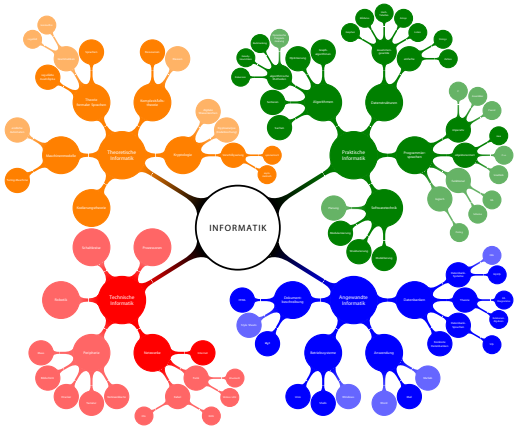
UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR
THEORETISCHE INFORMATIK

Kapitel 41

Fallstudie zu Datenbanken

Vom E/R-Modell zum Java-Code

Vorlesung Einführung in die Informatik 1 vom 24. Juni 2014 von Till Tantau



Lernziele von Kapitel 41

1. Datenbanken für eigene Programmentwicklungen nutzen können
2. Daten aus verschiedenen Quellen in Datenbanken einfügen können
3. Datenbanken zur Daten-Analyse einsetzen können.

Gliederung von Kapitel 41

Ein Programm für die Bibliometrie.

Unser heutiges Ziel ist es, ein Programm zu entwickeln, das bibliometrische Fragen beantworten kann wie:

- Wie hoch ist der Impact-Factor einer Zeitschrift?
- Wie hoch ist der Hirsch-Index eines Autors?
- Welche Zitierkartelle gibt es?

Dazu wird wie folgt vorgegangen:

- Wir modellieren zunächst die Daten (also Autoren, Artikel, Zeitschriften, Zitationen).
- Wir entwickeln ein Programm, das diese Daten aus einer Datenbank lesen und in sie schreiben kann.
- Wir entwickeln Algorithmen, die diese bibliometrischen Fragen beantworten.

Ein kleines »Pflichtenheft«.

Das Programm soll konkret Folgendes leisten können:

1. Das Programm soll bibliographische Daten zu Zeitschriftenartikeln verwalten.
2. Die Daten sollen in der Datenbank `biblio` gespeichert werden.
3. Neue Einträge sollen sich in die Datenbank aus einer anderen Quelle importieren lassen.
4. Berechnung des Impact-Factors einer Zeitschrift.
5. Berechnung des Hirsch-Index von Autoren.

Ein einfaches E/R-Modell für bibliometrische Fragen.

