Schema-Management ohne Schema? - Schema-Verwaltung in NoSQL-Datenbanksystemen

Helena Glatzel

helenaglatzel@gmx.de
Hochschule Darmstadt, Master Studiengang Informatik

Abstract: In der Anwendungsentwicklung finden sich mittlerweile immer kürzere Release-Zyklen, auf die auch im Bereich der Datenbanken reagiert werden muss. Gerade im Umfeld der Webanwendungen erfreuen sich daher NoSQL-Datenbanksysteme zunehmender Beliebtheit. Durch deren Schema-Flexibilität kann effizient auf die schnellen Änderungen von Anwendung und Datenschema eingegangen werden. Üblicherweise ist es in NoSQL-Systemen nicht erforderlich im Voraus ein Schema zu definieren. Dieses ergibt sich implizit durch die Struktur der Daten und muss nicht gesondert definiert und verwaltet werden. Dadurch sind jederzeit flexibel Änderungen am Schema möglich. Allerdings bedeutet dies auch, dass die Verwaltung des Schemas zum Großteil in der Verantwortung der Anwendung liegt. Um die Komplexität der Anwendungsentwicklung zu begrenzen, ist ein gewisses Maß an Unterstützung durch das eingesetzte Datenbanksystem wünschenswert. Dieses Paper gibt daher zunächst einen Überblick über die momentane Unterstützung des Schema-Managements durch die gängigsten NoSQL-Datenbanken und vielversprechendsten, aktuellen Entwicklungen. Im Anschluss werden die gewonnenen Erkenntnisse genutzt, um eine systemunabhängige Schema-Management-Komponente konzeptionell zu entwerfen, die sämtliche Aufgaben des Schema-Managements erfüllt.

1 Einführung

Gerade im heutigen Web 2.0-Umfeld findet man in der Anwendungsentwicklung immer kürzere Release-Zyklen. Dies wirkt sich auch auf Datenbanken und die Struktur der Daten aus. Es ist oft nicht mehr möglich im Vorfeld ein festes Schema zu definieren, das nur wenigen Änderungen unterliegt und über viele Releases stabil bleibt.

Um diesen Eigenschaften gerecht zu werden, eignen sich NoSQL-Datenbanksysteme, denn die Schema-Flexibilität bzw. Schemalosigkeit wird als eines ihrer Kern-Features betrachtet [Tiw11]. Im Gegensatz zu relationalen Datenbanksystemen muss in den meisten NoSQL-Systemen a priori kein Schema definiert werden. Dies bedeutet jedoch nicht automatisch das komplette Fehlen eines Schemas. Die Daten selbst weisen implizit ein Schema auf, selbst wenn dieses nicht explizit modelliert wird.

Weiterhin verhalten sich NoSQL-Datenbanken hinsichtlich Schema-Änderungen sehr flexibel. Es ist jederzeit möglich Objekten neue Eigenschaften zuzuweisen, ohne dass

diese im Vorfeld definiert werden müssen (in relationalen Datenbanken mit dem SQL-Statement ALTER TABLE t_name ADD column_name datatype). Da außerdem die Heterogenität von Objekten unterstützt wird, können sich Schema-Änderungen bei Bedarf auf ein einziges Objekt beschränken und müssen nicht wie in relationalen Datenbanken für die gesamte Tabelle vorgenommen werden. Dadurch können zwei Objekte der selben (logischen) Klasse durch vollkommen unterschiedliche Attribute beschrieben werden.

Die Verwaltung des Schemas obliegt in NoSQL-Datenbanksystemen im Normalfall der Anwendung. Diese Übertragung der Verantwortung für das Schema-Management bringt zusammen mit der Schemaflexibilität von NoSQL-Datenbanksystemen eine neue Ebene der Komplexität in die Anwendungsentwicklung. Da Anwendungen mit jeglichen Varianten von Objekten umgehen können sollen, ist viel Eigenentwicklung erforderlich. Dies ist sowohl kosten- und zeitintensiv als auch fehleranfällig. Daher wäre eine (teilweise) Automatisierung des Schema-Managements und somit eine tiefergehende Unterstützung seitens der NoSQL-Datenbanksysteme wünschenswert.

