Data modeling for column stores

Hausarbeit

Patrick Lieske

Inhaltsverzeichnis

1 Einleitung

Diese Arbeit beschreibt Modellierungstechniken, die bei spaltenorientierten Datenbanken eingesetzt werden. Spaltenorientierte Datenbanken gehören zur Gruppe der NoSQL Datenbanken. Was NoSQL bedeutet und welche Eigenschaften dieser Gruppe von Datenbanken zugeordnet werden, ist Gegenstand des zweiten Kapitels. Hier werden wichtige Begriffe und Eigenschaften, die zum weiteren Verständnis des Themas essentiell sind, eingeführt.

In Kapitel 3 wird das Datenmodell spaltenorientierter Datenbanken vorgestellt. Im weiteren Verlauf werden die Unterschiede bei der Modellierung von relationalen Systemen und spaltenorientierten Systemen herausgearbeitet. Die Erkenntnisse aus Kapitel 3 werden in Kapitel 4 zur Beschreibung von Beispielen genutzt. Die Arbeit endet mit einem Fazit.

2 Grundlagen spaltenorientierter Datenbanken - NoSQL

Für das Verständnis der Thematik spaltenorientierter Datenbanken, ist die Einführung wichtiger Begriffe notwendig. Diese Begriffe und Vorgehensweisen werden in diesem Kapitel eingeführt. Spaltenorientierte Datenbanken gehören zur Gruppe der NoSQL Datenbanken. Der Begriff NoSQL ist ein Schlagwort (Buzzword), dass oft mit Not only SQL (nicht nur SQL) übersetzt wird. Populär wurden NoSQL Datenbanken Anfang der 2000er Jahre mit dem Siegeszug des Web 2.0 Zeitalters. Das Ziel der Entwicklung dieser Datenbanken war die Verarbeitung von sehr großen Datenmengen im Terabyte -und Petabyte-Bereich. NoSQL Datenbanken weisen nach [EFH+11] folgende Eigenschaften auf:

- Das zugrundelegende Datenmodell ist nicht relational.
- Die Systeme sind von Anbeginn an auf eine verteilte und horizontale Skalierbarkeit ausgerichtet.
- Das NoSQL System ist Open Source.
- Das System ist schemafrei oder hat nur schwächere Schemarestriktionen.
- Aufgrund der verteilen Architektur unterstützt das System eine einfache Datenreplikation.
- Das System bietet eine einfache API.
- Dem System liegt meistens auch ein anderes Konsistenzmodell zugrunde: Eventually Consistenz und Base, aber nicht ACID.

Eine weitere wichtige Eigenschaft dieser Gruppe von Datenbanken betrifft die horizontale Skalierbarkeit. Hier sollte es möglich sein Cluster und Datacenter mit Low-Cost Hardware aufzubauen. Bei relationalen Datenbanken ist der Aufbau eines hochperformanten Clusters meist nur mit teurer Spezialhardware möglich.

NoSQL Systeme werden wie folgt kategorisiert.

- Key-Value Stores
- Document Stores
- Column Stores (Wide Column Stores/Column Families)
- Graphdatenbanken

Key-Value Stores, Document Stores und Column Stores verwenden die Eigenschaften Eventually Consistent (BASE), verteilte/horizontale Skalierung und Datenreplikation, aus obiger Liste. Da die genannten Eigenschaften für das Verständnis von NoSQL Datenbanken, insbesondere für das Thema Column Stores dieser Arbeit, essentiell sind, wird in den nächsten Abschnitten auf diese Thematik näher eingegangen.

2.1 CAP-Theorem und BASE-Modell

Vor dem Web 2.0 Zeitalter wurden die Architekturen von Datenbanksystemen nach dem ACID-Prinzip (Atomicity, Consistency, Integrity, Durability) entwickelt. Dies bedeutete, dass die Konsistenz der Daten das zentrale Thema waren. Inkonsistenzen durften insbesondere bei parallelen Zugriffen nicht auftreten. Mit den damals zu bewältigen Datenmengen war das kein Problem. Dies änderte sich allerdings mit dem Internet-Boom. Die Architekturen relationaler Datenbanksysteme lassen keine horizontale Skalierung, schnelle Replikation und ab bestimmten Datenmengen keine schnellen Reaktionszeiten (Zugriffszeiten) zu.

Eric Brewer stellte im Jahr 2000 in seinem Vortrag „Principles of Distributed Computing“ sein CAP-Theorem vor und bewies damit, dass die vollständige Vereinbarkeit von Konsistenz (Consistency), Verfügbarkeit (Availability) und Ausfalltoleranz (Partition Tolerance) nicht möglich ist. Die Kernaussage des CAP-Theorems besagt, dass nur zwei dieser drei Größen erreicht werden können. In [EFH+11] werden die drei Größen genauer beschrieben. So steht Konsinstenz (Consistency) im CAP-Theorem dafür, dass die verteilte Datenbank nach Abschluss einer Transaktion einen konsistenten Zustand erreicht hat, auch wenn der Datenbestand über Knoten verteilt ist. Eine folgende Leseoperation -über welchen Knoten auch immer- muss den aktuellen, konsistenten Wert zurückliefern. Das bedeutet aber auch, dass der aktualisierte Wert über alle Knoten repliziert werden muss. Diese Tatsache ist in sehr großen Clustern nicht mit schnellen Reaktionszeiten vereinbar.

Verfügbarkeit (Availability) setzt aber genau diese schnellen bzw. akzeptablen Reaktionszeiten voraus. Mit Ausfalltoleranz ist gemeint, dass der Ausfall eines Knotens oder einer Kommunikationsverbindung nicht zum Ausfall des gesamten Systems führt, sondern Anfragen weiter verarbeitet werden.

Da nur zwei der drei Größen nach dem CAP-Theorem bedient werden können, wurden die Architekturen von NoSQL Datenbanken auf Kosten der Konsistenz entwickelt. Gerade bei Web-Unternehmen wie Facebook, Google, Amazon oder Netflix entscheiden Verfügbarkeit und Ausfalltoleranz über Gewinn und Verlust.

