Artikeldaten in Datenbanken: Von einem relationalen Schema zu „Dokumenten“ (1/2)

Die Problemstellung

Die geeignete technische Ablage von Artikeldaten beschäftigt uns quasi beruflich. Für Leser, die keinen Bezug zu dieser Domäne haben, hier eine kurze Erläuterung.

Artikel haben (u.a.) Attribute. Die Art und Anzahl von Attributen pro Artikel ist je nach Art (Sortiment) des Artikels (Buch, Haus, Auto, Kaugummi) sehr unterschiedlich.

Artikel haben Attributwerte in diesen Attributen. Nicht jeder Artikel eines Sortiments hat für jedes Attribut einen Wert. Beispielsweise könnte eine Gewichtsangabe fehlen, weil der Hersteller diese nicht zur Verfügung gestellt hat.

Ein Artikel hat mäßig viele Attribute. Beispiel:

Abbildung 1 – die naive Version

Die Realität ist etwas anstrengender als diese Abbildung. Es kann jederzeit neue Attribute in bestehenden Sortimenten oder ganz neue Sortimente geben. Wollte man alle Sortimente in dieselbe DB Tabelle stecken, käme etwas langes und sehr breites heraus – wobei nur ein Bruchteil der Spalten pro Artikel wirklich befüllt

wäre. Der Rest bliebe leer.

Wollte man diesen Aspekt komplett flexibilisieren (z.B. in einem universell einsetzbaren PIM System), könnte man die Daten nach dem EAV (entity attribute value) Modell ordnen.

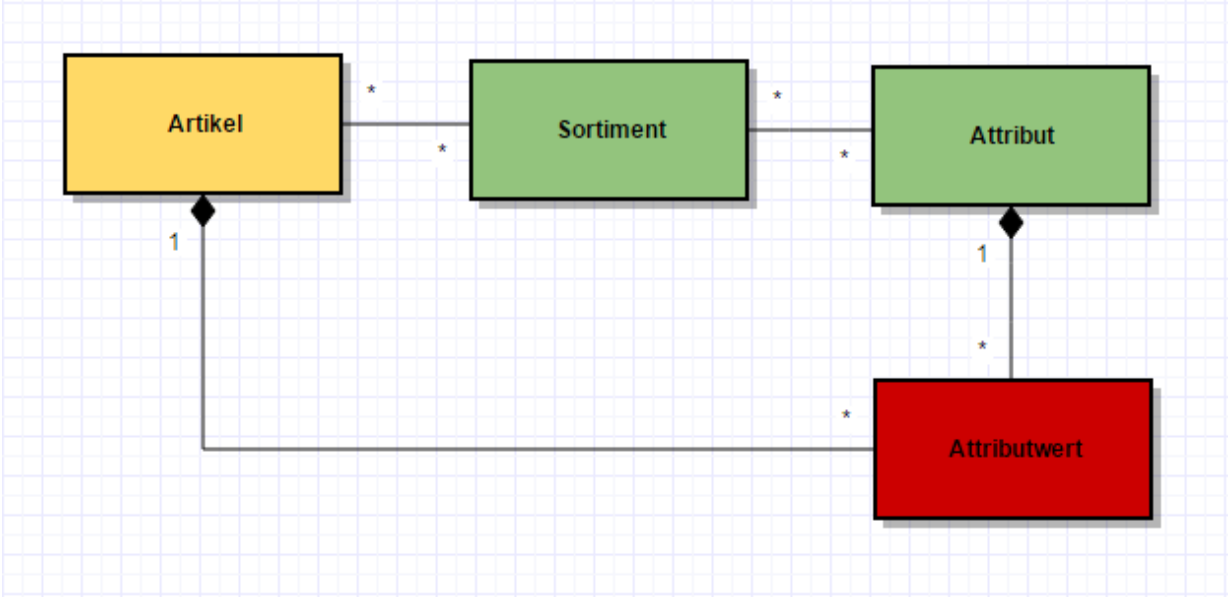

Abbildung 2 – entity, attribute, value (EAV)

