



Speicherung von RDF in Datenbanken

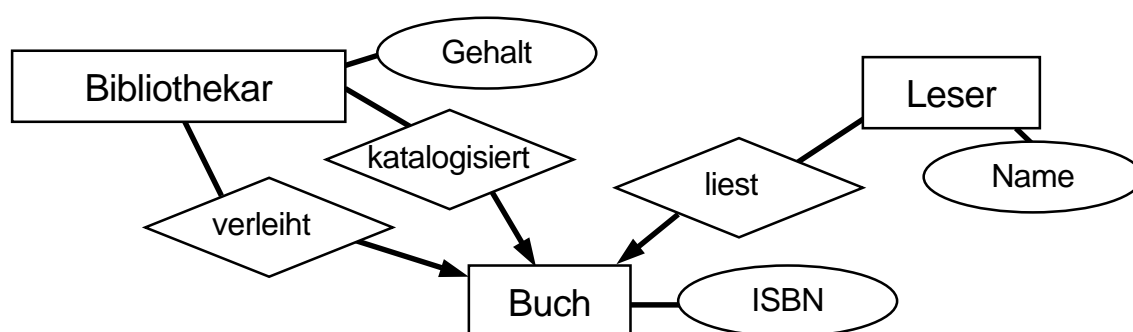
Um die spezielle Problematik, die RDF im Zusammenhang mit der Speicherung in Datenbanken verursacht, zu diskutieren, sollen zunächst die Grundzüge der Idee von Datenbanken wiederholt werden:

Es gibt viele Gründe Daten nicht in Dateien sondern in Datenbanken zu speichern. Einige davon sind die Vermeidung von Redundanz und damit verbundener Inkonsistenz. Werden z.B. Namen von Autoren zusammen mit jedem Buch, das sie geschrieben haben, gespeichert, wird einerseits unnötig Speicherplatz verbraucht. Andererseits kann es zu Unstimmigkeiten kommen, weil eine Korrektur des Namens nur in einem aber nicht in allen Datensätzen vorgenommen wird.

Sollen Daten miteinander verknüpft werden, so müssen alle Dateien unter hohem Rechenaufwand durchsucht werden. Ein Beispiel dafür ist die Erstellung einer Liste von Büchern eines Autors.

Aus diesen komplizierten Zugriffen auf die Daten folgen dann hohe Entwicklungskosten für relativ einfache Operationen.

Abstrakte Sicht auf Daten



Ein Modell zur abstrakten Darstellung von Daten ist das Entity-Relationship Modell. Einheiten oder Entitäten werden dabei als Rechtecke, Eigenschaften der Entitäten durch Ellipsen dargestellt.

Die Entitäten stehen zueinander in Beziehung. Diese Beziehungen werden durch Rauten gekennzeichnet. Das Modell erinnert sehr stark an RDF, obwohl es sehr viel älter ist.

Beziehungen oder Relationen können verschiedene Stelligkeit haben. Wir unterscheiden die 1:1-, 1:N-, N:1-, N:M-Relationen. Die 1:1-Relation begegnet uns in der Beziehung „verheiratet mit“, denn jede Ehefrau ist mit genau einem Ehemann verheiratet. Ein Beispiel für eine 1:N Relation ist, dass eine Bibliothek ein Buch aufstellt. Die Bibliothek hat diese Beziehung natürlich zu vielen Büchern, aber jedes Buch kann nur einer Bibliothek gehören. Damit ist auch intuitiv klar, was eine N:1 Beziehung ist. Darüber hinaus gibt es noch die N:M Beziehung, die im Bild in der Relation „lesen“ auftaucht. Die meisten Leser lesen viele Bücher und jedes Buch wird auch von vielen verschiedenen Lesern gelesen.

Aufbauend auf dem Entity-Relationship Modell sollen nun zwei Datenbank Architekturen vorgestellt werden, die Daten unter den Kriterien Redundanzfreiheit, Konsistenz, Sicherheit und Effizienz bei Speicherung und Zugriff aufbereiten. Es handelt sich dabei um relationale und objektorientierte Datenbanken.

