Dokumentenorientierte Datenbanken

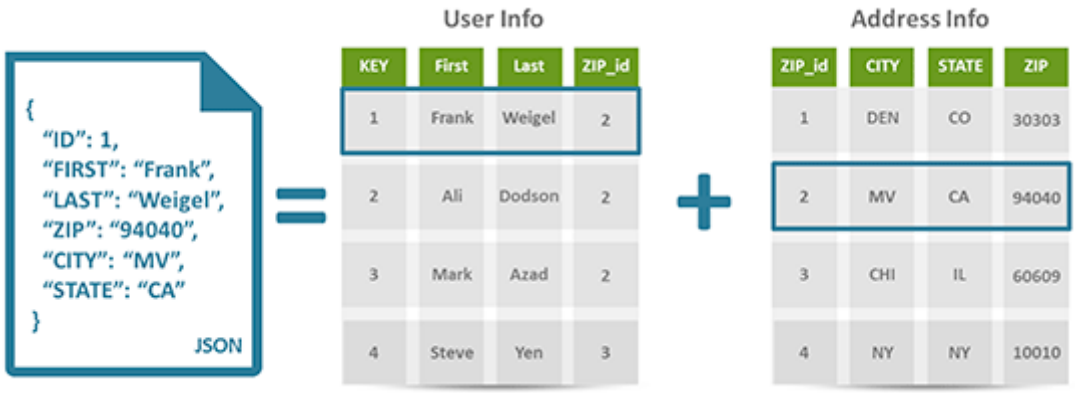
Dokumentenorientierte Datenbanken (oder auch **dokumenten*basierte* Datenbanken**) speichern Daten in Form von Key-Value-Paaren. Wobei zu jedem Schlüssel ein Wert, bzw. **Dokument**. Ein Dokument meint dabei eine Aggregation mehr oder weniger strukturierter Daten ohne Schemavorgabe (Pürner 2013).

Datenmodell

Das Datenmodell ist ähnlich dem der Key-Value-Datenbanken. Im Prinzip ist ihr Modell gleich: Es gibt einen Schlüssel, zu dem ein Wert gehört. Tatsächlich sind die Grenzen fließend und nicht immer klar zu unterscheiden. Oft findet man in unterschiedlichen Quellen unterschiedliche Zuordnungen, wenn es um die Kategorisierung einzelner Vertreter der beiden Datenmodelle geht. Wie auch bei Key-Value-Systemen werden Daten als Aggregate zusammengefasst. Für gewöhnlich handelt es sich dabei um JSON-Objekte (JavaScript Object Notation), da sie ein weitverbreitetes Datenformat für Webanwendungen und Apps sind (denkbar wären aber auch XML-Dokumente). JSON (und auch XML) verlangen keine vordefinierten Felder und können selbst wiederum beliebig verschachtelte Elemente enthalten. Im Gegensatz zu Key-Value-Stores ist die Dokumentenstruktur für die Datenbanken aber transparenter gestaltet (Sadalage/Fowler 2012: S. 20). Das ermöglicht sowohl das Suchen nach Dokumenten, als auch das Suchen nach-, sowie das Updaten von Abschnitten *innerhalb* dieser Dokumente. In gewisser Weise können sie als Erweiterung der Key-Value-Gattung betrachtet werden.

Wie bei den anderen NoSQL-Datenmodellen auch, wird hier kein Schema vorgegeben. Die (JSON-)Dokumente können also nach Belieben gestaltet und ergänzt werden, ohne dass solche Änderungen zuvor dem System bekannt gemacht werden. Das gewährleistet große Flexibilität und bedeutet einen großen Vorteil gegenüber den herkömmlichen relationalen DBMS.