Zusätzlich zu einfachen Attributen gibt es noch komplexere Daten, die einen Artikel beschreiben: Preise, Bilder, Autoren, Hersteller, Track-Listen, …

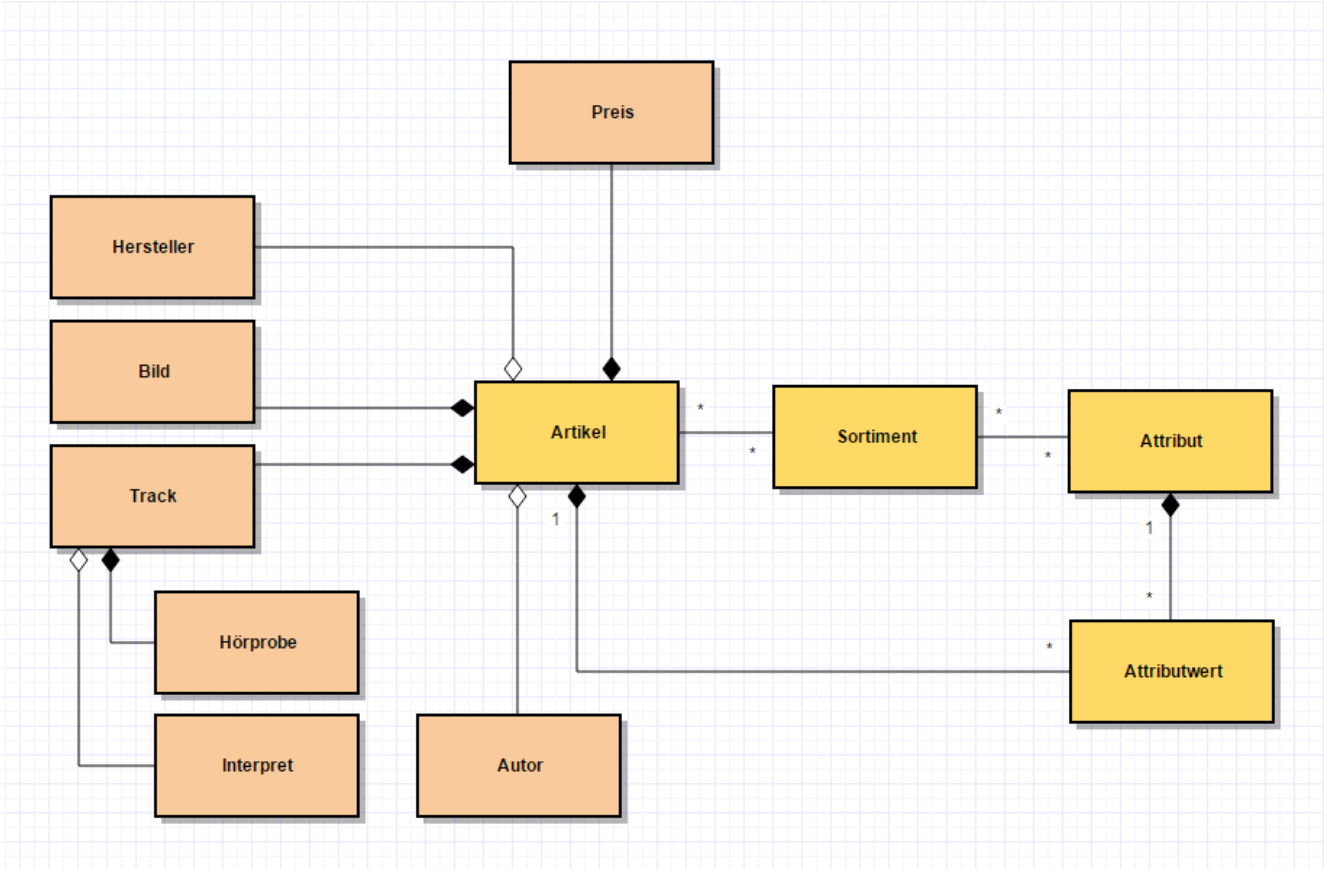

Abbildung 3 – EAV mit totaler Flexibilität

Klasse, total flexibel. Aber die Realität ist hier leider noch nicht am Ende.