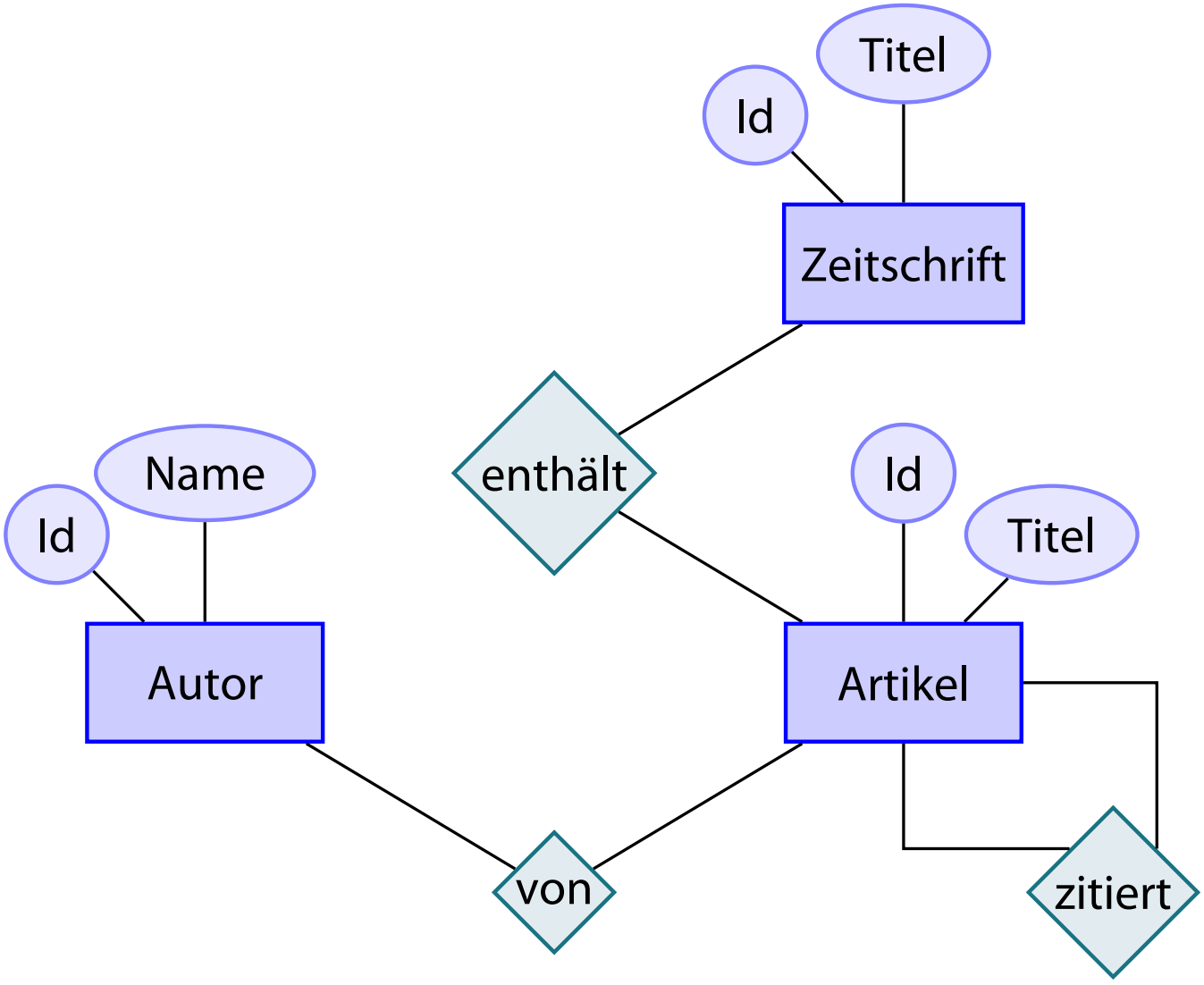


Abbildung des E/R-Modells in SQL.

Wie in einem vorigen Kapitel erklärt, wird das E/R-Modell in der Datenbank wie folgt abgebildet:

- Zu jedem Entitätstyp wird eine Tabelle erstellt.
- Zu jedem Relationshiptyp wird eine Tabelle erstellt.

Die Erstellung der Datenbank und Tabellen erfolgt *nur einmal*. Wir könnten dazu also beispielsweise ein Installationsskript benutzen:

```
echo Hallo, ich bin das Installationsskript.
echo
echo Ich richtet jetzt (einmalig) die Datenbank ein.
echo Als Parameter müssen der Nutzername und das Passwort
echo für die Datenbank übergeben werden

mysql --user=$1 --pass=$2 < db_installer.sql
```

Der Inhalt des »SQL-Skripts«.


```
-- Dieses Skript erstellt die Datenbank biblio
-- und legt die Tabellen an

create database biblio;
use biblio;

create table Autor       (id integer, name  varchar(100));
create table Artikel     (id integer, titel varchar(200));
create table Zeitschrift (id integer, titel varchar(200));

create table von         (autor_id        integer,
                          artikel_id      integer);
create table enthaelt    (artikel_id      integer,
                          zeitschrift_id  integer);
create table zitiert     (zitiert_id      integer,
                          wird_zitiert_id integer);
```

Abbildung des E/R-Modells in Java.

Aus Sicht von Java, siehe auch Info A, gibt es zu jedem Entitätstyp eine Klasse:

```
class Autor {
  int id;
  String name;
}
```

```
class Zeitschrift {
  int id;
  String titel;
}
```

```
class Artikel {
  int id;
  String titel;
}
```

```
class Datenbestand {
  Autor[]       alleAutoren;
  Artikel[]     alleArtikel;
  Zeitschrift[] alleZeitschriften;
}
```

Einlesen der Daten in der Datenbank in Java.

Schritt 1: Herstellen der Verbindung.