Relationale Datenbanken

Relationale Datenbanken sind die derzeit gängigste Datenbank Architektur (Oracle, DB2, Microsoft Access, msql,...), die durch SQL (Standard Query Language) weitgehend standardisiert sind. SQL geht zurück auf den von IBM Anfang der 70er Jahre entwickelten Prototyp System R mit der Anfragesprache Sequel. Theoretisch ist SQL eine Mischung aus einem prozeduralen (relationale Algebra) und deklarativen Ansatz (Relationenkalkül).

Die Idee bei den relationalen Datenbanken ist die Speicherung der Entitäten, Attribute und Relationen in verschiedenen Tabellen. Der Zugriff erlaubt die gleichzeitige Nutzung mehrerer Tabellen mit verschachtelten Bedingungen. Umgesetzt werden die Anfragen und die Eingabe der Daten durch SQL.

Anhand von Beispielen soll im Folgenden erklärt werden, wie eine effektive Nutzung der Architektur möglich ist. Wir denken uns dabei die ISBN als einen eindeutigen Kennwert für Bücher, auch wenn dies in der Realität nicht immer der Fall ist. Man ersetze ggf. ISBN durch eine bibliotheksinterne eindeutige Kennung.

Beispiel (schlecht)

Bücher		
ISBN	Titel	Autor
12345	Datenbanken	Erika Muster
34567	XML Spec	Erwin Mai
78910	Graf und Welt	Erika Muster, Erwin Mai

Dieses Beispiel ist schlecht, weil die Anfrage nach "Muster, Erwin" ein Ergebnis liefert, obwohl es keinen Autor mit diesem Namen gibt. Die **erste Normierung** fordert deshalb, dass es keine mengenwertigen Attribute gibt. Attribute müssen immer atomar sein.

Beispiel (schlecht)

Bücher		
ISBN	Titel	Autor
12345	Datenbanken	Erika Muster
34567	XML Spec	Erwin Mai
78910	Graf und Welt	Erika Muster
78910	Graf und Welt	Erwin Mai

Jetzt sind alle Attribute in der Tabelle atomar. Allerdings ist der Titel des Buches „Graf und Welt“ doppelt abgespeichert. Es entsteht der Eindruck, dass der Titel eines Buches von der ISBN und dem Autor abhängt. Der Titel ist aber durch die ISBN schon eindeutig bestimmt. Das Paar (ISBN, Autor) liefert einen sogenannten Schlüssel für die Tabelle. D.h. alle weiteren Attribute der Tabelle sind durch diese Schlüsselattribute eindeutig bestimmt. Die **zweite Normierung** fordert nun, dass es keine Attribute gibt, die nicht zur Menge der Schlüsselattribute gehören, aber bereits durch einen Teil des Schlüssels bestimmt sind, wie im Beispiel der Titel bereits durch die ISBN bestimmt ist. Um das Problem zu umgehen, wird die Tabelle gespalten.

(Der Vollständigkeit halber, sei hier auch die **dritte Normierung** erwähnt. Sie verlangt, dass Attribute, die keine Schlüsselattribute sind, nicht transitiv von den Schlüsselattributen abhängen. Ein Beispiel dafür wäre eine Tabelle von Personen mit Name, Geburtsdatum, Postleitzahl und Wohnort. Der Name liefert zusammen mit dem Geburtsdatum den Schlüssel. Von diesem Schlüssel hängt die Postleitzahl ab und davon wiederum der Wohnort. Das heißt, der Wohnort ist transitiv abhängig vom Schlüssel. Dies soll durch weiteres Aufspalten der Tabellen vermieden werden.)