Als Konsequenz aus dem CAP-Theorem wurde ein neues Modell entwickelt, dass so genannte BASE-Modell (Basically, Available, Soft State, Eventually Consistent). Hier wird Konsistenz der Verfügbarkeit untergeordnet. Die Systeme nach BASE erreichen auch „irgendwann“ einen konsistenten Status. Es ist aber auch möglich das veraltete, inkonsistente Daten ausgeliefert werden. In [DHJ+07] wird die Verwendung von

„Data Versioning“ bzw. „Version Evolution“ beschrieben, die das Problem inkonsistenter Daten abschwächt.

2.2 Consistend Hashing

Consistent Hashing wird verwendet, um zu einem Objekt in einem verteilten System möglichst schnell einen Speicherort, bei NoSQL-Datenbanken einen Server, zu finden. Dafür werden Hash- bzw. Streuwertfunktionen verwendet. Nach [EFH+11] definiert eine Streuwertfunktion eine Vorschrift, die einen Wert x aus einer großen Quellmenge auf einen Hashwert v = h(x) aus einer deutlich kleineren Wertemenge abbildet. Hashing-Verfahren haben den Vorteil, dass sie in konstanter Zeit O(1) ein Ergebnis liefern. In Abbildung 1 wird das Consistend Hashing Verfahren schematisch dargestellt.

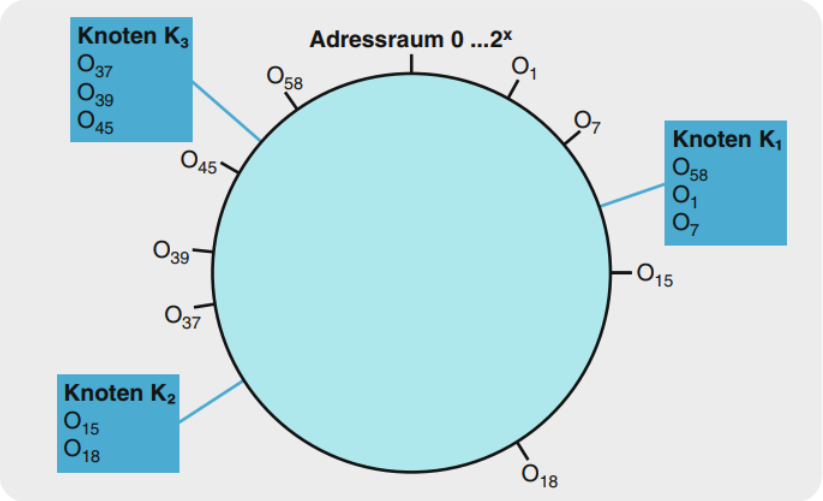

Abbildung 1: Consistent Hashing, Abbildung aus [MK16]

Ein weiteres Ziel beim Consistend Hashing ist die Gleichverteilung der zu speichernden Objekte innerhalb des in Abbildung 1 dargestellten Ringes. Der Ring wird als Adressraum von 0 bis 2^x Schlüsselwerten zusammengefasst. Danach wird eine Hashfunktion gewählt, die die Netzwerkadressen der Knoten auf Speicheradressen abbildet und im Ring einträgt. Die Schlüssel der zu speichernden Objekte werden mithilfe der Hashfunktion zu Adressen transformiert und auf dem Ring eingetragen.

So werden die Objekte O1, O7 und O58 aus Abbildung 1 dem Knoten K1 zugeordnet. Dabei erfolgt die Zuweisung im Uhrzeigersinn. Demnach werden die Objekte O15 und O18 auf Knoten K2 gespeichert. Die Stärke von Consistent Hashing zeigt sich beim Entfernen und Hinzufügen von Knoten, denn die Änderungen haben nur Auswirkungen auf die Objekte in unmittelbarer Nähe zu den veränderten Knoten im Ring. Abbildung 2 zeigt ein solches Szenario.

Wie in Abbildung 2 zu sehen, wird Knoten K2 entfernt und Knoten K4 hinzugefügt. Diese Änderung wirkt sich nur auf die Objekte O15 und O18 aus. Der Speicherort für O15 ist jetzt Knoten 4, da er im Uhrzeigersinn der nächstgelegende Knoten ist. Objekt O18 wird auf Knoten K3 aus dem selben Grund gespeichert.

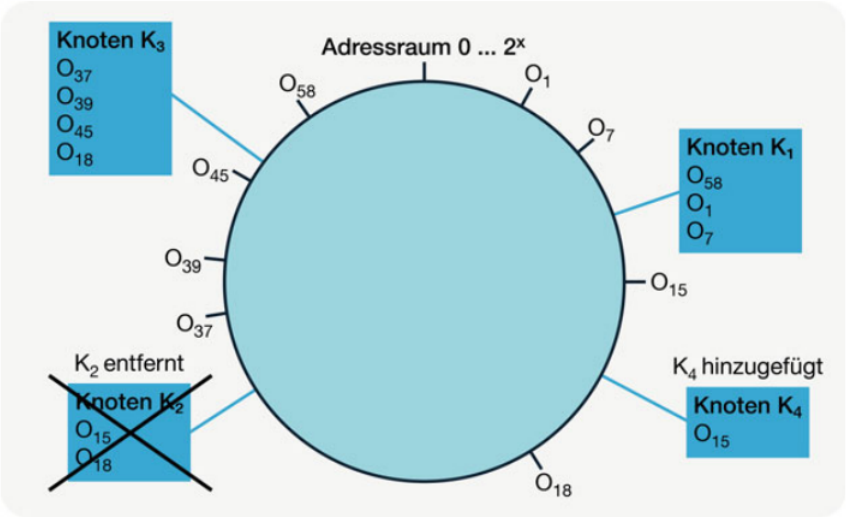

Abbildung 2: Consistent Hashing, Abbildung aus [MK16]

Für viele große Internet Unternehmen sind Verfügbarkeit und schnelle Reaktionszeiten zu priorisieren. Mit dem Consistent Hashing Verfahren haben sie ein Werkzeug zur Hand, um schnell auf Stoßzeiten zu reagieren. Werden viele Anfragen erwartet, so wird die Zahl der Knoten, also der Server im Ring, erhöht. In Zeiten geringer Anfragen kann die Anzahl der Knoten verringert werden. So kann flexibel reagiert und Kosten gespart werden.

