Polyglot Persistence und NoSQL - Mehr Flexibilität, mehr Komplexität!?

Stefan Kühnlein
OPITZ CONSULTING Deutschland GmbH
München

Schlüsselworte
NoSQL, Polyglot Persistence, EclipseLink, JPA

Einleitung
Mit Polyglot Persistence wird die Verwendung von mehreren unterschiedlichen Datenquellen für eine spezifische Anwendung bezeichnet. Würde man versuchen, z. B. JSON-Dokumente in einer relationalen Datenbank zu speichern, so würde dies sicherlich funktionieren, es gäbe jedoch massive Nachteile bzgl. der Flexibilität und Effektivität beim Speichern der Daten. Um den Anforderungen von unterschiedlichen Datenquellen und Datenformaten gerecht zu werden, wurden verschiedene Datenbanksysteme entwickelt, die jeweils eine spezielle Problemdomäne adressieren. So benötigt eine nicht triviale Multiplattform-Anwendung Daten aus unterschiedlichen Formaten und Bereichen, die jeweils eine differenzierte Abfrage benötigen. Hierzu gehören Aspekte wie die Volltextsuche, Key-Value-Pairs, Suche nach geographischen Punkten und graph-transversale Abfragen.

Polyglot Persistence
Die Speicherung von Daten jeglicher Art spielt in Unternehmen eine immer bedeutendere Rolle. Auch wenn dies im Grunde nichts Neues ist, so nimmt die zu speichernde Datenmenge heute doch rasanter zu als früher. Aus diesem Grund sind die Unternehmen darauf angewiesen, Informationen über verschiedenste Kanäle zu sammeln, zu speichern und auszuwerten. Jedoch stoßen die bisherigen Verfahren bei der zeitnahen Auswertung von großen und unterschiedlichen Datenmengen an die Grenzen von Machbarkeit und Wirtschaftlichkeit. Hinzu kommen die unterschiedlichen Datenformate, die zu verarbeiten sind und die Softwarearchitekturen vor neue Herausforderungen stellen.

Dem Software-Architekten stehen einerseits verschiedene Frameworks für die Erstellung von Anwendungen wie MVC und AJAX-Frameworks (Struts, Apache MyFaces, String MVC, Grails, jQuery, dojo usw.) zur Verfügung, andererseits verfügt er über unterschiedlichste Möglichkeiten zur Speicherung der Daten. So können die Daten entweder in strukturierter, semi-strukturierter oder unstrukturierter Form vorliegen. Um den Anforderungen an die unterschiedlichsten Datenstrukturen gerecht zu werden, entstanden diverse Datenbanken wie objektorientierte Datenbanken, dokumentenorientierte Datenbanken, BigTable, Key-Value-Pair-Datenbanken. Diese Auswahl macht es möglich, die Daten entsprechend der Problemdomäne effizient zu speichern.

Hier hilft das Architekturpattern Polyglot Persistence: Je nach Struktur der Daten und deren spezifischen Anforderungen bzgl. Speicherung und Selektion kann Polyglot Persistence bei der Auswahl des passenden Persistenz-Mechanismus unterstützen. Bei dieser Auswahl werden die Anforderungen entsprechend gruppiert und einem adäquaten Persistenz-Mechanismus zugeordnet. Dabei beschränkt man sich nicht auf eine einzige Datenbank, sondern kombiniert mehrere innerhalb einer Anwendung.

So kann eine Web-Anwendung die folgenden Persistenz-Mechanismen adressieren:

- relationale Datenbank zur Speicherung von strukturierten Daten wie Profile der Anwender, Rechnungsinformationen u. s. w.
- dokumentenorientierte Datenbank zur Speicherung von Logs, Tweets, Hit Counts, usw.…
- externer Datenspeicherservice zur Speicherung von Bildern

Aufgrund der Vielzahl von Anforderungen an Unternehmensanwendungen müssen entsprechende Faktoren bei der Auswahl des Persistenz-Mechanismus berücksichtigt werden:

- Skalierbarkeit
- Verfügbarkeit
- Fehlertoleranz
- Verteilbarkeit
- Flexibilität der Datenbankschemas

Nicht zuletzt sollten auch die neusten Trends wie Soziale Medien und Internet of Things mit berücksichtig werden, um die Anwendung für neue Anforderungen offenzuhalten.

NoSQL
NoSQL-Datenbanken sind aus der konsequenten Weiterentwicklung von unternehmensweiten Anwendungen entstanden. Diese Entwicklung startete in den Neunzigerjahren mit Client-Server-Anwendungen und entwickelte sich über die Drei-Schichten-Architektur bis zu serviceorientierten Architekturen und Micro Services.