Auf die Normierungen soll an dieser Stelle nicht weiter eingegangen werden, die schlechten Beispiele bekommen nur noch eine Überarbeitung:

Beispiel

Bücher	
ISBN	Titel
12345	Datenbanken
34567	XML Spec
78910	Graf und Welt

Autoren	
Id	Name
1	Erika Muster
2	Erwin Mai

geschrieben von	
ISBN	AutorenId
12345	1
34567	2
78910	1
78910	2

SQL Anfragen

Der Aufbau eines relationalen Datenbankschemas ist nun detailliert besprochen. Es sollen hier noch zwei kurze Beispiele gegeben werden, wie Anfragen an ein solches Schema gestellt werden können. Als erstes soll eine Liste aller Titel von Büchern aus dem oben konstruierten Datenbankschema ausgegeben werden:

```
select Titel from Bücher
```

Ein komplexeres Beispiel liefert die Frage nach den Autoren des Buches mit dem Titel „Graf und Welt“. Um die Spalten der einzelnen Tabellen nicht zu verwechseln, werden den Tabellen in der Anfrage Namen (a,g,b) gegeben. Die Spaltennamen werden dann mit einem Punkt an die Tabellennamen angehängt. Verwechslungen sind dadurch ausgeschlossen.

```
select a.Name from Autoren a, geschrieben_von g, Bücher b where  
b.Titel=„Graf und Welt“ and g.ISBN=b.ISBN and g.AutorenId=a.Id
```

Relationale Datenbanken können an dieser Stelle nicht umfassend behandelt werden, die obige Beschreibung reicht jedoch aus, um die Probleme und Möglichkeiten zu verstehen, die bei der Speicherung von RDF entstehen.

Objektorientierte Datenbanken

Ein weiteres Beispiel einer Datenbankarchitektur stellt der objektorientierte Ansatz dar. Ein Objekt wird hier durch Attribute und Relationen zu anderen Objekten beschrieben. Objekte können aus unseren obigen Beispielen wieder Bücher, Autoren, Leser oder einfach Personen sein.

Der erste Unterschied zu dem relationalen Ansatz ist also das Fehlen der Tabellen. Dies ist vielleicht ein Verlust an Übersichtlichkeit. Allerdings entspricht es eher dem Entity-Relationship Modell, dass die Relationen zusammen mit den Entitäten gespeichert werden.

Der große Vorteil des objektorientierten Ansatzes ist allerdings die Vererbung. Vererbung meint, dass es allgemeinere Objekte gibt und deren Spezialisierungen. Ein Beispiel ist die Speicherung von Autoren, Lesern und Bibliothekaren in einer Bibliothek. Alle drei sind Personen, haben einen Namen, ein Geburtsdatum, eine Adresse. Zusätzlich müssen aber weitere Daten gespeichert werden: Der Autor hat vielleicht einen Künstlernamen und Bücher geschrieben. Vom Leser muss gespeichert werden, welche Bücher er ausgeliehen hat. Eine Kodierung dieser Objekte könnte wie folgt aussehen.

```
class Person {  
    attribute string name; }  
class Autor extends Person {  
    attribute string Künstlernamen;  
    relationship set (Buch) schreibt inverse Buch::geschrieben_von;}  
class Leser extends Person {  
    relationship set (Buch) entleiht inverse Buch::entliehen_von;}  
}
```

Will man Vererbung oder Spezialisierung im relationalen Modell realisieren, so muss eine weitere Relation „ist ein“ eingeführt werden. Dies erscheint unnatürlich und ist auch sehr aufwendig.

Zusammenfassend erscheint uns das objektorientierte Modell als treuere Realisierung unseres Entity-Relationship Modells, auf dem unsere Datenwelt basiert. Allerdings gibt es bisher keine Standardisierung und keine so weite Verbreitung wie bei den relationalen Datenbanken.

Die Probleme, die bei der Speicherung von RDF in Datenbanken auftauchen, sind in beiden Architekturen analog. Aus diesem Grund beschränken wir uns im weiteren auf die relationale Architektur.