Ein wichtiger Unterschied zu relationalen Datenbanksystemen ist die Linearität der Performance bei Hinzunahme von Knoten. In relationalen Datenbanksystemen wird die Performance an einem bestimmten Punkt, durch Hinzunahme von Servern, nicht mehr gesteigert. Dieser Punkt ist relativ schnell erreicht. NoSQL Datenbanken nach BASE können bis ins unendliche skaliert werden.

2.3 Replikation

Im Abschnitt 2.2 -Consistend Hashing- wurde beschrieben wie die Daten möglichst gleich-verteilt im Ring gespeichert werden. Der Ausfall von Knoten im Ring würde allerdings bedeuten, dass die Daten verloren wären. Es ist demnach wichtig die Daten zu replizieren.

Abbildung 3 zeigt die Replikationsstrategie bei Cassandra, einer populären NoSQL Column Store Datenbank. Die Anzahl der Replikate richtet sich dabei nach dem Replikationsfaktor N, der bei Cassandra vom Nutzer bestimmt wird. In Abbildung 3 wird ein Replikationsfaktor N=3 verwendet. Der Datensatz A ist damit auf drei Knoten verteilt und die Robustheit des Systems ist sichergestellt. Die Replikate werden dabei, wie in Abbildung 3 zu sehen, auf die folgenden Knoten (hier Knoten 2 und 3) im Uhrzeigersinn geschrieben. Eine noch größere Robustheit des Gesamtsystems kann dadurch erreicht werden, dass die Replikate in verschiedenen Datacentern gespeichert werden.

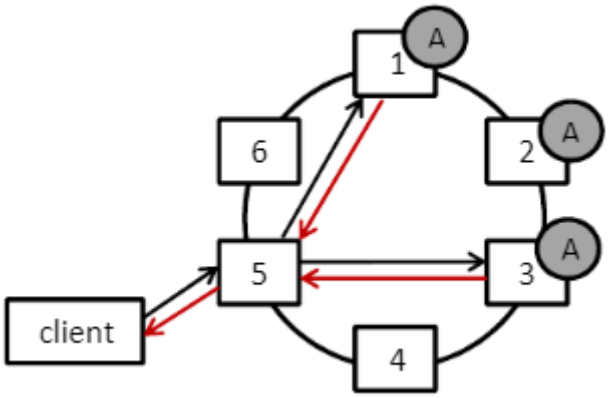

Abbildung 3: Replikationsstrategie, Abbildung aus [OP14]

In Abbildung 3 wird ferner gezeigt, wie ein Request eines Clients bei Cassandra abläuft. In der Regel werden die Anfragen der Clients über einen Load-Balancer verteilt. Nach [OP14] werden folgende Schritte ausgeführt.

1. Request, mit Schlüssel für Datensatz A, wird von Knoten 5 empfangen. Knoten 5 wird zum Koordinator-Knoten.
2. Der Datensatz befindet sich auf den Knoten 1, 2 und 3. Der Wert des Konsistenzlevels beträgt 2. Folglich wird vom Koordinatorknoten ein Read-Request an zwei der drei Knoten (hier 1 und 3) gesendet.
3. Der Koordinatenknoten (Knoten 5) vergleicht die Werte beider Knoten (1 und 3) und liefert den aktuellen aus.
4. Die Aktualisierung der Werte kann mit dem weiter oben genannten Verfahren „Data Versioning“ bzw. „Version Evolution“ erfolgen. Eine Beschreibung dieser Technik findet sich in [DHJ+07].
5. Falls Knoten 1 und 3 unterschiedliche Werte enthielten, ist nach dem Read-Request der Datensatz auf Knoten 1 und 3 in einem konsistenten Zustand.

3 Datenmodellierung

NoSQL Datenbanken werden oft als „schemafrei“ bezeichnet, weil keine Struktur der Daten, wie beispielsweise in der relationalen Welt, definiert werden muss. Trotzdem müssen auch hier Modellierungsentscheidungen getroffen werden. Bevor das Thema Modellierung aufgegriffen wird, muss geklärt werden, welches Datenmodell spaltenorientierte Datenbanken verwenden. Abschnitt 3.1 gibt eine Einführung zu diesem Thema. Ferner wird in Abschnitt 3.2 die Modellierung in der relationalen Welt mit der Modellierung spaltenorientierter Datenbanken verglichen.

3.1 Datenmodell spaltenorientierter Datenbanken

Anhand eines Beispiels soll das Datenmodell von spaltenorientierten Datenbanken erläutert werden. Dabei gibt es einen entscheidenden Unterschied zu zeilenorientierten Datenbanken, wie beispielsweise MySQL. Die Art der Datenspeicherung auf dem persistenten Medium (Festplatte) ist bei beiden Datenmodellen (zeilenorientiert und spaltenorientiert) unterschiedlich. Als Beispiel soll die in Abbildung 4 gezeigte Tabelle dienen. Die Tabelle beschreibt den Verdienst einzelner Personen.

