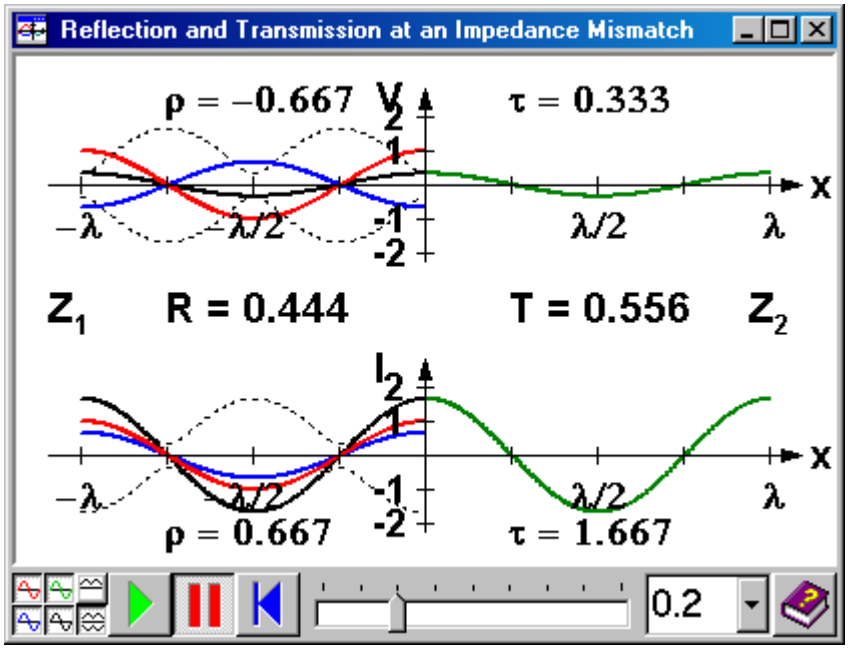
Warum mir der NoSQL-Hype auf den Keks geht

Die Bezeichnung NoSQL kommt immer häufiger in Buchtiteln und Vorträgen vor. Martin Fowler hält darüber Vorträge und hat zusammen mit Pramod Sadalge ein Buch „NoSQL Distilled“ (Addison-Wesley Longman, Amsterdam, 2012) geschrieben. Das Buch ist durchaus instruktiv und gut geschrieben. Sein Problem besteht darin, dass NoSQL als Kampfbegriff gegen SQL positioniert und der falsche Eindruck erweckt wird, dass hier eine ernstzunehmende Alternative zum relationalen Datenmodell entsteht.

Impedanz-Fehlanpassung

Fowler predigt, eines der beiden Hauptprobleme von SQL-Datenbanken sei die „Impedanz-Fehlanpassung“ („impedance mismatch“) zwischen relationaler Struktur der Daten in der Datenbank und Objektstruktur der Daten in Anwendungen. Der Begriff „Impedanz-Fehlanpassung“ kommt aus der Elektrotechnik, wo der Eingangswiderstand gleich dem Ausgangswiderstand sein sollte, wenn man Reflexion und Signalverschlechterung vermeiden will (http://de.wikipedia.org/wiki/Fehlanpassung).



Als unscharfe Bezeichnung für „*das* Problem mit relationalen Datenbanken“ wurde dieser Begriff in den Neunzigerjahren eingeführt, als man noch daran glaubte, dass relationale Datenbanken von objekt-orientierten Datenbanken abgelöst werden müssten. Ganz grob bedeutet es im Kontext der Datenbankprogrammierung die Mühe des Programmierers bei der Überwindung der Differenz zwischen hierarchisch ineinander geschachtelten Objekten und über mehrere Tabellen verteilten Datensätzen, die keine Zugriffsrichtung favorisieren.
Die Unschärfe dieses Begriffs für Programmiererversteher führt dazu, dass jeder Programmierer seine eigenen Schwierigkeiten mit relationalen Datenbanken darauf projizieren kann. Die Häufigkeit mit welcher der Begriff der Impedanz-Fehlanpassung in der ersten, vor allem propagandistischen, Hälfte des Buchs vorkommt, dokumentiert die Hilflosigkeit der Autoren, das Problem zu konkretisieren.
Die Anrufung der Impedanz-Fehlanpassung als Hauptproblem der relationalen Datenbanken krankt an mehreren Fronten: Zum einen sollte man eine konzeptionelle Strukturierung wohl nicht an der Bequemlichkeit für den Programmierer messen sondern nur an ihrer Adäquatheit bezüglich der Anforderungen. Denn die überflüssige Unbequemlichkeit einer Transformation kann in Programmier-