RDF in Datenbanken

Das große Problem bei der Speicherung von RDF in Datenbanken besteht darin, dass vor dem Lesen des RDF Datensatzes im Allgemeinen nicht bekannt ist, welche Attribute und Objekte in diesem Datensatz auftauchen.

Im relationalen Datenbankschema würde das bedeuten, dass eine Tabelle sich verändern oder neu entstehen muss, wenn ein Dokument mit bisher nicht verwendeten Attributen gespeichert werden soll. Dies ist aber nicht effektiv möglich, da dann ja auch die Schnittstellen zur Eingabe und Abfrage der Datenbank geändert werden müssten.

Im Internet werden in der RDF-special-interest-group [4] verschiedene Ansätze zur Speicherung von RDF in relationalen Datenbanken diskutiert. Zwei dieser Ansätze werden im folgenden vorgestellt und diskutiert. Eine technische Zusammenfassung der Diskussion findet sich unter [3].

Der naive Zugang (Eric Miller)

Der erste Vorschlag basiert auf der Idee, dass RDF sich grundsätzlich in Tripeln darstellen lässt. Alle Tripel werden in einer Tabelle gespeichert. Die Konfiguration sieht folgendermaßen aus:

```
CREATE TABLE triple (  
    property      varchar(255),  
    resource      varchar(255),  
    value         blob  
    hint          char(1));
```

Properties und Ressourcen werden in Strings abgespeichert. Die Objekte werden als binary large objects (blob) gespeichert. Dies ermöglicht auch die Speicherung von binären Daten. Durch das Attribute hint, das aus einem Buchstaben besteht, ist es möglich kenntlich zu machen, ob in einem der Felder genetic ids gespeichert sind, oder ob es sich bei dem Objekt um eine Ressource oder ein Literal handelt.

Dieser Zugang liefert eine korrekte Lösung des Problems, allerdings ist es in dieser Konfiguration nicht möglich nach oder in Objekten zu suchen: Ist beispielsweise der Autor „Erwin“ eines Dokumentes gespeichert, so ist sein Name ein Objekt, das als blob abgelegt ist. In blob Daten kann aber nicht gesucht werden, die Anfrage „welche Bücher hat Erwin geschrieben“ kann von der Datenbank nicht beantwortet werden. Dies ist eine große Einschränkung. Das Problem lässt sich aber leicht umgehen, indem ein weiteres Attribut eingeführt wird, das binäre Literals abspeichert, in denen dann nicht gesucht werden kann, und das Feld value ebenfalls als String deklariert wird, um so eine Suche zuzulassen.

Ein weiteres Problem bei der Suche taucht auf, das viel erheblicher und nicht so einfach zu lösen ist: Wird nach einem konkreten Statement gesucht, müssen im schlimmsten Fall alle Datensätze angeschaut werden. Dies ist sicherlich wenig effektiv und im Vergleich zur Speicherung von RDF im Dateisystem nicht wesentlich schneller.

Der spezifikationstreue Ansatz (Jonas Liljegen)

Ein weiterer Schwachpunkt im Vorschlag von Eric Miller liegt bei der Reifizierung. Wird auf ein Statement verwiesen, so muss das Statement in vier Tripel aufgelöst werden, damit der Verweis auf dieses Statement realisiert werden kann. Dies entspricht zwar der RDF Spezifikation, ist aber für eine effiziente Speicherung der Daten völlig ungeeignet. Der große Unterschied liegt jetzt in der Trennung von Statements und Ressourcen:

```
CREATE TABLE statements (  
    id, pred, subj, obj, fact      int);  
  
CREATE TABLE resources (  
    id, isprefix                  int,  
    uri, value, lang              text);  
CREATE TABLE prefix (  
    from, to                      int);
```

Die Tabelle `prefix` dient zur Speicherung von `about_each_prefix` Statements. Dies soll an dieser Stelle nicht diskutiert werden.