PersonalNr	Nachname	Vorname	Gehalt
1	Schmidt	Josef	40000
2	Müller	Maria	50000
3	Meier	Julia	44000

Abbildung 4: Einfache Tabelle, Abbildung aus [Wik19]

In zeilenorientierten Datenbanken werden alle Tupel einer Tabelle, wie in Abbildung 5 gezeigt, nacheinander abgespeichert. Zeilenorientierte Systeme sind effizient, wenn alle Spalten einer Zeile benötigt werden. Die Daten liegen auf dem Speichermedium hintereinander und können sequentiell gelesen werden. Die gleiche Operation auf spaltenorientierten Systemen ist ineffizient, da viele Sprünge auf dem Speichermedium verursacht werden.

```
1,Schmidt,Josef,40000;2,Müller,Maria,50000;3,Meier,Julia,44000;
```

Abbildung 5: Speicherung von Zeilen, Abbildung aus [Wik19]

In spaltenorientierten Datenbanken werden die Einträge der Spalten, wie in Abbildung 6 zu sehen ist, hintereinander gespeichert. Diese Systeme arbeiten sehr effizient

bei der Bildung von Aggregaten, beispielsweise bei der Berechnung der Summe aller Gehälter. Diese Operation, ausgeführt auf zeilenorientierten Systemen, ist ineffizient, da erst alle Zeilen (mit allen Spalten), also alle Daten, eingelesen werden müssen. Bei großen Datenbeständen kann diese Operation sehr lange dauern.

```
1,2,3;Schmidt,Müller,Meier;Josef,Maria,Julia;40000,50000,44000;
```

Abbildung 6: Speicherung von Spalten, Abbildung aus [Wik19]

Wichtige Vertreter von spaltenorientierten Datenbanken sind BigTable von Google und Cassandra (Apache Foundation). Genauer gesagt zählen diese Datenbanken zur Gruppe der Column Family Stores. In den weiteren Ausführungen zum Datenmodell, wird speziell auf die beiden genannten Vertreter eingegangen. Abbildung 7 zeigt das Datenmodell von BigTable. Auf die Datensätze kann nur über den sogenannten Row-Key zugegriffen werden. Er ist vergleichbar mit dem Primärschlüssel von relationalen Systemen. Eine Abfrage von Werten, wie bei relationalen Systemen, ist nicht möglich. Der Row-Key in Abbildung 7 setzt sich aus der URL (in umgekehrter Reihenfolge) zusammen ("com.cnn.www").

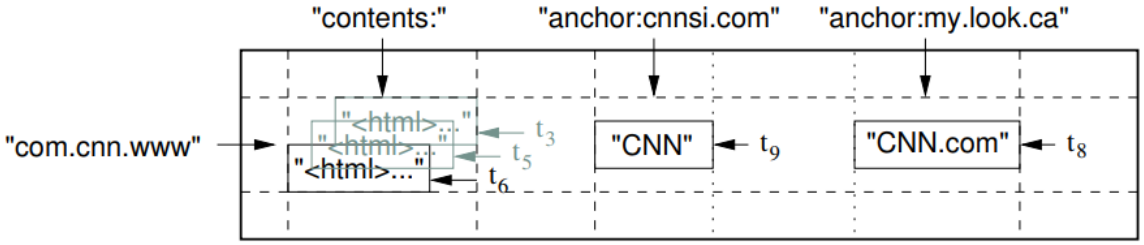

Abbildung 7: Datenmodell Column Store, Abbildung aus [CDG+06]

Column families, in Abbildung 7, bilden „contents:“ und „anchor:“. Unter der column family „contents:“ wird der Inhalt der Seite in drei zeitlich verschiedenen Versionen (t3, t5 und t6) abgespeichert. In der column family „anchor:“ erfolgt eine weitere Unterteilung in column keys. Die Ausdrücke, aus Abbildung 7, „anchor:cnnsi.com“ und „anchor:my.look.ca“ gehören zur column family „anchor:“, wobei die Column Keys aus „my.look.ca“ und „cnnsi.com“ gebildet werden. Die Werte sind „CNN“ und „CNN.com“.Die Reihe aus Abbildung 7 speichert folglich Verlinkungen auf die Seite „www.cnn.com“.

Dieses Beispiel sollte dem Leser bewusst machen, dass riesige Datensätze hinter einer Zeile stecken können. Was auch deutlich wird ist die Tatsache, dass die „Tabelle“ nicht wie bei relationalen Systemen nach unten (vertikal) gefüllt wird, sondern nach rechts (horizontal). Eine Zeile kann bei spaltenorientierten Systemen bis zu 2 Milliarden Spalten enthalten.

Das Beispiel zeigt auch, dass die Spalten dynamisch nach rechts wachsen und pro Zeile unterschiedlich viele Spalten existieren können. In relationalen Systemen ist dies nicht möglich. Hier muss, trotzdem eine Spalte keinen Wert enthält, Speicher allokiert werden. Was bei großen Datenbeständen nicht unerheblich sein kann.

Dem aufmerksamen Leser sollte aufgefallen sein, dass spaltenorientierte Systeme nichts anderes als verschachtelte Hash Maps sind. Auch bei steigenden, riesigen Datenmengen bleibt die Suchkomplexität gleich, da nur über den Row-Key die Zugriffe erfolgen. Relationale Systeme werden bei großen, steigenden Datenmengen immer langsamer.

Folgende Liste beschreibt wichtige Begriffe anhand des Datenmodells der Datenbank Cassandra. Viele Begriffe wurden bereits weiter oben verwendet. Die folgende Liste kann als Zusammenfassung betrachtet werden. Abbildung 8 zeigt die verwendeten Begriffe in einem Beispiel.

1. Der **Keyspace** ist die oberste Ebene der Datenstruktur. In relationalen Systemen gleichzusetzen mit der Datenbank.
2. In einem Keyspace werden **Column Families** angelegt. Column Families besitzen kein vorgegebenes Schema, so dass das Wissen zur Interpretation der Daten auf Applikationsebene verbleibt. Daten innerhalb einer Column Family haben den gleichen Typ, denn es wird davon ausgegangen, dass sie zusammen gelesen werden (siehe [MK16]).
3. Eine **Row** ist eine über Schlüssel eindeutig identifizierbare Einheit von Daten.
4. **Super Columns** sind Columns dessen Werte wieder Columns sind. Sie ermöglichen die Verschachtelung komplexer Datenstrukturen.
5. **Column** ist die kleinste Informationseinheit. Sie besteht aus Schlüssel-Wert Paaren. In Cassandra aus Name, Wert und Zeitstempel.

