NoSQL Schema Design

Andreas Hartmann
adesso AG
Dortmund

Schlüsselworte
NoSQL, Schema-Design, Performance

Einleitung

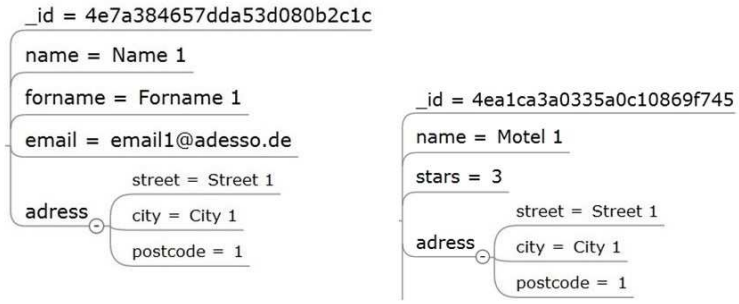
Das Schemadesign ist bei NoSQL Datenbanken ebenso eine wesentlicher Faktor für die Performance und Wartbarkeit wie bei den herkömmlichen Relationalen Datenbanken.
Bei den NoSQL Datenbanken geht man bei der Datenmodellierung von der Fragestellung aus „What Question do I have?“ während im Gegensatz bei Relationalen Datenbanken man die Datenmodellierung unter dem Aspekt „What Answer do I have?“ betrachtet. Im Rahmen dieser Session werden mögliche Ansätze zur Datenmodellierung vorgestellt, die jeweils immer im Kontext des jeweiligen Business Cases und der gewählten NoSQL Datenbankkategorie betrachtet werden.

Schema-Design in der NoSQL Welt

Das Schemadesign ist bei NoSQL Datenbanken ebenso ein wesentlicher Faktor für die Performance und Wartbarkeit wie bei relationalen Datenbanken.
Bei den NoSQL Datenbanken geht man bei der Datenmodellierung von „Welche Fragen müssen beantwortet werden?“ aus, während bei relationalen Datenbanken „Welche Daten sind vorhanden?“ betrachtet wird [HSB]. Relationales Schema-Design strebt durch Normalisierung eine möglichst redundanzfreie und flexible Modellierung der Daten an. Anfragen können sich durch Joins die benötigten Informationen zusammensuchen. Die Anfragen können durch Indizes optimiert werden.
Bei den meisten NoSQL-Datenbanken gibt es keine Joins - das gilt vor allem für Key/Value, Large Column und dokumentenorientierte Datenbanken. Daher müssen die Daten so zusammen abgelegt werden, dass die geplanten Anfragen beantwortet werden können. Dafür bekommt der Entwickler eine wesentlich bessere Skalierbarkeit bei den NoSQL-Datenbanken.
Außerdem sind NoSQL-Datenbanken flexibler: Die NoSQL Datenbanken sind schemalos und können daher praktisch beliebige Daten speichern und verarbeiten. Die Anwendung, die eine NoSQL Datenbank verwenden, haben natürlich immer noch eine festgelegte Datenstruktur, mit der sie arbeiten. Der limitierende Faktor bei der Flexibilität ist also nicht mehr die Datenbank, sondern die Anwendung.
Bei der Datenmodellierung haben sich folgende Ansätze als hilfreich erweisen:

Denormalisieren

Denormalisierung ist das bewusste schaffen von Datenredundanzen. Das kann die Abfrage vereinfachen oder optimieren. Es hat aber den Nachteil, dass das zu speichernde Datenvolumen steigt. Außerdem muss die Anwendung die Konsistenz zwischen den verschiedenen redundanten Informationen sicherstellen.
Im folgenden Beispiel wird dieselbe Adresse sowohl bei dem Benutzer als auch beim Hotel abgelegt. Dies führt zwar zur redundanten Ablage derselben Adresse im System, vereinfacht jedoch die Abfrage der jeweiligen Hotel oder Nutzerdaten, da sie jeweils mittels eines Datenbankzugriffs möglich ist.



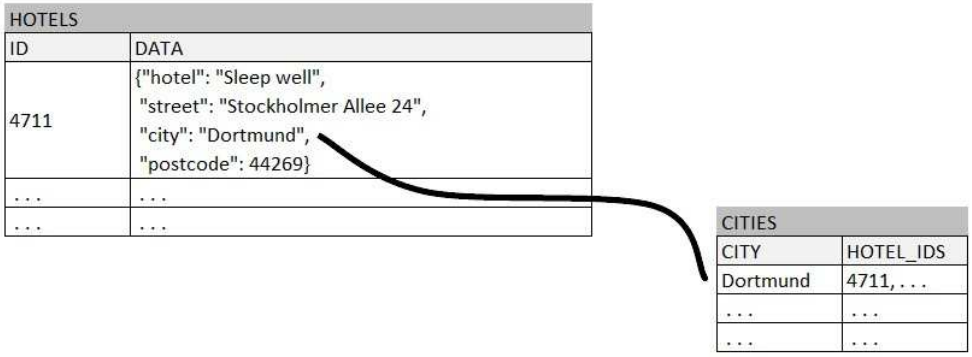
Aggregieren
Bei der Aggregation von Daten werden 1 zu 1 oder 1 zu n Beziehungen zusammengezogen. Das erhöht die Performance bei lesenden Zugriffen, da sofort alle abhängigen Daten ohne Nachladen verfügbar sind. Es führt aber eventuell zu einer höheren Komplexität bei der Verarbeitung der Daten und erhöhten Datentraffic bei der Änderung einzelner Bestandteile.
Im nachfolgenden Beispiel sind die Adressdaten zusammen mit den Personendaten abgelegt. Die führt zu Performancevorteilen beim Laden eines Benutzers, da hierbei die entsprechenden Daten jeweils zusammen abgelegt sind.