Frameworks ein für alle Mal überwunden werden und verschwindet für den Programmierer. Was übrig bleibt, ist diejenige Unbequemlichkeit, die nicht überflüssig ist. Denn gemäss Einstein soll man zwar alles so einfach wie möglich machen, aber nicht einfacher. Die Propagandisten von NoSQL lügen uns vor, man könne die Probleme rund um die Datenbankmanipulationen einfacher machen als sie tatsächlich sind. Sie schlagen nämlich Lösungen für einen deutlich kleineren Anwendungsbereich vor als er von den relationalen Datenbanken abgedeckt wird. Das sieht man schon daraus, dass man zwar jede Objekthierarchie in eine relationale Struktur verwandeln kann, nicht aber umgekehrt.

Anforderungen an eine relationale Datenbank:

1. Die Daten sollen viele Generationen von Hardware, Programmen und Programmiersprachen überleben.
2. Verschiedenste Programme in verschiedenen Programmiersprachen sollen auf verschiedene Ansichten desselben Datenstamms zugreifen können.
3. Es gibt keine ausgezeichnete Hierarchie des Zugriffs, sondern alle logischen Abfragen sollen gleichermassen möglich sein.

Auf der Basis der Anforderungen an eine relationale Datenbank wurde 1970 das Konzept der relationalen Datenbanken von Edgar F. Codd entwickelt und 1980 die Urform von SQL von IBM implementiert. Obwohl schon SQL einige irritierende Abweichungen vom ursprünglich klaren relationalen Konzept enthält, kam es dem Ideal genügend nahe, dass in den letzten dreissig Jahren praktisch jede Datenbank als SQL-Datenbank konzipiert wird. Sogenannte „objekt-orientierte Datenbanken“ und die meisten „objekt-relationalen Datenbanken“ mit hierarchischen XML-Strukturen in einzelnen Feldern können die obigen Anforderungen nicht erfüllen. Die klassischen objekt-orientierten Datenbanken der Neunzigerjahre sind dadurch charakterisiert, dass in ihren Objekten Code und Daten nicht getrennt vorliegen. Damit scheitern sie schon an der ersten Anforderung. Die objekt-relationalen Datenbanken können die erste Anforderung erfüllen und die zweite teilweise. Spätestens an der dritten scheitern sie aber ebenfalls kläglich. Martin Fowler stellt freimütig fest, dass er früher ein missionarischer Verfechter der objekt-orientierten Datenbanken wir. Nach deren Scheitern an der Realität hat er sich nun den NoSQL-Datenbanken zugewandt, die er mit den gleichen falschen Argumenten propagiert wie damals die objekt-orientierten Datenbanken.

Cluster-Problematik

Die Autoren des Buchs übernehmen die traditionelle, saubere begriffliche Unterscheidung von Datenmodellen und physischen Speicherstrategien. Sie folgen C. J. Date, der anerkannten Autorität auf dem Gebiet der relationalen Datenbanken, in der Unterscheidung eines speziellen Datenmodells einer konkreten Anwendung und dem abstrakten Metamodell der allgemeinen Datenstrukturierung für viele Anwendungen. Letztere Bedeutung benutzen wir, wenn wir vom relationalen Modell reden. Und der Begriff „NoSQL“ wird spezifisch als Kampfbegriff gegen Nutzung des relationalen Modells positioniert.

Das Datenmodell hat nichts mit dem Speichermodell zu tun. Und damit könnten wir das Buch schon fast zuklappen, weil damit die Frage „grosser Daten“ in Hardware-Clustern vom Tisch ist, welche das zweite Hauptargument gegen das relationale Datenmodell ist. Denn die Tatsache, dass viele kommerziell erhältliche SQL-Datenbanksysteme mit über viele Maschinen verteilten grossen Datenmengen nicht sonderlich gut umgehen können, spricht nicht gegen das Modell, sondern gegen die Wahl des implementierten Speichermodells.

Der zweite Teil des Buchs, der vor allem konkrete Implementationen von Datenspeicherungen vorstellt, welche nach der propagierten Begrifflichkeit als NoSQL-Datenbanken bezeichnet werden können, enthält viele brauchbare Ansätze für Teillösungen des Problems grosser verteilter Daten-

mengen. Diese könnten hervorragend zur Implementation von Indizes relationaler Datenbanken herangezogen werden und beweisen nichts gegen das Konzept relationaler Datenbanken.