Dieses Paper fasst die Ergebnisse einer Masterarbeit zum Thema „Schema-Management in NoSQL-Datenbanksystemen“ [Gla14] zusammen. Nach einer einleitenden Erläuterung der wichtigen Grundlagenbegriffe wird zunächst ein Überblick der derzeitigen Unterstützung des Schema-Managements in NoSQL-Datenbanksystemen geschaffen. Zu diesem Zweck werden die Anforderungen an ein Schema-Management identifiziert und mit Hilfe dieser Kriterien die populärsten NoSQL-Systeme analysiert. Darüber hinaus werden, basierend auf dem gleichen Anforderungskatalog, aktuelle Entwicklungen in diesem Bereich untersucht.

Im Anschluss wird der Entwurf einer modularen Schema-Management-Komponente beschrieben. Diese arbeitet systemunabhängig und erfüllt sämtliche Aufgaben des Schema-Managements.

2 Was bedeutet Schema-Management?

Grundsätzlich muss von einem Schema-Management eine Möglichkeit zur Modellierung eines Datenbankschemas geboten werden. Diese strukturelle Beschreibung umfasst die Definition verschiedener Entity-Typen mit ihren Attributen. Des Weiteren können verschiedenen Constraints bezüglich der Attribute (z.B. Einzigartigkeit „unique“ oder Existenz „not null“), sowie Beziehungen zwischen gleichen und unterschiedlichen Entity-Typen definiert werden. Werden Daten in die Datenbank eingefügt oder geändert, erfolgt eine Validierung gegen das definierte Schema. [EN05]

Darüber hinaus umfasst das Schema-Management eine Unterstützung der Schema-Evolution, also der späteren Weiterentwicklung des Schemas. Außerdem sollte eine durch Schema-Updates eventuell erforderliche Migration vorhandener Daten unterstützt werden. Hier sollten die beiden Varianten *eager* und *lazy* Migration zur Verfügung stehen. Unter eager Migration ist die Anpassung aller Datensätze an das neue Schema nach einer Schema-Änderung zu verstehen. So kann sichergestellt werden, dass nach Abschluss der Migrationsroutine alle Datensätze dem neuen Schema entsprechen. Lazy Migration führt

eine Migration erst dann durch, wenn die Daten nach einer Schema-Änderung das erste mal abgefragt werden. Ältere Einträge, auf die nicht mehr oder nur noch selten zugegriffen wird, müssen nicht an das neue Schema angepasst werden und verbleiben in der ursprünglichen Schema-Version. Dadurch werden unnötige Migrationen vermieden. Allerdings müssen Anwendungen beide Versionen des Schemas unterstützen. [KSS14]

Theoretisch ist auch eine dritte Variante denkbar. Nach einem Schema-Update können sowohl alte als auch neue Schema-Version erhalten bleiben, so dass keinerlei Datenmigration erforderlich ist. [KSS14]

Soll für eine Anwendung, die zunächst ohne Schema-Management gearbeitet hat, zukünftig eine Schema-Verwaltung eingebunden werden, muss eine Möglichkeit bestehen eine Schema-Extraktion, basierend auf den bereits in der Datenbank existierenden Daten, auszuführen.

3 Status Quo Analyse

Die nachfolgend beschriebenen Anforderungen an ein Schema-Management bauen auf typischen Eigenschaften eines Schema-Managements in relationalen Datenbanksystemen und [SKS13] auf. Basierend auf diesen Kriterien wurden sechs NoSQL-Datenbanksysteme und fünf aktuelle Entwicklungen analysiert. Es wurden NoSQL-Systeme der Kategorien Column-Family- und Document-Datenbanken betrachtet. [EFH11]

Da Key/Value-Datenbanken ein Datenmodell von geringer Komplexität nutzen, in dem einfache Key/Value-Paare ohne weitergehende Struktur-Informationen gespeichert werden, ist nicht davon auszugehen, dass diese Systeme eine Unterstützung des Schema-Managements bieten können. Daher wurden keine Datenbanken dieser Kategorie untersucht. Auch auf die Betrachtung von Graphdatenbanken wurde, aufgrund ihrer Andersartigkeit zu den Kategorien der untersuchten Datenbanksystemen, verzichtet. [RW12]

Die Ergebnisse der Analysen werden im zweiten Teil dieses Kapitels vorgestellt.