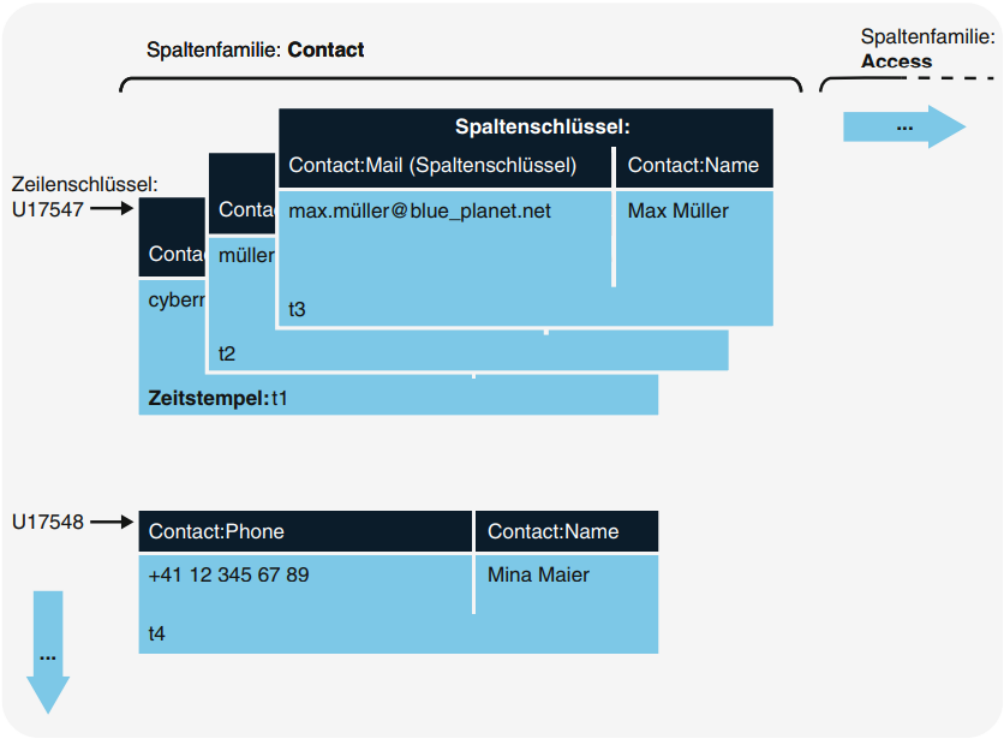

Abbildung 8: Beispiel Datenmodell, Abbildung aus [MK16]

3.2 RDBMS im Vergleich zu spaltenorientierten Datenbanken

Die Grundlage relationaler Datenbanksysteme bildet die mathematisch definierte Relation, die auch relationale Algebra genannt wird. Alle Daten in relationalen Datenbanken sind in Tabellen gespeichert, die miteinander in genau definierten Beziehungen stehen. Tabellen bestehen aus Zeilen, die auch Tupel genannt werden, und Spalten. Die Spalten sind mit Attributen gleichzusetzen. Attribute nehmen einen Bereich von Werten an, die über ein Relationsschema den Typ und die Anzahl der Attribute festlegen. In relationalen Datenbanken wird jedes Tupel (Datensatz) eindeutig über den Schlüssel identifiziert. Sämtliche Daten müssen konsistent und redundanzfrei abgelegt werden, was durch Normalisierung erreicht wird. In relationalen Datenbanksystemen wird als Datenbankabfragesprache SQL verwendet.

Der Modellierungsprozess relationaler Datenbanksysteme startet, ähnlich wie bei Softwareprojekten, mit einer Anforderungsanalyse. Mit den Anforderungen als Grundlage, wird ein ER (Entity-Relationchip) Modell erstellt. Das ER-Modell besteht aus Entitäten und Attributen. Entitäten sind Objekte der realen Welt, die durch Attribute näher beschrieben werden. Diese Entitäten stehen mit anderen Entitäten in Beziehung. Die Beziehungen werden mit Kardinalitäten gekennzeichnet. Entitäten werden Primär- und Fremdschlüssel zugewiesen. Die Schlüssel bilden die spätere Grundlage für JOIN-Operationen, die aus mehreren Tupeln, unterschiedlicher Tabellen, ein

Ergebnistupel erzeugen.

Das fertige ER-Modell ist eine erste Grundlage für das Datenbankschema. Der nächste Schritt ist die sogenannte Normalisierung. Durch Normalisierung wird verhindert das Redundanzen auftreten. Nach der Normalisierung kann mithilfe der DDL (Data Definition Language) das Datenbankschema geschrieben werden.

Dieser Modellierungsprozess wurde stark komprimiert beschrieben. Eine sehr genaue Beschreibung der Abläufe findet der interessierte Leser in [EN09].

Entscheidend ist folgende Tatsache. Im ganzen Modellierungsprozess wurde nicht darauf eingegangen, welche Abfragen letztendlich an die Datenbank gestellt werden. Das ist ein gravierender Unterschied zur Modellierung von spaltenorientierten Datenbanken, denn hier wird das Datenmodell anhand der Abfragen an die Datenbank entwickelt.

Die folgenden Beschreibungen zur Modellierung beziehen sich auf Cassandra, einem populären Vertreter spaltenorientierter Datenbanken. Bevor einige Regeln zur Modellierung von spaltenorientierten Datenbanken genannt werden, ist es wichtig daran zu denken, dass es sich um verteilte Datenbanken handelt. Das Ziel sollte immer sein, das Modell so zu wählen, dass linear skaliert werden kann, wenn dem Ring (siehe Kapitel 2.2) Knoten hinzugefügt werden. Ferner gibt es bei spaltenorientierten Datenbanken keine Fremdschlüsselbeziehungen und keine JOIN-Operationen, da diese Datenbanksysteme schemafrei sind. Die Datensätze sind in verschachtelten Hashmaps organisiert.