Wir haben viele Artikel. Bei hypothetisch angenommenen 50 Millionen Artikel aus 10 Sortimenten mit jeweils 30 Attributen von denen 20 mit Werten belegt sind, ergibt das schon mal **eine Milliarde Zeilen** in der Attributwert-Tabelle. So eine Tabelle (wenige Spalten und viele Zeilen) wird auch „long and skinny“ genannt. Geht – ist aber in der Größe garantiert spaßbefreit.

Auf diese Tabellen müssen hochperformante CRUD-Aktionen möglich sein. Das ist heutzutage gleichzusetzen mit massiv parallelem Schreiben und Lesen. Ersteres ist hinsichtlich von DB Sperren problematisch, zweiteres macht aufgrund verschachtelter „joins“ auch keinen Spaß.

Die Einträge in der „Attributwert“-Tabelle sind relativ generisch, d.h. die Tabelle muss Attributwerte unterschiedlicher Datentypen halten. Das läuft auf String/textbasierte Spalten hinaus, eventuell sogar um Metainformationen bereichert. Die Validität der geschriebenen Daten (z.B. Typsicherheit) kann das DBMS nicht sicherstellen, d.h. dies (und die Konvertierung) muss die Applikation machen. Dabei ist das Abändern von Daten außerhalb der Applikation („mal eben

auf der Datenbank“) ein gefährlicher Stunt.

Eine weitere, relativ normale Anforderung ist die Möglichkeit der Auswertung und Analyse von Daten. Beispielsweise soll die Qualität und Quantität der Befüllung des Attributs „Laufzeit“ im Sortiment „Bluray“ für die Artikel geprüft werden, deren Titel mit „A“ beginnen. Das Entwerfen so einer Abfrage ist nicht wild – eine effiziente und effektive Indizierung mit DB-Mitteln schon.

Und was nun?

Ein Lösungsweg

Wie häufig bei neuen Lösungen besteht der Trick darin, an der richtigen Stelle an den Anforderungen zu drehen.

Die Zugriffszeiten müssen sehr gut sein, da kommen wir im eCommerce nicht drum herum. Ebenso muss die Schreibgeschwindigkeit so hoch sein, dass sämtliche anfallenden Aktualisierungen souverän und zeitnah weggearbeitet werden.

Der Aufwand zur Erweiterung oder Kürzung der Attributmengen und Sortimente sollte nahe null gehen. Das ist einfach Tagesgeschäft und muss ohne großen Eingriff der Softwareentwicklung und Koordinationsaufwände mit Abnehmern der Daten passieren.

Daher sind wir im Bereich der Artikeldaten (PIM) gerade bei dem Schlagwort „schemalos“ hellhörig geworden. Wir haben uns entschlossen, am Grad der Normalisierung des Datenmodells zu experimentieren. Was bedeutet das?

Ich verweise nochmal auf das Diagramm aus Abbildung 3. **Niemand** hat gefordert, dass der „Autor“ eines Artikels normalisiert werden **muss**. Die Anforderung lautet (korrekterweise): Eine Änderung an den Metadaten eines Autors soll auch an jedem Artikel des Autors sichtbar werden. Dem Anforderer selber ist es aber total egal, wie das in einer Persistenzschicht realisiert wird!

Noch einfacher wird es bei Bildern, Tracks und deren Hörprobe. Diese Daten „gehören“ zu einem Artikel (siehe Komposition), d.h. hier wird es keine Änderungen an den Metadaten mit Auswirkung auf viele Artikel geben.

Unser Lösungsweg ist also, alle Informationen, die nicht unbedingt normalisiert

werden müssen, in ein „Dokument“ in der Datenbank zu schreiben. Das reduziert die Anzahl der benötigten Tabellen und Referenzen drastisch. Speziell die oben erwähnte Attributwert-Tabelle mit ihren 10^9 Einträgen verschwindet komplett.