3.1 Anforderungskatalog

- Es wird eine geeignete Möglichkeit zur Schema-Beschreibung und -Speicherung benötigt. Dies schließt die Beschreibung der folgenden Eigenschaften ein:
    - verschiedene Entity-Typen
    - Struktur der Entity-Typen (Attribute)
    - Einschränkung der Attribute
        - Unterstützung der benötigten Datentypen
        - Constraints wie Einzigartigkeit (unique) oder Existenz (not null)

        - weitergehende Einschränkung des Wertebereiches auf bestimmte Wertgrenzen (z.B. Value between 0 and 10)
    - Kennzeichnung des Primary Keys
    - Beziehungen zwischen gleichen und unterschiedlichen Entity-Typen
- Schema-Extraktion
- Schema-Evolution
    - Hinzufügen eines neuen Entity-Typs
    - Löschen eines Entity-Typs
    - Umbenennen eines Entity-Typs
    - Update eines Entity-Typs
        - Hinzufügen eines Attributes
        - Löschen eines Attributes
        - Umbenennen eines Attributes
        - Verschieben eines Attributes
        - Kopieren eines Attributes
- Validierung gegen das Schema
- Koexistenz verschiedener Schema-Versionen zum Erhalt der Schema-Flexibilität
- Unterstützung der Datenmigration
    - Eager Migration
    - Lazy Migration

3.2 Analyse-Ergebnisse

Wie erwähnt wurden die sechs gängigsten NoSQL-Datenbanksysteme evaluiert. Es wurden die Column-Family-Systeme HBase [Hba14], Cassandra [Cas14] und Google App Engine Datastore (im Folgenden kurz mit Datastore bezeichnet), das auf Google's Big Table basiert [GAE14], untersucht. Die Ergebnisse des Datastore basieren auf der Nut-

zung mit dem verfügbaren Python SDK. Weiterhin wurden die Document-Datenbanken MongoDB [MDB14], CouchDB [CDB14] und Couchbase [Cou14] betrachtet.

Ergänzend wurden fünf verschiedene, aktuelle Software-Entwicklungen im Bereich des Schema-Managements analysiert. Dazu gehört KijiSchema, eine Komponente des Open Source Frameworks KijiProject, das Schema-Design und -Evolution für HBase bereitstellt. [Kij14] Außerdem wurden die beiden Mapper Hibernate OGM [OGM14] und Kundera [Kun14], sowie die Library Mongoose (für MongoDB) [Mon14] und das Java-Interface Objectify-Appengine, kurz Objectify, (für den Datastore) [OAE14] betrachtet.

Die Ergebnisse der Analysen sind in Abbildung 1 dargestellt. Ein ✓ bedeutet, dass das jeweilige Kriterium (annähernd) vollständig erfüllt wurde. Wohingegen ein **(✓)** eine nur teilweise Erfüllung, ein **X** eine mangelhafte oder keine Unterstützung kennzeichnet.

Es ist zu erkennen, dass die untersuchten NoSQL-Datenbanksysteme nur eine rudimentäre Unterstützung des Schema-Managements bieten können. CouchDB und Couchbase können sogar keine der Anforderungen erfüllen.

Im Vergleich dazu können MongoDB und HBase zumindest teilweise die Aufgaben eines Schema-Managements realisieren. In diesen beiden Systemen können zwar verschiedene Entity-Typen definiert werden, jedoch kann deren Struktur nicht modelliert werden. Daher sind in diesem Fall auch die Möglichkeiten der Schema-Extraktion und -Evolution stark eingeschränkt.

Lediglich Cassandra und der Datastore, in Verbindung mit dem Python SDK, können eine relativ umfassende Unterstützung des Schema-Managements bieten. Mit beiden Systemen kann die Schema-Modellierung fast vollständig umgesetzt werden. Einzig Einschränkungen in der Verfügbarkeit von Constraints und der Definition von Beziehungen existieren. Auch Methoden zur Schema-Extraktion und -Evolution werden von Cassandra und dem Datastore teilweise bereitgestellt.

Die Analysen der zusätzlich untersuchten Software-Entwicklungen (Tabelle 1) zeigen, dass die Schema-Modellierung und auch die Validierung in diesen Fällen sehr gut unterstützt wird. Allerdings kann alleine KijiSchema einige Methoden zur Schema-Extraktion und -Evolution zur Verfügung stellen. Weiterhin bietet einzig KijiSchema eine wirkliche Unterstützung der Koexistenz mehrerer Schema-Versionen, ist aber auf HBase beschränkt. Kundera und Hibernate OGM haben den Vorteil, dass sie den Einsatz verschiedener Datenbanksysteme ermöglichen. Jedoch befindet sich Hibernate OGM derzeit noch im Beta-Stadium und kann daher noch nicht alle geplanten Funktionen anbieten.

