Jahren gab es Datenbanken, die andere Datenmodelle nutzten, wie Netzwerkdatenbanken oder hierarchische Datenbanksysteme, die ebenfalls nichtrelational waren und auch kein SQL nutzten. So gesehen könnte man sie ebenfalls zu den NoSQL-Systemen zählen. Sie sind jedoch nicht damit gemeint. Die NoSQL-Bewegung entstand aus den Bedürfnissen heraus, die sich mit der Entstehung des Web 2.0 entwickelten und denen die traditionellen relationalen Datenbanken nicht mehr gerecht werden konnten.

Kriterien

Es existiert zwar kein allgemeiner Standard oder eine offizielle Definition des Begriffs, da er auch eher zufällig entstand, jedoch ist den meisten NoSQL-Systemen folgendes gemein:

- **Nichtrelationales Datenmodell**
- **Schemafrei** (oder nur schwache Restriktionen)
- **Bieten einfache API**
- **Verteilte Architektur, optimiert für einfache Replikation und horizontale Skalierung**
- **Kein ACID-Konsistenzmodell**
- **Open Source**

(Vgl. Sadalage/Fowler 2012: S. 12; Gull 2012: S. 18; Pürner 2013).

Nichtrelationales Datenmodell

Klassische relationale Datenbanksysteme erfassen strukturierte Daten, die durch Normalisierung aufgeteilt und dann in viele, durch Schlüssel-Beziehungen miteinander verknüpfte Tabellen gespeichert werden. Um diese verteilten Datensätze nun zu verwenden (z.B. für eine Bestellung), werden sie mit Hilfe des SQL JOIN-Operators anhand von Schlüsseln aus den unterschiedlichen

Tabellen (z.B. Artikel, Kundendaten) zusammengeführt. Diese Operation ist jedoch immer auch mit Plattenzugriffen verbunden und entsprechend zeitaufwendig. Je stärker ein Modell normalisiert wurde und je mehr Tabellen dadurch verknüpft werden müssen, desto mehr JOINs fallen auch an und drosseln die Performance und Reaktionszeit der Abfrage.

Derartige JOINs gibt es in NoSQL-Systemen nicht. Stattdessen umgehen einige NoSQL-Vertreter Referenzen durch JOINs, indem sie jene Informationen zweckmäßig als Aggregat zusammenfassen, die auch zusammen gespeichert, verwaltet und betrachtet werden sollen (Sadalage/Fowler 2012: S. 25). Das macht das Datenmodell wesentlich flexibler und ermöglicht so auch schnellere Schreib- und Lesezugriffe und vereinfacht ihre Verwaltung auf verteilten Systemen. Dafür nehmen sie Redundanzen in Kauf, da Plattenspeicher mittlerweile günstig ist und ausreichend zur Verfügung steht.

Sollen doch einmal Beziehungen hergestellt werden, so ließen sich in den Aggregaten entsprechende Key-IDs einfügen. Diese werden jedoch nicht vom System als solche erkannt und machen nach Lesen der ID einen erneuten Zugriff auf die Datenbank nötig, um das entsprechend referenzierte Aggregat zu laden.

Schemafreiheit

Das relationale Modell eignet sich bestens für den Umgang mit immer gleich strukturierten Datensätzen. Im Prozess der Datenmodellierung wird dafür ein Schema festgelegt, nach welchem sie in Attribute unterteilt in Tabellen abgespeichert werden. Mittels Normalisierung werden sie in atomare Einheiten (die kleinste zu bearbeitende Form) zerlegt und als Attribute einmal in einer Tabelle zusammengefasst abgespeichert. So gehören etwa zu einem Kunden Attribute wie der Name, Vorname, Adresse, usw. Eine solche Tabelle mit Kundendaten würde nur ein einziges Mal existieren. Dieses Vorgehen verhindert das redundante Abspeichern dieser Daten und erlaubt zudem eine einfache Aktualisierung, da etwaige Änderungen an nur einer Stelle vorgenommen werden müssen.

Allerdings bedeutet das Festlegen auf ein Schema auch, dass nur solche Daten gespeichert werden können, die auch dem vorher definierten Schema entsprechen. Änderungen sind zwar möglich (etwa mit dem SQL-Befehl ALTER TABLE), können sich aber, je nach Umfang des Systems, als schwierig, umständlich und zeitintensiv erweisen und das System für mehrere Minuten oder gar Stunden lahmlegen. Doch gerade im Big Data-Zeitalter fallen viele Daten an, die nur semi- oder ganz unstrukturiert sind und sich eben nicht so einfach in atomare Attribute zerlegen und in Tabellen abspeichern lassen. Um dieser Problematik zu begegnen, verzichten NoSQL-Systeme auf Schemarestriktionen und wollen so einen flexibleren Umgang mit unstrukturierten Daten ermöglichen. Sie erlauben es, Datenbankeinträge frei um zusätzliche Felder zu erweitern, ohne diese Änderung vorher im Schema festlegen zu müssen.