- Unser Programm muss (eventuell wiederholt) Daten (Autoren, Artikel) aus der Datenbank lesen.
- Dazu muss als erstes eine Verbindung zur Datenbank hergestellt werden,
- wofür wir eine eigene Methode benutzen, die am Anfang einmal aufgerufen wird.

```
import java.sql.*;

class Datenbestand {
  ...
  Connection c;
  Statement s;

  void createConnection (String user, String password) {
    Class.forName("com.mysql.jdbc.Driver");
    c = DriverManager.getConnection
      ("jdbc:mysql://localhost/biblio", user, password);
    s = c.createStatement ();
  }
}
```

Einlesen der Daten in der Datenbank in Java.

Schritt 2: Einlesen aller Autoren.

Zum Einlesen der Autoren, benötigt man zunächst deren Anzahl.
Dies lässt sich mit einem SQL-Befehl erledigen:

```
class Datenbestand {
  ...
  ResultSet results;

  void readAllAuthors () {
    s.executeQuery ("select␣count(*)␣as␣n␣from␣autor");
    results = s.getResultSet ();
    results.next();
    int n = results.getInt("n");
```

Einlesen der Daten in der Datenbank in Java.

Schritt 2: Einlesen aller Autoren.

Nun kann man den Array erstellen. . .

```
alleAutoren = new Autor [n];
```

. . . und füllen:

```
    s.executeQuery ("select␣*␣from␣autor");
    results = s.getResultSet ();
    int i = 0;
    while (results.next()) {
      alleAutoren[i] = new Autor ();
      alleAutoren[i].id = results.getInt("id");
      alleAutoren[i].name = results.getString("name");
      i++;
    }
  }
}
```

Der Code für die Artikel und die Zeitschriften sieht ganz analog aus, siehe Skript.

Einlesen der Relationships.

Die *Relationships* muss man anders behandeln, *wenn man sie überhaupt einlesen möchte*.
Betrachten wir beispielsweise die `enthaelt`-Tabelle:

- Dieser entspricht *nicht* in natürlicher Weise in Java eine Klasse

```
class Enthaelt {
  // So nicht!
  int zeitschriftId;
  int artikelId;
}
```

- Vielmehr speichert diese Tabelle *ein Attribut des Artikels*:

```
class Artikel {
  int id;
  String title;
  // Was die enthaelt-Tabelle eigentlich speichert:
  Zeitschrift erschienenIn;
}
```

- Wenn man also die Tabelle `enthaelt` liest, muss man für jeden Artikel das passende Zeitschriften-Objekt anhand seiner Id heraussuchen.

Wie bekommt man Daten in die Datenbank?

Es gibt mehrere Arten, wie man Artikel mittels des Programms in die Datenbank einspeisen könnte:

1. Man könnte die Möglichkeit schaffen, in einer Maske Daten einzutragen (so wie es viele bei den Miniprojekten gemacht haben). Das ist aber nicht praktikabel, wenn man große Datenmengen hat.
2. Man könnte die Daten direkt aus einer anderen Datenbank lesen.
3. *Man könnte die Daten direkt aus einer Datei lesen, in der sie in einem einfachen Format stehen.*

Mögliche Formate für Artikel.

Informationen über Artikel liegen typischerweise in verschiedenen Formaten vor:

- Als BibTEX-Eintrag:

```
@book{Codd:1990:RMD:77708,
  author = {Codd, E. F.},
  title = {The relational model for database management:
       version 2},
  year = {1990},
  isbn = {0-201-14192-2},
  publisher = {Addison-Wesley Longman Publishing Co., Inc.},
  address = {Boston, MA, USA},
}
```

Um so etwas in Java zu verarbeiten,
bieten sich *reguläre Ausdrücke* an.

Mögliche Formate für Artikel.

- Als XML-Eintrag:

```
<RDF>
  <description about="2879" publish="false">
    <creator type="AUTHOR">Johannes Textor</creator>
    <creator type="AUTHOR">Antonio Peixoto</creator>
    <creator type="AUTHOR">Sarah E. Henrickson</creator>
    <creator type="AUTHOR">Mathieu Sinn</creator>
    <creator type="AUTHOR">Ulrich H. von Andrian</creator>
    <creator type="AUTHOR">Jürgen Westermann</creator>
    <date type="CREATED" scheme="W3C_DTF">2011</date>
    <description type="BIBLIOGRAPHIC_CITATION"
         scheme="OPEN_URL">
      <article>
        <jtitle>PNAS</jtitle>
      </article>
    </description>
    <relation type="IS_PART_OF">Publikationen TCS</relation>
    <title>Defining the Quantitative Limits of Intravital
         Two-Photon Lymphocyte Tracking</title>
    <type>ARTICLE</type>
  </description>
</RDF>
```

Um so etwas in Java zu verarbeiten, *benutzt man eine Java-Bibliothek für die Verarbeitung von XML-Dokumente*.