Abschließend lässt sich sagen, dass sowohl die analysierten NoSQL-Systeme als auch aktuelle Software-Entwicklungen in diesem Bereich die Anforderungen an ein Schema-Management nur ansatzweise erfüllen. Selbst eine Kombination der beiden, mit denen sich die Möglichkeiten der Schema-Modellierung und Validierung deutlich verbessern, ist die Funktionalität nicht ausreichend. Gerade an einer Unterstützung der Schema-Extraktionen und -Evolution sowie einer Automatisierung der Datenmigration fehlt es. Aus

diesem Grund wird im nächsten Kapitel der Entwurf einer Schema-Management-Komponente beschrieben, die systemunabhängig arbeitet und sämtliche Anforderungen des Schema-Managements erfüllen kann.

Tabelle 1: Gesamtüberblick der derzeitigen Unterstützung des Schema-Managements durch NoSQL-Datenbanksysteme und aktuelle Entwicklungen

Feature	HBase	Cassandra	Datastore	MongoDB	CouchDB	Couchbase	Kiji Schema	Mongoose	Hibernate OGM	Kundera	Objectify
Schemamodellierung	(✓)	(✓)	✓	(✓)	X	X	(✓)	✓	✓	✓	✓
Entity-Typen	(✓)	✓	✓	(✓)	X	X	✓	✓	✓	✓	✓
Struktur der Entity-Typen	X	✓	✓	X	X	X	✓	✓	✓	✓	✓
Attribut-Einschränkungen	X	(✓)	(✓)	X	X	X	(✓)	(✓)	(✓)	(✓)	(✓)
Primary Key	X	✓	✓	X	X	X	✓	X	✓	✓	✓
Beziehungen	X	X	✓	X	X	X	X	✓	✓	✓	✓
Schema-Extraktion	(✓)	(✓)	✓	(✓)	X	X	(✓)	X	X	X	X
Schema-Evolution	(✓)	(✓)	(✓)	(✓)	X	X	(✓)	X	X	X	X
Add Entity-Type	(✓)	✓	(✓)	(✓)	X	X	✓	X	X	X	X
Delete Entity-Type	(✓)	✓	X	(✓)	X	X	✓	X	X	X	X
Rename Entity-Type	X	X	X	(✓)	X	X	X	X	X	X	X
Update Entity-Type	X	(✓)	X	X	X	X	(✓)	X	X	X	X
Add Property	X	✓	✓	X	X	X	✓	X	X	X	X
Delete Property	X	✓	X	X	X	X	✓	X	X	X	X
Rename Property	X	X	X	X	X	X	✓	X	X	X	X
Move Property	X	X	X	X	X	X	X	X	X	X	X
Copy Property	X	X	X	X	X	X	X	X	X	X	X
Validierung	(✓)	✓	✓	X	X	X	✓	✓	✓	✓	✓
Mehrere Schema-Versionen	X	X	(✓)	X	X	X	✓	X	X	X	(✓)
Datenmigration	X	X	X	(✓)	X	X	X	X	X	X	(✓)
Eager Migration	X	X	X	(✓)	X	X	X	X	X	X	X
Lazy Migration	X	X	X	X	X	X	X	X	X	X	(✓)

4 Entwurf der Schema-Management-Komponente

An die nachfolgend beschriebene Schema-Management-Komponente werden ebenfalls die in 3.1 festgelegten Anforderungen gestellt. Zusätzlich soll die Komponente eine Automatisierung für den Prozess der Schema-Evolution und der Datenmigration bereitstellen. Werden Operationen auf den Daten ausgeführt, die implizit das Schema verändern, ist ein automatisches Schema-Update durchzuführen. Im Gegenzug soll im Fall expliziter Schema-Änderungen eine ggf. erforderliche Datenmigration automatisch durchgeführt werden. Im Zuge der Planung einer möglichen Umsetzung der Komponente wurde ebenfalls untersucht, welche Module der bisher analysierten Systeme ggf. wiederverwendet werden können, um den Entwicklungsaufwand möglichst gering zu halten.