Sofern jedoch nicht sämtliche Daten eines Eintrags ausgegeben werden sollen, ist davon auszugehen, dass zumindest ein „indirektes Schema“ befolgt wird, das bspw. Annahmen darüber ermöglicht, dass Felder wie „Name“ oder „Preis“ mit entsprechenden Werten und Datentypen dahinter existieren, auf die zugegriffen werden kann. Doch die Datenbank selbst kann dieses nicht validieren, es beschränkt sich also auf Annahmen über die Datenstruktur innerhalb des Applikationscodes, der die Daten manipuliert (Sadalage/Fowler 2012: S. 29).

Einfache API

SQL wurde in den 70er Jahren (als „SEQUEL“ für „structured English query language“) entwickelt und

hat sich durch seine Verbreitung auf RDBMs zu einer Art Sprach-Standard für Datenbanken entwickelt und unterstützt Manipulieren und Verwalten von Daten in tabellarischer Form. Doch gerade dann, wenn Abfragen komplexer werden und viele JOINs und verschachtelte SELECTs anfallen, kann es schnell ziemlich unübersichtlich und somit fehleranfälliger werden. NoSQL-Systeme bieten zwar teilweise recht einfache Schnittstellen, jedoch sind diese oft nicht so mächtig. Abfragen gestalten sich aber teilweise, Dank der aggregierten Speicherform mancher NoSQL-Systeme, sowieso einfacher als bei SQL. Bei komplexeren Abfragen kommt oft die MapReduce-Technik zum Einsatz. Inwiefern das tatsächlich einfacher ist, bleibt offen. Wohl auch aufgrund der teils sehr unterschiedlichen Datenmodelle gibt es bislang keine einheitliche Abfragesprache.

Skalierbarkeit

Die Lasten auf moderne internetbasierte Angebote wie etwa Onlineshops, soziale Netzwerke, Medienplattformen und dergleichen wachsen ständig. Sie müssen 24 Stunden am Tag, 7 Tage die Woche verfügbar sein und sehen sich mit immer größeren Lasten durch eine Unzahl an Usern und enormen Datenmengen mit vielen Transaktionen konfrontiert. Um bei solchen Lasten noch reibungslose Abläufe garantieren zu können, müssen die Systeme auf Clustern realisiert werden, die sich bei Bedarf möglichst einfach erweitern lassen können. Daher spielt die Skalierbarkeit bei NoSQL eine zentrale Rolle. (Gull 2012: S. 18) Um nun den steigenden Anforderungen an eine Datenbank gerecht zu werden, ist ein flexibles Aufrüsten des ganzen Systems nötig. Relationale Datenbanken wurden für den Einsatz auf einem zentralisierten (shared everything) Ein-Server-System entwickelt und optimiert und eignen sich durch ihre strengen ACID-Konsistenzeigenschaften bei Transaktionen nur bedingt für eine horizontale Skalierung, da das Verteilen der Datenbank nur noch mehr Umstände bei der Integritätswahrung bedeuten würde. Daher bleibt hier nur ein kostspieliges Aufrüsten der Hardware der bestehenden Server. NoSQL-Systeme wurden dagegen von Anfang an als verteilte Datenbanksysteme und für horizontale Skalierung konzipiert. Sie eignen sich daher hervorragend für eine Lastenverteilung auf ein beliebig großes Cluster, das auch bei Lastenspitzen ohne Probleme flexibel vergrößert werden kann, verfolgen dafür mit „BASE“ aber ein weniger strenges Konsistenzmodell („eventual consistency“) und verzichten zudem auf Transaktionen. Auch Replikationen spielen bei NoSQL-Systemen eine wichtige Rolle. Da die Daten auf verteilten Serverknoten gespeichert liegen, aber trotzdem eine Erreichbarkeit und Funktionalität vorausgesetzt wird, ist es nötig, Duplikate anzufertigen, die im Falle eines Ausfalls einzelner Knoten ohne Datenverluste zum Einsatz kommen können.