Was bedeutet „NoSQL“?

Die Autoren gestehen, dass „NoSQL“ ein nur in der Opposition zu relationalen Datenbanken klar abgegrenzter Begriff ist.

Folgende Eigenschaften sollen NoSQL-Datenbanken charakterisieren:

a) Sie benützen kein SQL, sind keine relationalen Datenbanken.
b) Sie wurden im 21. Jahrhundert entwickelt und sind Open-Source Projekte.
c) Die meisten sind auf grosse Hardware-Cluster ausgerichtet.
d) Sie operieren ohne Datenbank-Schema – d.h. man kann ein neues Feld einfügen, ohne erst die Datenstruktur aller anderen Datensätze um ein solches zu erweitern.

Punkt a) überrascht angesichts der Bezeichnung „NoSQL“ nicht. Punkt b) ist schlicht debil. Die Tatsache, dass heute viele Programmentwicklungen als Open-Source-Projekte und alle im 21. Jahrhundert durchgeführt werden und man nicht „in“ ist, wenn man dies nicht tut, kann in der Diskussion um das relationale Konzept keinerlei kognitive Würde beanspruchen. Mit dem 21. Jahrhundert wollen sich die Autoren abgrenzen von schrecklichen alten vor-relationalen Datenbanken, die Codd überhaupt erst zu seiner Konzeption relationaler Datenbanken veranlassten. Leider bleiben sie die Antwort auf die Frage schuldig, wie man das vor-relationale Datenchaos im 21. Jahrhundert verhindert. Punkt c) ist naheliegend, wenn besonders das Problem grosser Datenmengen zur Entwicklung treibt. Es ist aber nicht einzusehen, warum nicht ebensogut relationale Datenbanken für grosse Hardware-Cluster eingerichtet werden sollen. In sogenannten Data Warehouses (ein anderer Hype!) ist genau das der Fall. Punkt d) macht schliesslich einfach die falsche Unterstellung, dass man im relationalen Modell immer dazu gezwungen ist, jede Objektart mit einer starren Struktur von Eigenschaften zu versehen. Die erwähnte Eigenschaft ist nämlich keinesfalls ein Unterscheidungsmerkmal zu SQL-Datenbanken. In diesen kann man problemlos eine 1-n Beziehung der Objekte zu einer variablen Anzahl von Eigenschaften definieren. Sogenannte Schlüssel-Wert-Datenbanken („key-value databases“), d.h. relationale Datenbanken, wo alle Tabellen nur zwei Spalten haben, tun dies bis zum Exzess. Der einzige Nachteil einer solchen „Schemalosigkeit“ besteht darin, dass sich das Programm auch nicht darauf verlassen kann, nur ihm bekannte Eigenschaften vorzufinden. Deshalb wird dieses Designmuster beim Entwurf von relationalen Datenbanken sparsam eingesetzt. Schliesslich versuchen die Autoren, es als Vorteil zu verkaufen, dass nicht so klar ist, was eine NoSQL-Datenbank ist, weil man dann die Bedeutung noch bequem flexibel anpassen kann. Es ist sozusagen eine schemalose Definition.

Und dann kommt der Hammer:

„Man denkt bei NoSQL besser an eine soziale Bewegung als an eine Technologie.“

Eine Institution, die sich auf NoSQL-Datenbanken einlässt, schliesst sich also einer Bewegung an. Da möchte man doch gerne wissen, welche Ziele von dieser Bewegung verfolgt werden und warum man sich ihr anschliessen soll!
Mit dieser Formulierung outet sich Fowler als ein Zelot, der das erste und einzige Gebot der Programmierung verletzt:

Du sollst nicht Religion und Programmierung vermischen!

Schliesslich wird der Begriff „NoSQL“ in seinem Buch zu „not only SQL“ abgeschwächt. Und in dieser Form kann er durchaus ein nützliches Konzept darstellen, wenn man ihn nicht als Kampfbe-

griff gegen relationale Datenbanken positioniert, sondern konkret für grosse Dokumentkollektionen einsetzt, die natürlich in einem „Filesystem“ relational verwaltet werden, aber selber ausserhalb der Datenbank gespeichert sind. Schon im relationalen Design ist es nicht erwünscht, wenn eigenständige Objekte – wie etwa Bilder oder Filme – als grosse binäre Objekte in Datenbankzellen abgelegt werden. Denn ihre Inhalte können so nicht sinnvoll für die Suche oder Integration benutzt werden.