4.1 Architektur

Generell existieren für die Einbindung der Schema-Management-Komponente zwei verschiedene Varianten. Die Komponente kann innerhalb des Datenbanksystems oder als externe, zusätzliche Schicht realisiert werden. Um die Unabhängigkeit vom Datenbanksystem, sowie eine Austauschbarkeit des DBMS oder ggf. eine Einbindung mehrerer, verschiedener Systeme zu unterstützen, wird die entworfene Schema-Management-Komponente, wie in Abbildung 2 dargestellt, als externe, zusätzliche Schicht geplant. [vgl. KSS14]

Wie auch in [KSS14] beschrieben, ist es auf diese Weise möglich für jede Anwendung individuell zu entscheiden, ob die Schema-Management-Komponente genutzt werden, oder ein direkter Zugriff auf das NoSQL-Datenbanksystem erfolgen soll. Die Möglichkeit ohne Schema-Management zu arbeiten wird allerdings durch die Art der Datenmigration und den unterstützten Grad der Schema-Flexibilität (vgl. hierzu 4.3 Validierungsmodi) beeinflusst. Wird beispielsweise lazy Migration genutzt ist eine Umgehung der Komponente u.U. nicht möglich, falls die Anwendung nicht mit sämtlichen Schema-Versionen kompatibel ist, auch wenn die Anwendung selbst kein Schema-Management nutzt.

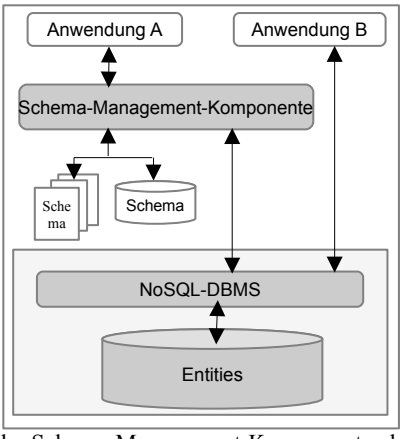

Abbildung 1: Einbindung der Schema-Management-Komponente als externe Schicht [vgl. KSS14]

Um die verschiedenen Anforderungen an ein Schema-Management zu erfüllen, ist die Schema-Management-Komponente in drei Schichten gegliedert (Abbildung 3).

Die oberste Schicht, die Daten- und Schema-Manipulations-Schicht, fungiert als Schnittstelle zum Anwender für Operationen auf den Daten und zur Schema-Evolution. In der mittleren Schema-Management-Schicht werden die unterschiedlichen Aufgaben des Schema-Managements realisiert. Diese Schicht ist in sieben Manager-Module gegliedert, damit die individuellen Anforderungen umgesetzt werden können. Die unterste Schicht, die Schema-To-NoSQL-DB-Schicht, umfasst datenbankspezifische Mapper, mit denen durchzuführende Operationen auf verschiedene Datenbanksysteme abgebildet werden.