NoSQL wird im Sprachgebrauch oft als ein nicht SQL-basiertes Datenbankmanagementsystem beschrieben. In den meisten Fällen wird damit jedoch nicht erklärt, was eine NoSQL-Datenbank wirklich ist. Betrachtet man die letzten vierzig Jahre in der Entwicklung von relationalen Datenbanken, so werden diese durch die transaktionalen ACID-Eigenschaften wie Atomic, Consistency, Isolation und Durability charakterisiert. Diese Eigenschaften lassen sich nicht auf eine NoSQL-Datenbank übertragen. Im Gegensatz dazu werden NoSQL-Datenbanken durch das Akronym BASE beschrieben:

- **B**asically **A**vailable: Mit Hilfe von Replikation wird die Wahrscheinlichkeit reduziert, dass Daten nicht verfügbar sind. Alternativ kann die Verfügbarkeit mit Hilfe von Partitionierung und durch die Verteilung der Daten auf verschiedene Datenspeicher gewährleistet werden. Damit kann erreicht werden, dass die Systeme verfügbar sind, auch wenn nicht auf alle Daten für einen bestimmten Zeitraum zugegriffen werden kann.

- **S**oft State: Bei ACID-Systemen wird davon ausgegangen, dass die Konsistenz der Daten jederzeit gewährleistet ist. Bei NoSQL wird eine Inkonsistenz der Daten zugelassen und es ist den Anwendungsentwicklern überlassen, damit adäquat umzugehen.

- **E**ventually consistent: Obwohl Anwendungsentwickler bei NoSQL-Datenbanken mit inkonsistenten Daten umgehen müssen, sorgen die Datenbanken dafür, dass von einem konsistenten Zustand der Daten zu einem späteren Zeitpunkt ausgegangen werden kann. Im Gegensatz zu ACID-System, bei denen die Transaktion die Konsistenz der Daten erzwingt, wird diese bei NoSQL-Datenbanken zu einem späteren Zeitpunkt gewährleistet.

Früher bauten NoSQL-Lösungen auf relationalen Datenbanken auf. Jedoch wurde sehr schnell klar, dass mit relationalen Datenbanken nicht die Zugriffsmuster und Latenzanforderungen einer echten NoSQL-Datenbank erreicht werden können. Die Grundlage für die heutige Oracle NoSQL Datenbank lieferte die Berkeley DB, die als reiner Key-Value-Speicher ausgelegt war. Die Oracle Berkeley DB Java Edition gibt es nun schon seit mehr als acht Jahren und sie gewährleistet die Anforderungen an Robustheit, Stabilität und Verfügbarkeit einer NoSQL-Landschaft.

Bis vor kurzem waren für die Integration von NoSQL-Lösungen in eine Unternehmensarchitektur noch eine anwendungsspezifische Entwicklung und eine manuelle Integration notwendig. Mit der Oracle NoSQL-DB ist nun die nahtlose Integration einer NoSQL-Anwendung in eine Unternehmensarchitektur möglich.

NoSQL-Datenmodell
In der einfachsten Form erstellt die Oracle NoSQL-Datenbank eine Abbildung der benutzerdefinierten Schlüssel zu den dahinter liegenden Daten. Obwohl für jedes Key/Value Pair eine interne Versionsnummer hinterlegt ist, wird nur die neueste Version im Speicher verwaltet. Anwendungen nutzen diesen Umstand, um die Konsistenz der Daten im Rahmen von Read-Modify-Write-Operationen sicherzustellen.

Wie oben bereits erwähnt, können die Daten zu einem bestimmten Zeitpunkt inkonsistent sein. Aus diesem Grund stellt die Oracle NoSQL-Datenbank verschiedene Policies zur Datenkonsistenz bereit. So kann der Anwendungsentwickler zwischen Policies wie „None“, „Time-Based“, „Version-Based“ und „Absolute“ auswählen.

NoSQL-APIs
Mit der Oracle NoSQL-Datenbank werden APIs für Create-, Read-, Update- und Delete-Operationen ausgeliefert. Ebenso sind in den APIs in einer einzigen jar-Datei Funktionen für Iterationen über die Datensätze sowie zum Setzen der Policies für Konsistenz und Lebensdauer enthalten. Anwendungen können diese jar-Datei in den Klassenpfad mitaufnehmen und so auf die Datenbank zugreifen.

Für die Erstellung und Aktualisierung der Daten stehen folgende Funktionen zur Verfügung:

- `putIfAbsent` -> erzeugt einen neuen Datensatz
- `putIfPresent` -> aktualisiert einen bestehenden Datensatz
- `putIfVersion` -> ermöglicht die Implementierung von konsistentem Schreiben und Lesen
- `put` -> erstellt einen neuen Datensatz, sofern dieser noch nicht vorhanden ist

Zum Löschen von Datensätzen stehen die Methoden `delete` und `deleteIfVersion` zur Verfügung wobei eine entsprechende Bedingung mitgegeben werden kann.

Zum Lesen von Datensätzen werden verschiedene zwei Gruppen von Methoden angeboten. So können mit `get` einzelne Datensatzes gelesen werden. Mit `multiGet` können mehrere Datensätze gelesen werden.

Zusätzlich zu den CRUD-Operationen werden mit der API der NoSQL-Datenbank noch zwei Arten für die Iteration mitgeliefert: unsortierte Iteration und sortierte Iteration über den Major Key der Datensätze. Die API unterstützt sowohl individuelle Key/Value über die Funktionen `storeIterator` und Bulk Key/Value über die Funktionen `multiGetIterator`.