Die Tabelle `statements` enthält jetzt nicht mehr die eigentlichen Daten, sondern nur noch die ids der entsprechenden Ressourcen. Subject, Predicate und Object eines RDF Statements werden jeweils als Ressource abgespeichert. Zusätzlich wird jedes Statement durch seine id zu einer Ressource, in dem ein lokaler Namespace genutzt und die id angehängt wird (`URI=local#id`). Im Feld `fact` kann durch eine Ziffer gekennzeichnet werden, ob das jeweilige Statement als Ressource verwendet wird (`fact=0`).

Die ids der Statements und der Ressourcen werden eindeutig generiert, d.h. wenn eine id bereits für eine Ressource vergeben wurde, kann es kein Statement mehr mit derselben id geben.

In der Tabelle `resources` werden die eigentlichen Daten gespeichert. Handelt es sich dabei um einen Literal, so wird nicht das Feld `uri` belegt, sondern das Feld `value`. Zusätzlich wird das Language Attribut abgespeichert.

Durch die effektivere Speicherung von reifizierten Statements liegt der Suchaufwand bei diesem Ansatz unter dem Suchaufwand im Vorschlag von Eric Miller, allerdings ist er noch immer sehr hoch.

Zusammenfassung

Speicherung von RDF in relationalen Datenbanken ist ohne Verluste möglich. Allerdings ist die Suche nach vorgegebenen Attributen, z.B. nach einem Autorennamen sehr zeitaufwendig. Dies liegt daran, dass es keine ausgezeichneten Tabellen für spezielle Attribute gibt.

Allgemeine Strategie sollte es ist in einem konkreten Datenbankschema daher sein, die Tripel wie oben beschrieben abzuspeichern, allerdings für häufig genutzte Attribute (Autor, Titel) spezielle Tabellen anzulegen. Das führt zu einer erheblichen Beschleunigung der Suchabfragen, wobei die Information noch immer vollständig gespeichert ist.

Alternativ ist es auch möglich, die RDF Daten im Filesystem zu speichern und nur für ausgewählte Attribute Tabellen in einem Datenbankschema anzulegen. Das jeweilige Vorgehen hängt dabei wesentlich von der konkreten Anwendung ab.

Zu berücksichtigen ist dabei natürlich auch die Art und Weise in der Nutzer auf die Datenbank zugreifen können sollen.

Weitere Ansätze zur Speicherung von RDF

Neben den Ansätzen zur Speicherung von RDF in Datenbanken, gibt es Modelle zur Speicherung, die auf der Idee basieren, dass RDF in XML darstellbar ist. XML selbst ist eine Baumstruktur, auf der Suche effektiv möglich ist.

Leider gibt es bei diesem Vorgehen zwei Probleme: Erstens setzen viele der Systeme voraus, dass eine DTD für die zu verwaltenden XML Dokumente vorliegt. Dadurch können beliebige XML Dokumente nicht angemessen verarbeitet werden. Zweitens wird durch die Reduktion auf XML die Graph Struktur von RDF nicht vollständig widerspiegelt.

Beispiele für diesen Ansatz sind Lore [5] von der Stanford University oder das in Entwicklung befindliche System DesIRe [6] der Universität Dortmund.

Literatur

1. Heuer, A. und Saake, G.: *Datenbanken - Konzepte und Sprachen*. International Thompson Publishing, 1. korrigierter Nachdruck, 1997.
2. Kemper, A. und Eickler, A.: *Datenbanksysteme - Eine Einführung*. Oldenbourg, 2. Aktualisierte Auflage, 1997.
3. Sergey Melnik: Storing RDF in a relational database. Request for comments, <http://www-db.stanford.edu/~melnik/rdf/db.html>.
4. RDF Interest Group: <http://www.w3.org/RDF/Interest/>.
5. Lore: <http://lists.w3.org/Archives/Public/www-rdf-interest/2000May/att-0125/01-data.html>
6. DesIRe: <http://ls6-www.cs.uni-dortmund.de/ir/projects/DesIRe/>