Auch bei spaltenorientierten Datenbanken beginnt der Modellierungsprozess mit der Analyse der Anforderungen. Auch hier kann ein ER-Modell zur Modellierung der realen Welt verwendet werden. Auch sind die Beziehungen zwischen den Entitäten zu modellieren. Das fertige ER-Modell wird allerdings jetzt genutzt, um Abfragen an die Datenbank zu formulieren, die für die Anwendung wichtig sind.

Diese Art der Modellierung wird auch als Abfrage-Modell bezeichnet. Damit eine gute Performance erzielt werden kann, sollten die Abfragen so wenige Partitionen wie möglich lesen. Optimal wäre, eine Partition wird pro Abfrage gelesen.

Eine Partition ist ein Datensatz, der mithilfe des Partitionsschlüssels (Row-Key) gelesen werden kann. Dies hat zur Folge, dass das Datenmodell De-Normalisiert wird. Es werden bewusst Redundanzen in Kauf genommen bzw. sind Redundanzen ein Mittel zur besseren Modellierung. Das Datenmodell sollte zwei Zielen genügen. Die Daten müssen gleichmäßig im Cluster verteilt werden und die Anzahl der gelesenen Partitionen sollte minimiert werden.

Folgende Tabelle fasst die unterschiedlichen Ansätze der Modellierung zusammen.

	Relationale Datenbank	Spaltenorientierte Datenbank
Use Cases	Ja	Ja
ER-Modell	Ja	Ja
Analyse von Abfragen an die Datenbank	Nein	Ja - Abfrage-Getriebene-Modellierung. Die Abfrage von Partitionen muss minimiert werden.
Normalisierung	Ja	De-Normalisierung (Redundanzen als Werkzeug der Modellierung)
Primärschlüssel	Ja	Ja - Wahl des Primärschlüssels (Partitionsschlüssel) ist wichtig für gleichmäßige Verteilung im Ring.
Fremdschlüssel	Ja	Nein
JOIN-Operation	Ja	Nein - Muss auf Anwendungsebene verlagert werden.

4 Fallbeispiel Datenmodellierung

Die gewonnenen Erkenntnisse aus Kapitel 3 sollen nun an zwei praktischen Beispielen verdeutlicht werden. Das erste Beispiel, in Abschnitt 4.1, befasst sich mit einer E-Commerce Anwendung. In Abschnitt 4.2 wird eine typische Anwendung spaltenorientierter Datenbanken aus dem IOT (Internet of Things) Bereich erläutert.

4.1 E-Commerce Beispiel

Das folgende Beispiel beschreibt die Anwendungsdomäne von Hotelreservierungen. Eine detaillierte Beschreibung des Beispiels kann in [Car16] nachgelesen werden.

Als Ergebnis der Anforderungsanalyse und der Beschreibung der Use-Cases wurde das in Abbildung 9 dargestellte ER-Diagramm erstellt. Dieser Schritt ist sowohl bei relationalen Datenbanksystemen, als auch bei spaltenorientierten Datenbanken gleich.

In spaltenorientierten Datenbanken wird die Datenmodellierung mit dem Abfrage-Modell fortgesetzt. Hierzu werden die Abfragen an die Datenbank modelliert. Eines der Hauptziele der Datenmodellierung sollte die Minimierung der gesuchten Partitionen sein, um eine Abfrage zu erfüllen.

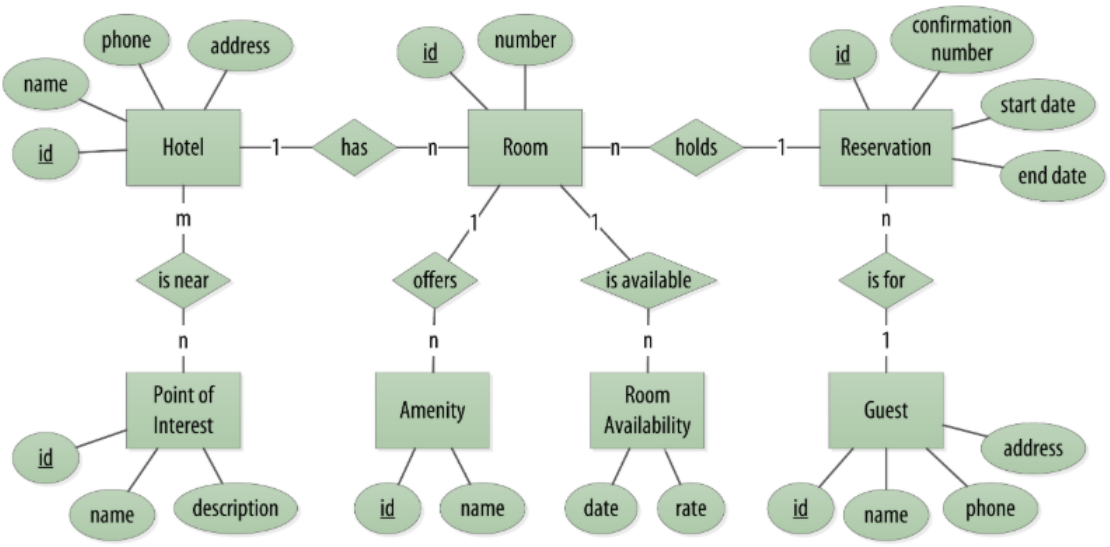

Abbildung 9: ER-Modell, Abbildung aus [Car16]

Folgende Abfragen sollte die Anwendung verstehen.

- Q1. Finden Sie Hotels in der Nähe einer bestimmten Sehenswürdigkeit.
- Q2. Finden Sie Informationen zu einem bestimmten Hotel, wie Name und Ort.
- Q3. Finden Sie Sehenswürdigkeiten in der Nähe eines Hotels.
- Q4. Finden Sie ein verfügbares Hotelzimmer in einem bestimmten Zeitraum.
- Q5. Finden Sie den Preis und die Ausstattung für ein Hotelzimmer.

Aus den Abfragen wird nun ein Workflow Diagramm erstellt. In einer guten Modellierung kann das Ergebnis einer Abfrage als Input einer folgenden Abfrage dienen. Daraus ergibt sich das in Abbildung 10 dargestellte Workflow-Diagramm.