Beispiele für NoSQL-Datenbanken

Für NoSQL-Datenbanken werden vier Datenmodelle vorgestellt:

A) Schlüssel-Wert (Key-Value) Datenbanken

B) Dokument-Datenbanken

C) Spaltenfamilie (Column-Family)

D) Graph-Datenbanken

Die ersten drei dieser Modelle werden als sogenannte aggregat-orientierte Datenbanken zusammengefasst, wo eine vorgegebene, hierarchische Zugriffsrichtung auf eine Kollektion gleichartiger Objekte ausgezeichnet ist. Das letzte Modell entspricht der Strukturierung der Datensammlung als Netz (mathematisch: Graph), wo beliebige Datenobjekte als Punkte mit Verbindungen (Links) verknüpft sind.

Die sogenannten Schlüssel-Wert-Datenbanken sind klassische relationale Datenbanken mit zweispaltigen Tabellen – eine Spalte für den Schlüssel und eine für den Wert. Solche Datenbanken können für gewisse Anwendungen sinnvoll sein. Jede Tabelle einer relationalen Datenbank kann in eine Schlüssel-Wert-Tabelle konvertiert werden. Die Bezeichnung „NoSQL“ passt also hier überhaupt nicht. Vielmehr ist es das altbekannte Prinzip einer variabel langen Property-Liste eines Objekts.

Dokument-Datenbanken sind wie der Name schon sagt eine Kollektion von intern hierarchisch strukturierten, gleichartigen Dokumenten. Diese könnte man als „relationale“ Tabelle mit einer einzigen Spalte verstehen, aber das relationale Modell hat hier nicht viel zu suchen – ausser wenn es dann wieder um die Verwaltung der Dokumente anhand von Metadaten geht.

Die sogenannten Spaltenfamilien ähneln dem Konzept eines hierarchischen XML-Felds in einem nicht-hierarchischen, flachen relationalen Datensatz. Einige Spalten werden „immer zusammen“ gespeichert. (Irgendwie dachte ich, dass das Datenmodell nichts mit dem Speichermodell zu tun hat ...) Einige der SQL-Felder sind einfach nicht flache Werte sondern hierarchisch strukturiert.

Die Graph-Datenbanken basieren auf einem Konzept, wo ein Objekt in einem beliebigen Graph mit anderen Objekten verlinkt werden kann. Aus relationaler Sicht ist die Verlinkung eine Relation und solche Datenbanken schreien förmlich nach relationaler Modellierung. Was bei den Graph-Fans immer wieder überrascht, ist das unbezwingbare Bedürfnis, ausschliesslich *binäre* Relationen zuzulassen. (Siehe auch meine Kritik eines entsprechenden Konzepts 1994 einer Museumsdatenbank in Basel: http://www.enterag.ch/hartwig/gnosarch.html.) Immerhin sind dann später im Beispiel schon ternäre Relationen als Graphen dargestellt, insofern jedes Graph-Link auch noch einen Namen hat.

Schemalose Datenbanken

Die Autoren propagieren als besonderen Vorteil von NoSQL-Datenbanken, dass sie kein Datenbank-Schema besitzen. In SQL-Datenbanken soll es unmöglich sein, unstrukturierte Daten zu speichern. Dabei passiert genau das in relationalen Schlüssel-Wert-Datenbanken. Das Schema wird bei diesem Ansatz „in die Applikation verschoben“ und somit muss diese nie einen Schema-Update durchführen. Es bindet allerdings die Daten derart eng an die Anwendung (bzw. sogar an den spezifischen Anwendungsrelease oder .-build), dass man geradeso gut einen Speicher-Dump der Pro-

grammobjekte abspeichern könnte. Sämtliche oben aufgeführten Anforderungen an relationale Datenbanken können jedenfalls von schemalosen Datenbanken nicht erfüllt werden.
Die Interoperabilität würde dann nicht mehr über SQL sondern über Schnittstellen (beispielhalber über Webschnittstellen von Services) gewährleistet.
Mit diesem Ansatz wird die Datenbank-Anwendung zur Verkörperung des Schemas. Somit können nicht verschiedene Anwendungen auf dieselbe Datenbank zugreifen. Und es ist undenkbar, dass die Daten länger leben als die Software. Nachhaltigkeit ist für schemalose Datenbanken a priori ausgeschlossen.
So wird die Anwendung auch die einzige Garantin der Konsistenz der Datenbank. Die Autoren singen das Loblied der NoSQL-Datenbanken, welche den Programmierer nicht in das Korsett von Konsistenzbedingungen zwängen. Dabei unterschlagen sie, dass man auch relationale Datenbanken inkonsistent gestalten kann, wenn einem der Sinn danach steht. Das führt dann zu den angeblich nur von NoSQL-Datenbanken realisierbaren Performance-Verbesserungen. Offenbar ist aber das Korsett von Konsistenzbedingungen unter Programmierern sehr beliebt.