Verarbeitung eines BibTEX-Eintrags mit regulären Ausdrücken.

Wir wollen eine Methode schreiben, die

- einen BibTEX-Eintrag eines Artikels als String als Eingabe erhält,
- hieraus mit regulären Ausdrücken die Autoren, den Titel und die Zeitschrift bestimmt und
- in der Datenbank entsprechende Einträge anlegt.

Bestimmung des Titels: Die Idee

Der *Titel-Eintrag* ist in BibTEX wie folgt aufgebaut:

- Er beginnt mit `titel`.
- Dann können Leerzeichen kommen, dann ein Gleichheitszeichen, dann wieder Leerzeichen, dann eine öffnende geschweifte Klammer.
- Dann kommt der eigentliche Titel.
- Dann kommt wieder eine geschweifte Klammer.

Als regulärer Ausdruck ergibt sich:

```
title *= *\{(.*?)\}
```

Bemerkungen:

- Die runden Klammern in der Mitte sind nötig, damit man sich in Java »darauf beziehen kann«.
- Das Fragezeichen sorgt dafür, dass `.*` »möglichst wenig liest«. Lässt man das Fragezeichen weg, würde der Titel alles enthalten bis zur letzten schließenden Klammer im BibTEX-Eintrag.

Schritt 1: Bestimmung des Titels und Anlegen eines neuen Datenbankeintrags.


```
void insertArticleIntoDatabse (String bibtexEintrag) {
  // Extrahiere den Artikelname:
  Pattern p = Pattern.compile("title␣*=␣*\\{(.*?)\\}");
  Matcher m = p.matcher(bibtexEintrag);
  m.find ();
  String title= m.group (1);

  // Hole neue ID:
  s.executeQuery ("select␣max(id)␣as␣m␣from␣artikel");
  results = s.getResultSet ();
  results.next();
  int newId = results.getInt("m") + 1;

  // Schreibe neuen Eintrag:
  s.execute ("insert␣into␣artikel␣values␣("
             + newId + ",␣\"" + title + "\")");
  ...
```

Schritt 2: Bestimmung der Autoren.

Idee

Der *Autoren-Eintrag* ist in BibTEX wie folgt aufgebaut:

- Er beginnt mit `autor`, gefolgt vom Gleichheitszeichen und einer geschweiften Klammer, wie beim Titel.
- Gibt es mehrere Autoren, so sind diese mittels dem Wort `and` getrennt.
- Am Ende kommt wieder eine geschweifte Klammer.

Wir brauchen zwei regulärer Ausdrücke:

1. Um den Anfang der Autoren zu finden:

```
author *= *\{
```

2. Um den nächsten Autor zu finden:

```
(.*?)( and |\})
```

Schritt 2: Bestimmung der Autoren.

Programmtext


```
  ... // Fortsetzung von insertArticleIntoDatabse

  // Finde nun alle Autoren:
  Pattern p1 = Pattern.compile("author␣*=␣*\\{");
  Matcher m1 = p1.matcher(bibtexEintrag);
  m1.find ();
  // Ok, Start-Position der Autoren gefunden...

  Pattern p2 = Pattern.compile("(.*?)(␣and␣|\\})");
  Matcher m2 = p2.matcher(bibtexEintrag);
  m2.find (m1.end()); // Suche ab gefundener Position.

  while (!m2.group(2).equals("}")) {
    connectAuthorWithArticle (m2.group(1), newId);
    m2.find();
  }
  connectAuthorWithArticle (m2.group(1), newId);
}
```

Schritt 3: Einfügen und Verknüpfen der Autoren.