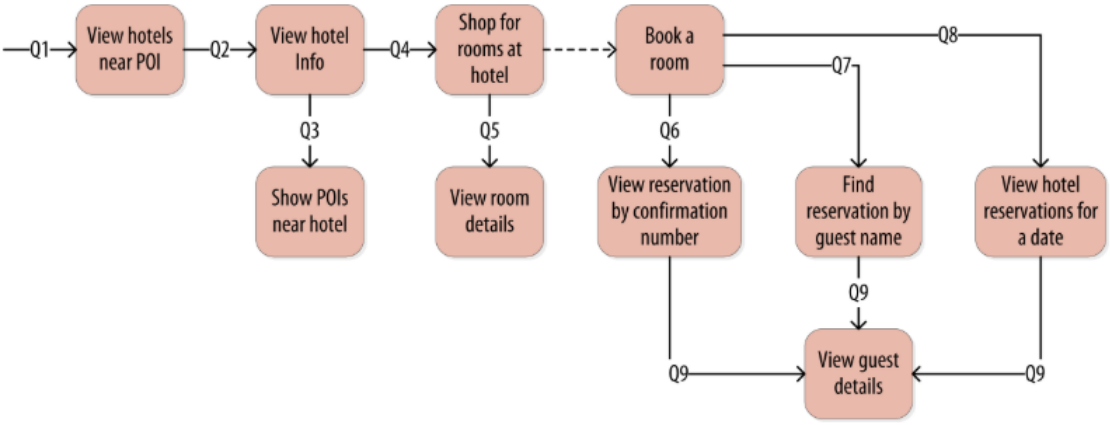

Abbildung 10: Workflow, Abbildung aus [Car16]

Das Diagramm beschreibt beispielhaft folgenden Workflow. Das Ergebnis der Abfrage Q1 enthält unter anderem die Partitionsschlüssel der Hotels, die benötigt werden, um Abfrage Q2 zu erstellen. Es ist wichtig daran zu denken, dass Datensätze nur über den Partitionsschlüssel angesprochen werden können.

Nachdem alle Abfragen erstellt wurden, können die Tabellen erstellt werden. Es ist darauf zu achten, dass Tabellen in spaltenorientierten Datenbanken mit Column Families gleichzusetzen sind. Für jede Abfrage muss eine Tabelle erstellt werden. Für diesen Teil der Modellierung dienen sogenannte Chebotko-Diagramme (siehe Abbildung 11).

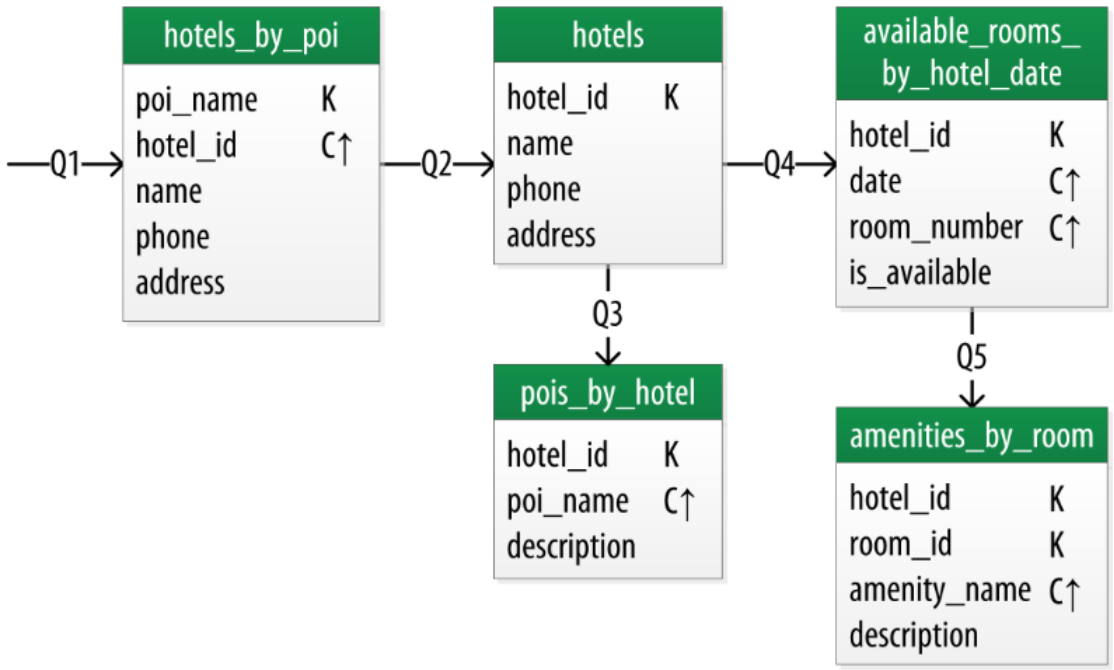

Abbildung 11: Chebotko Diagramm, Abbildung aus [Car16]

Der Workflow beginnt mit der ersten Frage Q1 „ Finden Sie Hotels in der Nähe einer bestimmten Sehenswürdigkeit“. Daraus kann der Tabellenname abgeleitet werden (hotels_by_poi). Als Input der Suche Q1 sollte der Name der Sehenswürdigkeit verwendet werden. Demzufolge ist der Name der Sehenswürdigkeit Teil des Primärschlüssels (im Diagramm mit K gekennzeichnet). Das Ergebnis der Abfrage liefert in der Regel mehr als ein Hotel. Um sicherzustellen das für jedes Hotel eine eigene Partition vorhanden ist, muss es Teil des Primärschlüssels sein (im Diagramm mit C gekennzeichnet).

Nach dem beschriebenen Muster können alle Tabellen erstellt werden. Besonders das Bestimmen des Primärschlüssels ist nicht trivial. Hier sollte auf eine gute Modellierung geachtet werden, da dies große Auswirkungen auf die Performance haben kann.