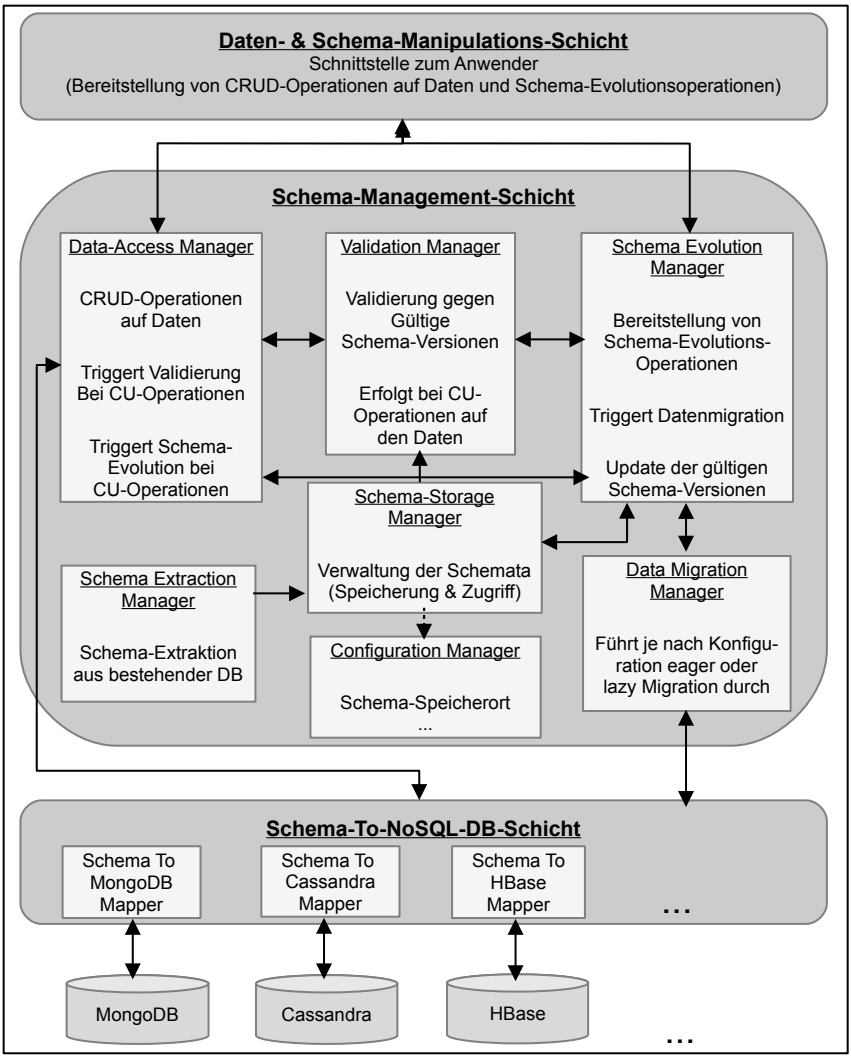

Abbildung 2: Architektur der Schema-Management-Komponente

4.2 Schema-Modellierung

Für die Umsetzung der Schema-Modellierung existieren verschiedene Möglichkeiten. So können in Cassandra die Schema-Informationen beispielsweise mit Hilfe der systemeigenen Cassandra Query Language (CQL) beschrieben werden. Auch die Nutzung von JPA-Annotationen (Java Persistence API) oder ähnlichen Abwandlungen, wie in Hibernate OGM, Kundera oder Objectify genutzt, sind denkbar.

JSON Schema [JSc14] ist eine deklarative, JSON-basierte Sprache, mit der JSON-Dokumente in ihrer Struktur beschrieben werden können. Da sich JSON Schema als Format zur Beschreibung von JSON-Dokumenten, besonders im Umfeld der Webanwendungen, etabliert hat und es viele der an die Modellierung gestellten Anforderungen erfüllen kann, wird JSON Schema in der Schema-Management-Komponente für die Schema-Beschreibung eingesetzt. Darüber hinaus verwenden viele NoSQL-Datenbanksysteme ohnehin JSON-Dokumente zur Speicherung der Daten. Aber auch die Datenstrukturen anderer Systeme lassen sich mit Hilfe dieses Formates problemlos modellieren.

Um die Schema-Verwaltung zu realisieren werden einige Attribute, wie z.B. der Entity-Typ oder die Versionsnummer, definiert, die jedes, in Verbindung mit der Schema-Management-Komponente genutzte, JSON Schema enthalten muss. Diese Eigenschaften werden in einem Metaschema definiert, gegen das alle erstellten JSON-Schema-Dokumente validiert werden.

4.3 Validierung

Werden Daten in die Datenbank eingefügt, müssen diese gegen gültige Schema-Versionen validiert werden. Hierfür existieren verschiedene Libraries für diverse Programmiersprachen (u.a Python, Java, JavaScript, PHP, ...) Diese können für die Datenvalidierung eingesetzt werden.

Für den Fall invalider Daten sind zwei verschiedene Vorgehensweisen möglich. Zum einen können invalide Daten, wie aus relationalen Datenbanksystemen bekannt, abgelehnt werden. Zum anderen ist es, dank der Schema-Flexibilität von NoSQL-Systemen, auch möglich, dass durch die Speicherung invalider Daten eine neue Schema-Version definiert wird.

Aus diesem Grund führt die Schema-Management-Komponente zwei verschiedene Validierungsmodi ein, die vom Anwender konfiguriert werden können. Wird *strictValidation* genutzt, werden invalide Daten nicht von der Komponente akzeptiert. Wird *updateValidation* konfiguriert, ruft der Data-Access-Manager für invalide Daten den Schema-Evolution-Manager auf, der die entsprechenden Schema-Updates automatisch durchführt.

4.4 Datenmanipulation

Damit der Zugriff auf die Daten über die Schema-Management-Komponente realisiert werden kann und somit eine Validierung, wie auch die automatische Schema-Evolution

und Datenmigration möglich sind, muss eine datenbanksystemunabhängige Anfragesprache zur Verfügung gestellt werden. Um die Lernkurve möglichst flach zu halten, verwendet die Komponente eine SQL-ähnliche Sprache. Voraussichtlich kann zu diesem Zweck eine Anpassung der relevanten Operationen der CQL an die Nutzung in der Komponente vorgenommen werden.