NoSQL und JPA
Seit EclipseLink 2.4 unterstützt EclipseLink über die Java Persistence API (JPA) auch NoSQL-Datenbanken wie MongoDB und Oracle NoSQL. Somit wird die Entwicklung von Anwendungen unterstützt, die sowohl relationale als auch NoSQL-basierte Datenbanken benötigen. Hierzu wurden die JPA-API- und die JPA-Annotationen erweitert, so dass aus Java-Programmen direkt auf die NoSQL Daten zugegriffen werden kann. Der Zugriff auf die NoSQL-Datenbank mithilfe von EclipseLink erfolgt durch den JavaEE Connector Architecture. Hierzu muss ein JCA Adapter verwendet werden, der entweder von EclipseLink mitgeliefert wird, durch einen Drittanbieter bereitgestellt wird oder selbst entwickelt werden muss.

Soll z. B. in einer Anwendung ein JSON-Objekt in einer NoSQL-Datenbank gespeichert werden, so sind sechs Schritte durchzuführen:

1. **Mapping der Daten**
   Die Konfiguration des Daten-Mappings zu einer NoSQL-Datenbank erfolgt durch die Annotation `@NoSQL` oder über das XML-Element `<no-sql>`. Über diese Annotation erfolgt das Mapping der Klasse auf eine nicht relationale Datenstruktur. Die Annotation `@NoSQL` kann sowohl für JPA-Entitäten als auch für Embeddable-Klassen verwendet werden.

2. **Definition der IDs**
   JPA benötigt für jede Entität eine eindeutige ID. Wie bei relationalen Datenbanksystemen kann bei NoSQL entweder ein natürlicher Schlüssel oder ein durch EclipseLink generierter Schlüssel verwendet werden. Bei der Verwendung einer Oracle NoSQL-Datenbank wir automatisch das Feld `id` verwendet.

3. **Definition des Mappings**
   Für jedes Attribut eines Objektes sollte ein Mapping definiert werden. Wird kein Mapping definiert, so wird die Default-Einstellung verwendet. Die Regeln für die Defaults bei NoSQL Datenbanken folgen den Regeln der Defaults bei relationalen Datenbanken. So werden z. B. bei einem einfachen Mapping keine weiteren Annotationen benötigt.

4. **Definition der Lockingstrategie**
   Die Lockingstrategie ist aktuell von der NoSQL-Datenbank abhängig Die Oracle NoSQL-Datenbank unterstützt kein Optimistic Locking. Sollte jedoch ein Optimistic Locking gewünscht sein, so kann die Annotation `@Version` verwendet werden, damit die Objekte im Rahmen der Operation `merge()` validiert werden.

5. **Definition von Abfragen**
   Die Definition von Abfragen ist ebenfalls von der NoSQL-Datenbank abhängig. Mit JPA wird die Java Persistence Query Language (JPQL) mitgeliefert. JPQL kann sowohl für das Lesen als auch für Bulk Updates und das Löschen verwendet werden. JPQL kann über die Annotation `@NamedQuery` definiert werden oder dynamisch über die Methode `createQuery()` des EntityManager verwendet werden.

6. **Verbindung zur Datenbank**
   Der Verbindungsaufbau zur NoSQL Datenbank erfolgt ebenfalls mit Hilfe der persistence.xml-Datei. Mit dem Element `<eclipselink.target-database>` wird die NoSQL-Datenbank spezifiziert. Die Verbindung zur Datenbank erfolgt mit der Spezifikation des Attributs `<eclipselink.nosql.connection-spec>`.

Mehr Flexibilität mehr Komplexität?
Mit der Kombination von relationalen und NoSql-Datenbanken können hybride Anwendungen entwickelt werden, die sowohl strukturierte als auch unstrukturierte Daten verarbeiten und speichern können. So können weiterhin die strukturierten Daten in den relationalen Datenmodellen abgelegt werden. Alle anderen Daten können in einer NoSQL-Datenbank ausgelagert werden. Dadurch wird eine maximale Flexibilität erreicht, da eine NoSQL-Datenbank die Daten nicht in ein starres Schema presst.

Polyglot Persistence kann sowohl in unternehmensweiten Anwendungen als auch in einer einzigen Anwendung zu Einsatz kommen. Durch die Kapselung der Datenzugriffe durch Services kann die Auswirkung auf die Datenspeicherung reduziert werden. Allerdings sollte auch berücksichtig werden, dass die Verwendung von mehreren Speicherungsarten die Komplexität der Daten anwachsen lässt und dies entsprechend abgewägt werden sollte.

Kontaktadresse:
Stefan Kühnlein
OPITZ CONSULTING Deutschland GmbH
Weltenburger Strasse 4
D-81677 München

Telefon:	+49 (89) 680098-0
Fax:	+49 (89) 680098-4400
E-Mail	stefan.kuehnlein@opitz-consulting.com
Internet:	**http://www.opitz-consulting.com**