Clientseitige Joins
Die meisten NoSQL Datenbanken unterstützen keine Joins. Nichts desto trotz sind diese für bestimmte Anwendungsfälle fachlich unerlässlich. So lassen sich häufig 1 zu 1 oder 1 zu n Beziehungen mit den Ansätzen Denormalisieren oder Aggregieren abdecken. Bei n zu m Beziehungen ist dies jedoch meist inhaltlich nicht vertretbar. Hier kann auf Ansatz des Clienseitigen Joins der Daten zurückgegriffen werden. Das hat natürlich in Bezug auf die übertragene Datenmenge und damit die Laufzeit signifikante Auswirkungen, da gegebenfalls erheblich Datenmengen verarbeitet werden müssen.

Atomare Daten
In den meisten Datenbanken sind atomare Änderungen auf einen Datensatz beschränkt. Oft ist es möglich, mit einer einzigen Aktion einen Datensatz zu suchen und dann auch gleich zu ändern. Atomare Änderungen über mehrere Datensätze werden von den meisten NoSQL Datenbanken bewusst nicht unterstützt, um das Handling von verteilen Locks und Deadlocks zu vermeiden.
Eine wesentliche Herausforderung beim Schema-Design ist also, die Daten so zu aggregieren, dass fachliche Datensätze, die zusammen atomar geändert werden müssen, im Schema auch gemeinsam abgelegt werden.

Indextabellen
Selbstverwaltete Indextabellen können zur Beschleunigung von Abfragen eingesetzt werden, wenn dies nicht die eingesetzte NoSQL Datenbank von sich aus unterstützt. Hierbei ist zu berücksichtigen, dass die Indextabellen entsprechend zu aktualisieren sind. Dies kann in Batchläufen oder während

eines Inserts/Update/Deletes geschehen. Das hat allerdings negative Auswirkungen auf die Performace bei schreibenden Operationen.
In der nachfolgenden Abbildung wird der Index für die Eigenschaft Stadt aufgebaut, um auf alle Hotels in einer Stadt zuzugreifen. Zur Suche nach allen Hotels in Dortmund wird zunächst bei CITIES der Eintrag mit CITY=Dortmund gesucht. Dieser enthält die IDs aller Hotels in Dortmund. Mittels der IDs der Hotels erfolgt dann ein gezielter Zugriff auf die jeweiligen Hotels.



HOTELS	
ID	DATA
4711	{"hotel": "Sleep well", "street": "Stockholmer Allee 24", "city": "Dortmund", "postcode": 44269}
. . .	. . .
. . .	. . .

CITIES	
CITY	HOTEL_IDS
Dortmund	4711, . . .
. . .	. . .
. . .	. . .

Composite Key Indexing
Composite Keys, die sich aus den entsprechenden fachlichen Werten zusammensetzten, können bei einigen NoSQL Datenbank als mehrdimensionale Indizes verwendet werden. Anfragen, die eine solche Zusammensetzung von Schlüsseln nutzen, können so beschleunigt werden. Zur Gruppierung von Daten können Composite Keys ebenfalls eingesetzt werden.
Das Beispiel in der nachfolgenden Abbildug zeigt, wie zur Abfrage von Hotels in einer bestimmten Stadt und einer bestimmten Anzahl von Sternen ein Composite Key, basierend auf CITY:STARS:ID, eingesetzt werden kann. So können z.B. mittels “Dortmund:2” alle 2 Sterne Hotels in Dortmund ermittelt werden, ohne dass jeweils der eigentliche Datensatz eines Hotels zu analysieren ist. Die Anfrage kann durch die Daten aus dem Index bereits abgearbeitet werden.

HOTELS	
CITY:STARS:ID	DATA
Dortmund:4:4711	{"id": "4711", "hotel": "Sleep well", "stars": 4, "street": " . . . ", "city": "Dortmund", "postcode": . . . }
Dortmund:2:4712	{"id": "4712", "hotel": "Sleep cheap", "stars": 2, "street": " . . . ", "city": "Dortmund", "postcode": . . . }
Dortmund:2:4713	{"id": "4713", "hotel": "Sleep relay cheap", "stars": 2 "street": " . . . ", "city": "Dortmund", "postcode": . . . }
. . .	. . .
. . .	. . .

Die angeführten Möglichkeiten der Datenmodellierung sind ebenso, wie bei RDBMS, ausschlaggebend für die Performance und die fachliche Flexibilität der Anwendung.

Kontaktadresse:
Andreas Hartmann
adesso AG
Stockholmer Allee 24
D-44269 Dortmund

Telefon:	+49 (0) 49 178 2808012
Fax:	+49 (0) 49 231 930-9331
E-Mail	hartmann@adesso.de
Internet:	www.adesso.de