Je nach eingesetztem Datenbanksystem werden die Methoden der Anfragesprache in der Schema-To-NoSQL-DB-Schicht mit Hilfe von spezifischen Mappern auf Operationen des konkreten Datenbanksystems abgebildet. Für die Implementierung der Mapper können Funktionalitäten der datenbanksystemspezifischen Module der beiden systemübergreifenden Mapper Hibernate OGM und Kundera, nach vorheriger Anpassung, genutzt werden. Damit weitere, von diesen Mappern nicht unterstützte, Systeme von der Schema-Management-Komponente eingesetzt werden können, müssen die entsprechenden Mapper selbst entwickelt werden, falls nicht anderweitige Vorlagen existieren.

4.5 Schema-Evolution

Zur Realisierung der Schema-Evolution ist ebenfalls die Definition einer adäquaten Sprache erforderlich. Auch in diesem Fall sind verschiedene Umsetzungsvarianten denkbar. Da als Format zur Schema-Beschreibung JSON Schema gewählt wurde, könnten Evolutions-Operationen mit Hilfe von JSONiq [Jiq], einer Abfragesprache für JSON-Dokumente, durchgeführt werden. Jedoch soll die Schema-Management-Komponente nicht auf ein bestimmtes Format der Schema-Modellierung beschränkt werden, da es denkbar ist zu einem späteren Zeitpunkt ein anderes oder mehrere verschiedene Formate anzubieten. Daher wird die Nutzung einer unabhängigen Evolutionssprache bevorzugt.

Da erforderliche Anpassungen verfügbarer Lösungen, wie beispielsweise der CQL oder der von KijiSchema, relativ umfangreich sind und diese Systeme ohnehin keine vollständige Unterstützung der Schema-Evolution bieten, wird eine Eigenentwicklung vorgezogen. Zu diesem Zweck ist eine Implementierung der in [SKS13] beschriebenen Schema-Evolution-Language für die Schema-Management-Komponente vorgesehen. Diese Sprache besteht aus nur wenigen, intuitiv nutzbaren Befehlen, mit denen alle notwendigen Evolutions-Operationen ausgeführt werden können. Aufgrund des geringen Umfangs ist sie schnell zu erlernen und relativ einfach implementierbar.

Intern werden die Evolutions-Operationen, sofern entsprechende Methoden verfügbar sind, auf JSONiq abgebildet, um Änderungen an den JSON-Schema-Dokumenten durchzuführen.

4.6 Datenmigration

Die Schema-Management-Komponente unterstützt, wie oben bereits beschrieben, sowohl eager als auch lazy Migration. Die Migrationsstrategie ist Entity-Typ-basiert vom Anwender konfigurierbar.

Der Schema Evolution Manager initiiert automatisch eine erforderliche Datenmigration, die durch den Data Migration Manager ausgeführt wird. Hierfür werden die erforderlichen Migrationsschritte aus den durchgeführten Schema-Änderungen abgeleitet.

4.7 Schema-Extraktion

Eine Schema-Extraktion kann, wie in [KSS14] beschrieben, mit Hilfe eines Spanning Graph realisiert werden. Zunächst werden die Strukturinformationen der JSON-Dokumente eines bestimmten Entity-Typs eingelesen. Anschließend wird inkrementell ein Spanning Graph aufgebaut, der alle Strukturvarianten des jeweiligen Entity-Typs enthält. Aus dem Graphen kann schließlich das JSON Schema abgeleitet werden.

Dieser Prozess kann nicht nur für sämtliche Daten eines Entity-Typs durchgeführt werden, sondern auch auf einem Ausschnitt der Daten, der beispielsweise durch einen konkreten Timestamp oder ein Attribut bestimmt wird. Auf diese Weise ist die Ableitung verschiedener Schema-Varianten bzw. -Versionen eines Entity-Typs möglich. [KSS14]

5 Fazit und Ausblick

In den durchgeführten Analysen konnte festgestellt werden, dass sowohl die evaluierten NoSQL-Datenbanksysteme als auch aktuelle Entwicklungen in diesem Bereich nur eine unzureichende Unterstützung des Schema-Managements zur Verfügung stellen.