Zwar ist keinerlei Struktur vorgeschrieben und tatsächlich kann jedes Dokument vollkommen anders aufgebaut sein, doch in der Regel wird man nicht wahllos Felder anlegen, sondern dabei ein gewisses, der Anwendung entsprechendes (indirektes) Schema verfolgen, um im Nachhinein auch eine Voraussetzung für sinnvolle Abfragen zu schaffen. So würde man bei einem Shop instinktiv davon ausgehen, dass in einem Dokument für eine Bestellung sowohl ein Feld „Preis“, als auch ein Feld „Anzahl“ mit den entsprechenden Werten und Datentypen dahinter vorhanden ist (und das jene Felder auch genau so heißen und nicht etwa „Kostenpunkt“ und „Menge“). Nichtsdestotrotz ließe sich ein solches Bestellungs-Dokument z.B. ohne weiteres um ein Feld „Anmerkungen“ erweitern. Sollte es zu einer Abfrage kommen, bei der dieses Feld eine Rolle spielt, würden auch nur die Dokumente berücksichtigt, die auch über ein solches Feld verfügen. Bei einer relationalen Datenbank hätte man diese Flexibilität nicht. Dieses Feld müsste auf jeden Fall für alle Bestellung existieren.



(Grafik-Quelle: Couchbase o. J.)

Vorteile

Da Zugriffe auf den Inhalt der Dokumente möglich sind, ist ein Vorteil gegenüber den Key-Value-Stores, dass, je nach Art und Strukturierung der Dokumente, hier Abfragen möglich sind, wie sie auch in SQL denkbar wären. Dabei lassen sich Tabellen mit Dokumentensammlungen (**Collections**) -und Reihen mit einzelnen Dokumenten vergleichen. Zudem erlaubt der Verzicht auf Schemarestriktionen große Flexibilität beim Umgang mit großen Datenmengen von unterschiedlicher Struktur, da jedes Dokument einen eigenen Strukturaufbau verfolgen kann. Weiterhin bedeutet die Datenverwaltung durch Dokumente einen Geschwindigkeitsvorteil gegenüber RDBMS, da hier Informationen aggregiert werden, wo sie sonst erst über mehrere Tabellen durch JOINs hätten zusammengefasst werden müssen. Außerdem bietet die aggregierte Speicherform Vorteile bei der horizontalen Skalierung des Systems.

Nachteile

Auch hier ist es nicht ohne Weiteres möglich, Beziehungen zwischen den Dokumenten herzustellen. Es ist zwar möglich, ID-Felder in die Dokumente zu schreiben, jedoch sind darüber keine Direkten JOINs möglich, da zunächst ein Dokument geladen und nach der ID gesucht wird und dann im nächsten Schritt erneut die Datenbank befragt werden muss, um ein Dokument mit der passenden ID zu suchen und zu laden, um dann eine Schnittmenge zu erstellen. Da es sich um ein nichtrelationales Modell handelt, bietet die Datenbank dafür auch keinerlei Abfragemöglichkeiten und müssen daher selbst programmiert werden. Für Anwendungen, bei denen komplexere Beziehungen wichtig sind, eignen sie sich also eher nicht. Während die Schemafreiheit einerseits zwar große Flexibilität bei der Modellierung bietet, bedeutet sie andererseits jedoch Aufwand bei der Abfrageprogrammierung, da zuvor der Applikationscode studiert werden muss, um Einblicke über das indirekte Schema der Datenbank gewinnen zu können (Sadalage/Fowler 2012: S. 29). Zudem müssen Festlegungen zu Dokumentenstrukturen, eventuelle Wertprüfungen, Trigger und dergleichen in der Anwendung programmiert werden und können nicht von der Datenbank selbst übernommen werden.

Verwendung

Da dokumentenorientierte Datenbanken, ähnlich den Key-Value-Datenbanken, ein sehr allgemein gehaltenes Datenmodell haben, sind sie freilich vielseitig einsetzbar. Jedoch gilt zu beachten, dass

nicht alle Datenbanken gleich sind, da unterschiedliche Hersteller wohlmöglich unterschiedliche Schwerpunkte bei der Entwicklung gesetzt haben und somit hat man unter Umständen je nach Produkt einen unterschiedlich mächtigen Funktionsumfang.

Allgemein eignen sich dokumentenbasierte Systeme, dank ihrer oftmals intern implementierten JSON-Notation, besonders gut für Web-Applikationen, da dort die verbreitetsten Datenaustauschformate sowieso XML und JSON sind.

Bekannte Produkte dieser Kategorie u.a.: Lotus Notes, MongoDB, CouchDB.

From:
https://wi-wiki.de/ - **Wirtschaftsinformatik Wiki - Kewee**

Permanent link:
https://wi-wiki.de/doku.php?id=bigdata:dokumentdb

Last update: **2015/10/05 21:19**