Konsistenz

Geschwindigkeit ist im schnelllebigen Web 2.0-Zeitalter entscheidend. Unternehmen können sich lange Reaktionszeiten oder gar einen Ausfall des Systems nicht mehr leisten (sei er technischer Ursache oder wartungsbedingt). Services müssen täglich und rund um die Uhr performant verfügbar und funktionstüchtig sein. Untersuchungen haben gezeigt, dass lange Antwortzeiten von potentiellen (Online-)Nutzern nicht toleriert werden und die Wahrscheinlichkeit erhöhen, dass der Nutzer den Vorgang abbricht und eventuell andere Anbieter bevorzugt (Borland 2013). Das Credo der neuen NoSQL-Generation ist daher Geschwindigkeit und Verfügbarkeit vor Konsistenz. Das steht nun aber im Gegenteil zu den traditionellen relationalen Datenbanken, die für einen konsistenten Zustand nach dem ACID-Prinzip mit Sperren arbeiten und so Verzögerungen in Kauf nehmen. NoSQL-Systeme richten sich nach den BASE-Richtlinien, die weniger Wert auf unbedingte Konsistenz legen, sondern davon ausgehen, dass sich einige Knoten für kurze Zeit inkonsistent sind und „eventuell“ den Konsistenten Zustand erreichen. Das System ist also *„eventually consitent“* (optimistisch gesehen „irgendwann konsistent“), wobei Konsistenz eher eine Art Prozess ist. Für den Umgang mit kritischen Daten, etwa im Finanzbereich, sind Relationale Datenbanksysteme, die mit Transaktionen nach dem

strengen ACID funktionieren, diesen NoSQL-Systemen vorzuziehen, da es hier darauf ankommt, dass z.B. ein Geldbetrag auch tatsächlich abgebucht wurde und nicht etwa nichtvorhandenes Geld ausgegeben wurde (weil der Kontostand eventuell nicht aktualisiert wurde).

Open Source

Dies ist kein wirkliches Merkmal von NoSQL-Systemen. Die Vorreiter im Bereich der NoSQL-Technologien Google (mit BigTable) oder Amazon (mit Dynamo) haben ihre Entwicklungen intern betrieben und Papers zu ihren Arbeiten veröffentlicht und stellen ihre Produkte nicht Open Source zur Verfügung. Jedoch ist auffällig, dass der größte Teil der NoSQL-Community seine Projekte als quelloffene Produkte anbietet. Das bietet für Unternehmen einerseits einen interessanten Anreiz, sich mit NoSQL auseinanderzusetzen und bietet sich andererseits auch als kostengünstige Alternative zu den bekannten Datenbankanbietern an. Zudem ist es möglich Open-Source-Projekte auf eigene, individuelle Belange anzupassen oder zu optimieren.

Unterteilung

Trotzdem NoSQL-Datenbanken viele Gemeinsamkeiten haben, sind nicht alle gleich, verfolgen unterschiedliche Ziele und wurden für unterschiedliche Einsatzszenarien entwickelt und optimiert. Mittlerweile unterscheidet man zwischen Core-NoSQL-Systemen, die sich aus den Bedürfnissen des Web 2.0-Zeitalters heraus entwickelt haben und Soft-NoSQL-Systemen, die ihren Ursprung überwiegend nicht in den Web 2.0-Anforderungen haben, aber dennoch nicht-relational sind (Edlich o. J.).

Zu den wichtigsten Core-NoSQL-Datenbankmodellen zählen:

- Key-Value-Datenbanken
- Dokumentenorientierte Datenbanken
- Spaltenorientierte Datenbanken
- Graphdatenbanken

Wobei die ersten drei Modelle die Verwaltung der Daten ähnlich handhaben. Daten die man gemeinsam betrachten möchte, werden auch zusammen als ein Aggregat behandelt, geladen und gespeichert. Aggregate entsprechen in diesen Modellen entweder den Values, den Dokumenten oder eben den Spalten-Familien. Man kann sie als **aggregatorientierte Datenbanken** zusammenfassen. Dabei werden diese Aggregate bei einer verteilten Architektur auch als Ganzes auf einem Knoten abgelegt, ohne dass ihre Komponenten im Cluster verstreut liegen und erst zusammengesucht, bzw. aufgeteilt werden müssten. Ein Nachteil ergibt sich jedoch dann, wenn einzelne Attribute der Aggregate betrachtet werden sollen. Dafür müssen sämtliche Aggregate durchsucht werden. (Sadalage/Fowler 2012: S. 19f)

Zu den Soft-NoSQL-Vertretern zählen u.a.:

- Objektdatenbanken
- Grid- und Cloud-Datenbanken
- XML-Datenbanken
- Andere nicht-relationale Datenbanken