Dennoch wurden einige Teilbereiche identifiziert, in denen die Anforderungen relativ umfassend erfüllt werden. So kann, vor allem unter Einsatz zusätzlicher Aufsätze, die Schema-Modellierung weitgehend unterstützt werden. Dies ist jedoch bei Weitem nicht ausreichend, um ein annähernd zufriedenstellendes Schema-Management zu realisieren.

Aus diesem Grund wurden die gewonnenen Erkenntnisse genutzt, um eine modulare Schema-Management-Komponente zu entwerfen. Diese arbeitet systemunabhängig, erhält die Schema-Flexibilität und erfüllt alle Anforderungen, die an ein Schema-Management gestellt werden.

Als logische Fortführung dieser Arbeit bietet sich die prototypische Implementierung der entworfenen Schema-Management-Komponente an. Dies ist im Rahmen einer Projektgruppe ab dem Wintersemester 2014/15 an der Hochschule Darmstadt geplant.

Abschließend lässt sich sagen, dass die Entwicklung im Bereich des Schema-Managements von NoSQL-Datenbanksystemen noch am Beginn steht und weitere Forschungs- und Entwicklungsarbeit erforderlich ist, bevor sich erste, systemunabhängige Lösungen etablieren und den Arbeitsalltag mit NoSQL-Datenbanken merklich erleichtern.

Danksagung

Ich möchte mich bei Prof. Dr. Uta Störl bedanken, die mich während meiner Masterarbeit betreut und auch zu diesem Beitrag ein kurzes Feedback gegeben hat.

Literaturverzeichnis

[Cas14] The Apache Cassandra Project: http://cassandra.apache.org, 2014, letzter Zugriff: 21. April 2014

[CDB14] Apache CouchDB: http://couchdb.apache.org, 2014, letzter Zugriff: 20. April 2014

[Cou14] Couchbase: http://www.couchbase.com, 2014, letzter Zugriff: 21. April 2014

[EFH11] Edlich, S; Friedland, A.; Hampe, J.; Brauer, B.; Brückner, M.: NoSQL: Einstieg in die Welt nichtrelationaler WEB2.0 Datenbanken; 2. Auflage; Münschen: Carl Hanser Verlag, 2011

[EN05] Elmasri, R.; Navathe, S. B.: Grundlagen von Datenbanksystemen – Ausgabe Grundstudium; 3. Auflage. Münschen: Pearson Studium; 2005.

[HBa14] Apache HBase: https://hbase.apache.org, 2014, letzter Zugriff: 22. April 2014

[Jiq14] JSONiq: http://www.jsoniq.org, 2014, letzter Zugriff: 22. April 2014

[JSc14] JSON Schema: http://json-schema.org, 2014, letzter Zugriff: 21. April 2014

[GAE14] Google Developers, Google App Engine: https://developers.google.com/appengine/, 2014, letzter Zugriff: 21. April 2014

[Gla14] Glatzel, H.: Schema-Management in NoSQL-Datenbanksystemen. Master's thesis, Hochschule Darmstadt, 2014.

[Kij14] The KijiProject: http://www.kiji.org, 2014, letzter Zugriff: 20. April 2014

[KSS14] Klettke, M.; Scherzinger, S.; Störl, U.: Datenbanken ohne Schema? - Herausforderungen und Lösungs-Strategien mit schema-flexiblen NoSQL-Datenbanksystemen. Datenbank-Spektrum, Volume14, Nummer 2, Juli 2014.

[Kun14] Kundera: https://github.com/impetus-opensource/Kundera/wiki/Object-mapper, 2014, letzter Zugriff: 20. April 2014

[MDB14] MongoDB: http://www.mongodb.org, 2014, letzter Zugriff: 21. April 2014

[Mon14] Mongoose: http://mongoosejs.com, 2014, letzter Zugriff: 22. April 2014

[OAE14] Objectify-Appengine: https://code.google.com/p/objectify-appengine/, 2014, letzter Zugriff: 20. April 2014

[OGM14] Hibernate OGM: http://hibernate.org/ogm/, 2014, letzter Zugriff: 21. April 2014

[RW12] Redmond, E.; Wilson, J.R.: Seven Databases in Seven Weeks: A Guide to Modern Databases and the NoSQL Movement; Pragmatic Programmers, 2012

[SKS13] Scherzinger, S.; Klettke, M.; Störl, U.: Managing Schema Evolution in NoSQL Data Stores. In: Proc. DBPL; 2013.

[Tiw11] Tiwari, S.: Professional NoSQL. Indianapolis, Indiana: John Wiley & Sons; 2011.