Idee

- Wir wissen nun, welche Autoren ein Paper hat und wir kennen die Id des Papers.
- Der Autor kann schon in der Datenbank sein oder er muss neu angelegt werden.
- Dann muss in der `von`-Tabelle ein Eintrag hinzugefügt werden zwischen dem Autor und dem Artikel.

Schritt 3: Einfügen und Verknüpfen der Autoren.

Programmtext


```
void connectAuthorWithArticle (String author, int articleId) {
  int authorId;

  // Existiert Autor bereits?
  s.executeQuery ("select id from autor where name = \""
                  + author + "\"");
  results = s.getResultSet ();
  if (results.next()) {
    authorId = results.getInt("id");
  } else {
    // Hole neue ID:
    s.executeQuery ("select max(id) as m from autor");
    results = s.getResultSet ();
    results.next();
    authorId = results.getInt("m") + 1;
    // Schreibe neuen Eintrag:
    s.execute ("insert into autor values ("
               + authorId + ", \"" + author + "\")");
  }

  // Schreibe die Verbindung in die Datenbank:
  s.execute ("insert into von values ("
             + authorId + ", " + articleId + ")");
}
```

Der Impact-Factor einer Zeitschrift

- Der Impact-Factor ist ein (höchst umstrittenes) Maß, wie gut eine Zeitschrift ist.
- Gemessen wird die durchschnittliche Anzahl an Zitaten der Artikel in der Zeitschrift innerhalb von zwei Jahren.

Zur Diskussion

Nennen Sie Gründe, die gegen die Benutzung von Impact-Factors sprechen.

- Im Folgenden soll das Problem etwas vereinfacht werden, indem auf den Zwei-Jahres-Zeitraum verzichtet wird.
- Wir wollen also zählen, wie viele Zitate die Artikel einer Zeitschrift überhaupt haben relativ zur Anzahl der Artikel der Zeitschrift.
- Dies lässt sich sowohl innerhalb des Javaprogramms wie auch durch eine geschickte SQL-Anfrage lösen.

Die SQL-Anfragen

Zählen der Artikel in der Zeitschrift mit der Id 42:

```
select count(*) from artikel, enthaelt
 where artikel.id      = enthaelt.artikel_id and
       zeitschrift_id = 42
```

Zählen der Zitate von Artikeln in der Zeitschrift mit der Id 42:

```
select count(*) from artikel, enthaelt, zitiert
 where artikel.id      = enthaelt.artikel_id and
       zeitschrift_id = 42 and
       artikel.id      = zitiert.wird_zitiert_id;
```

Der Programmtext.


```
double computeImpactFactor (int zeitschriftId) {
  s.executeQuery
    ("select␣count(*)␣as␣n␣from␣artikel,␣enthaelt␣" +
      "where␣artikel.id␣␣␣␣=␣enthaelt.artikel_id␣and␣" +
            "zeitschrift_id␣=␣" + zeitschriftId);
  results = s.getResultSet ();
  results.next();
  double n = results.getInt("n");

  s.executeQuery
    ("select␣count(*)␣as␣z␣from␣artikel,␣enthaelt,␣
        zitiert␣" +
      "where␣artikel.id␣␣␣␣=␣enthaelt.artikel_id␣and␣" +
            "zeitschrift_id␣=␣" + zeitschriftId + "␣and␣"
                +
            "artikel.id␣␣␣␣=␣zitiert.wird_zitiert_id");
  results = s.getResultSet ();
  results.next();
  double z = results.getInt("z");

  return z / n;
}
```

Zusammenfassung

1. Die in Java-Objekten gespeicherten Informationen lassen sich gut in relationalen Datenbanken ablegen:

Java	SQL
Klasse	Entitäts-Tabelle
Objekt	Tabellen-Zeile
Basis-Attribut	Tabellen-Attribut
Verweis auf anderes Objekt	Zeile in Relationship-Tabelle

2. Mit regulären Ausdrücken lassen sich Formate gut zerlegen:

```
Pattern p = Pattern.compile("title␣*=␣*\\{(.*?)\\}");
Matcher m = p.matcher(bibtexEintrag);
m.find ();
return m.group (1);
```

3. Analysen von Daten bestehen aus einer Mischung von Java- und SQL-Code:

```
s.executeQuery (...);                // SQL
double n = results.getInt("n"); // SQL nach Java
double z = results.getInt("z");
return z / n;                        // Java
```