Die Erstellung der Tabellen ist die zeit intensivste Teilaufgabe im Modellierungsprozess. Nach Abschluss dieses Prozesses kann mit der physischen Datenmodellierung fortgefahren werden, sprich die Tabellen werden in die Datenbank geschrieben.

4.2 Zeitreihen - Wetterstation Beispiel

Eine der häufigsten Anwendungen spaltenorientierter Datenbanken ist die Aufzeichnung von Zeitreihendaten. Hier sind sie relationalen Datenbanken überlegen. Die Speicherung der Daten in Spalten lässt die Zeile nach rechts wachsen. Es sind bis zu 2 Milliarden Spalten in einer Zeile möglich. Ein weiterer Vorteil ist die Tatsache, dass nur Speicherplatz verbraucht wird, wenn Daten geschrieben werden. In relationalen Systemen würde ein NULL-Wert gespeichert werden, der Speicherplatz verbraucht.

Mit dem Einzug des IOT (Internet of Things) wurden spaltenorientierte Datenbanken sehr populär. Viele Sensoren liefern ständig Werte. Diese Werte sind abzuspeichern.

Als kleines Beispiel soll die Modellierung einer Datenbank von Wetterstationen dienen. Eine Abfrage der Datenbank könnte lauten „Finde alle abgespeicherten Temperaturen der Wetterstation X“. Eine weitere Anfrage wäre „Finde alle Temperaturen eines bestimmten Tages“. Das daraus resultierende Modell zeigt Abbildung 12.

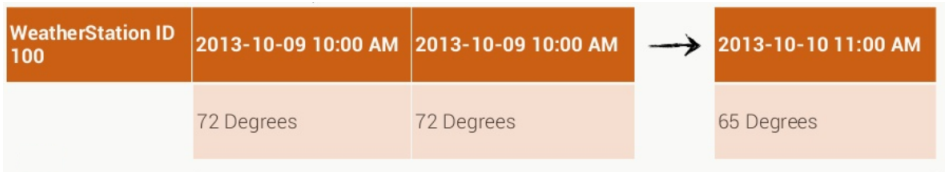

Abbildung 12: Datenmodell von Wetterstationen

Der Partitionsschlüssel ist demnach aus einer ID der Wetterstation oder aus dem eindeutigen Namen der Wetterstation zu bilden. Der Datensatz setzt sich aus den Spalten zusammen. Der Spaltenschlüssel beschreibt den Tag an dem der Wert aufgenommen wurde. Somit sind sehr schnelle Lese- und Schreib- Operationen möglich.

5 Fazit

Die Arbeit zeigte, dass spaltenorientierte Datenbanken andere Modellierungstechniken verlangen, als das bei relationalen Systemen der Fall ist. So wird bei spaltenorientierten Datenbanken das Abfrage-Modell eingesetzt, um die Performance der Datenbank nicht zu mindern. Ferner wurde festgestellt, dass Redundanzen erwünscht und als Modellierungstechnik genutzt werden. Das Erstellen der Tabellen erfordert viel Erfahrung des Entwicklers.

Beide Datenbanksysteme haben weiterhin ihre Legitimation. Es kommt bei der Wahl des Datenbanksystems immer auf die Use-Cases an. So wird ein Geldinstitut niemals ein Datenbanksystem nutzen, dass nach dem BASE-Modell entwickelt wurde. Andererseits sind relationale Datenbanken bei Zeitreihen-Daten wenig sinnvoll.

Literatur

[Car16] CARPENTER, Hewitt: *Cassandra: The Definitive Guide - Distributed Data at Web Scale*. Sebastopol : Ö'Reilly Media, Inc.", 2016. – ISBN 978–1–491–93363–3

[CDG+06] CHANG, Fay ; DEAN, Jeffrey ; GHEMAWAT, Sanjay ; HSIEH, Wilson C. ; WALLACH, Deborah A. ; BURROWS, Mike ; CHANDRA, Tushar ; FIKES, Andrew ; GRUBER, Robert E.: Bigtable: A distributed storage system for structured data. In: *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI'06)*, 2006

[DHJ+07] DECANDIA, Giuseppe ; HASTORUN, Deniz ; JAMPANI, Madan ; KAKULAPATI, Gunavardhan ; LAKSHMAN, Avinash ; PILCHIN, Alex ; SIVASUBRAMANIAN, Swaminathan ; VOSSHALL, Peter ; VOGELS, Werner: Dynamo: amazon's highly available key-value store. In: *Proceedings of the 21st ACM Symposium on Operating Systems Principles 2007, SOSP 2007, Stevenson, Washington, USA, October 14-17, 2007*, 2007, 205–220

[EFH+11] EDLICH, Stefan ; FRIEDLAND, Achim ; HAMPE, Jens ; BRAUER, Benjamin ; BRUECKNER, Markus: *NoSQL - Einstieg in die Welt nichtrelationaler Web 2.0 Datenbanken*. 2. aktualisierte und erweiterte. Muenchen : Hanser, 2011. – ISBN 978–3–446–42753–2

[EN09] ELMASRI, Ramez A. ; NAVATHE, Shamkant B.: *Grundlagen von Datenbanksystemen* -. 3. aktualisierte. Muenchen : Pearson Deutschland GmbH, 2009. – ISBN 978–3–868–94012–1

[MK16] MEIER, Andreas ; KAUFMANN, Michael: *SQL- & NoSQL-Datenbanken* -. 8. Aufl. Berlin Heidelberg New York : Springer-Verlag, 2016. – ISBN 978–3–662–47664–2

[OP14] OSMAN, Rasha ; PIAZZOLLA, Pietro: Modelling Replication in NoSQL Datastores. In: NORMAN, Gethin (Hrsg.) ; SANDERS, William H. (Hrsg.): *QEST* Bd. 8657, Springer, 2014 (Lecture Notes in Computer Science). – ISBN 978–3–319–10695–3, 194-209

[Wik19] *Spaltenorientierte Datenbanken*. Webseite, 2019. – https://de.wikipedia.org/wiki/Spaltenorientierte

Abbildungsverzeichnis