Datenbanken und Transaktionen

Die Autoren verwenden einige Kapitel auf Fragen der Konsistenz, Parallelität und Isolation von Transaktionen. Diese Themen werden zwar in realen SQL-Datenbanken aus praktischen Gründen realisiert. Sie haben aber absolut nichts mit dem relationalen Datenmodell zu tun. Das ersieht man schon daraus, dass Konsistenz und Isolation von Transaktionen auch bei über Schnittstellen kommunizierenden Anwendungen schwierig zu lösende Probleme darstellen.

Als Möglichkeit zur Implementation optimistischer Parallelität werden Versionsstempel propagiert. Solche Generationenstempel von Datensätzen sind auch im relationalen Modell ein sehr empfehlenswertes Designmuster.

Fazit

Selbstverständlich muss nicht jede externe Speicherung von Programmdaten relational strukturiert sein. In gewissen Fällen ist ein einfacher Speicher-Dump der Programmobjekte angemessener, die heute im Fachjargon als Persistierung dieser Objekte bezeichnet wird. Auch ist es nicht sinnvoll, grosse Kollektionen unabhängiger Dokumente mit Gewalt in einer relationalen Datenbank zu speichern. Nur die Metadaten solcher Dokumente gehören in eine Datenbank, in welcher die Dokumente nur referenziert werden.

Schliesslich ist es sehr häufig, dass konkrete Datenbanken und Datenbankanwendungen nicht optimal für ihren Anwendungsbereich strukturiert in relationalen Datenbanken abgelegt werden oder eine schlechte Performance aufweisen. Wenn man konkrete, ungünstige Implementationsentscheide als Argument verwendet, dass SQL ersetzt werden müsse, dann müsste man nach derselben Logik alle Programmiersprachen abschaffen, weil man mit ihnen fehlerbehaftete oder schlecht strukturierte Programme erzeugen kann. Man müsste also etwa JAVA durch „NoJAVA“ ersetzen, weil JAVA die Entwicklerproduktivität behindert.
Die Autoren haben eine tiefe Abneigung gegen das Konzept der relationalen Datenbanken und wenden jeden rhetorischen Trick von Sektengurus an, um die Leser auf ihre Seite zu ziehen. Dabei benutzen sie unverständliche Insiderbegriffe („Impedanz-Fehlanpassung“), die jeder mit seinem eigenen Inhalt füllen kann und appellieren an weit verbreitete Wunschvorstellungen. (Das Organisieren beliebig komplexer Inhalte sollte einfacher sein.)

Empfehlungen

Es ist immer sinnvoll ist, alternative Speichermodelle für eine konkrete Anwendung zu prüfen. Für grosse Dokument-Datenbanken ist oft eine Abbildung im Filesystem mit einer Verwaltung von Me-

tadaten in einer relationalen Datenbank ein geeigneter Ansatz, der sich auch gut auf Clusters verteilen lässt.

Auf keinen Fall darf das Vorhandensein anderer Speichermodelle dahin ausgelegt werden, das relationale Datenmodell sei durch andere Modelle ersetzbar. Man sollte also sehr vorsichtig sein, wenn man eine Institution von relationalen Datenbanken auf NoSQL-Datenbanken umstellen will. Die Verluste können fatal sein!

Generell sollten Bücher über Informatik-Konzepte Performance-Argumente vermeiden, ausser wo es um mathematische Komplexität von Algorithmen geht. Sie sollten Konzepte nicht mit physischen Realisierungen verwechseln und absolut nie mit der Bequemlichkeit des Programmierers argumentieren, denn dieser könnte durch ein Programm ersetzt werden.

Das relationale Modell ist nach wie vor eine der brauchbarsten Datenabstraktionen, die uns zur Verfügung steht. Ihre Stärke liegt darin, dass es die Ausdrucksmacht der mathematischen Logik für die Daten verfügbar macht.

Datenbank-Interessierte sollten doch lieber C .J. Date als Martin Fowler lesen.

05.09.2013 Hartwig Thomas