Wie viele und welche Dokumente? Wie kommt man zum korrekten Schnitt?

In unserem konkreten Fall führen wir die Artikeldaten aus unterschiedlichen Quellsystemen zusammen. Jedes Quellsystem pflegt seine eigenen Update-Frequenzen und –Formen. Um sich beim parallelen Schreiben auf keinen Fall ins Gehege zu kommen und um im Falle einer Aktualisierung auch nicht ganz viel vermeidbares Zeug schreiben zu müssen, kriegt jede dieser Quellen sein eigenes Dokument. Ausgelesen wird dies – in SQL-Termini gesprochen – über ein „join“ über alle Dokumente eines Artikels, wobei Pflichtteile und optionale Teile unterschieden werden.

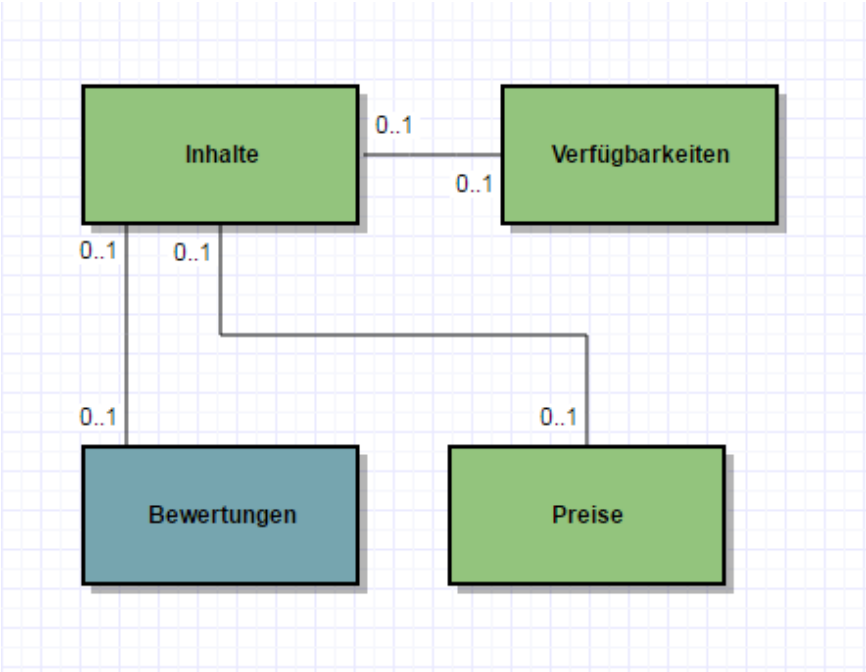

Abbildung 4 – Vereinfachung durch Dokumente

Ein „select“ eines ganzen Artikels wird somit trivial, und unvollständige Artikel gibt es in der Ergebnismenge nicht…

Dokumente! Und nun? NoSQL??

Im eCommerce und speziell im Bereich großer Datenmengen und/oder hoher Abfrageraten existiert eine Menge von Speziallösungen, um die verschiedensten Anforderungen adäquat erfüllen zu können. Ein einziges Werkzeug, um alle

Probleme zu beherrschen, wird es – da leg ich mich mal fest – nicht geben.

Speziell in den Bereich der Datenbankmanagementsysteme (DBMS) geschaut, tummeln sich unter dem Begriff des „NoSQL“ sehr viele Spezialisten. Sind also die bisherigen DBMS alle ungeeignet für eCommerce und nur Dinosaurier? Natürlich nicht. Die Spezialisten sind sehr stark in ihrem Bereich, weil sie bewusst auf Features verzichten, die in anderen Anwendungsfällen überlebenswichtig wären (z.B. ACID Transaktionen).

Aber welcher „Spezialist“ passt auf unseren Anwendungsfall und den beschriebenen Lösungsweg?

Das haben wir ausprobiert – wir berichten im nächsten Teil dieses Artikels…