RÜCKWÄRTSSALTO

Adaresh Soni,Dominik Strasser

4CHITM

Inhalt

RÜCKWÄRTSSALTO

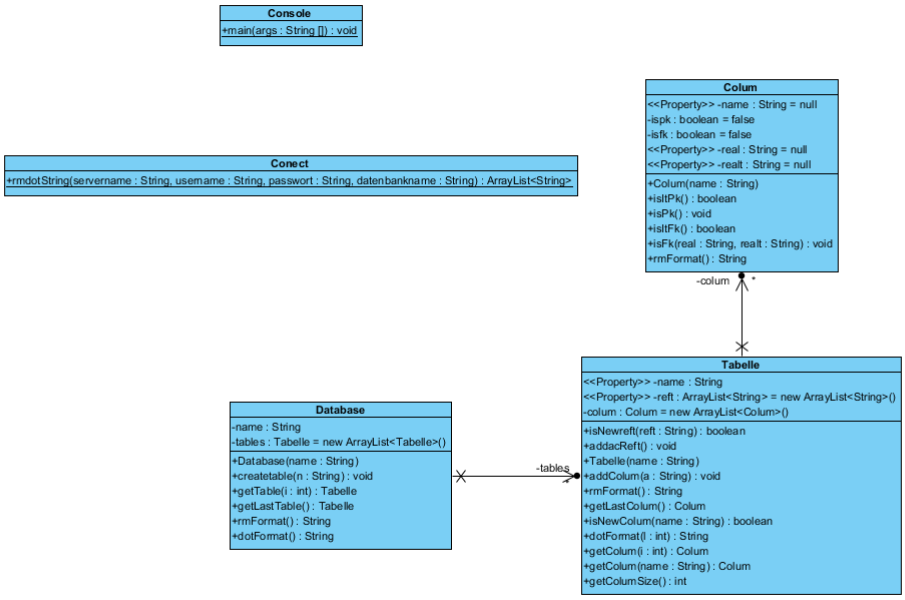
Aufgabenstellung

Erstelle ein Java-Programm, dass Connection-Parameter und einen Datenbanknamen auf der Kommandozeile entgegennimmt und die Struktur der Datenbank als EER-Diagramm und Relationenmodell ausgibt (in Dateien geeigneten Formats, also z.B. PNG für das EER und TXT für das RM)
Verwende dazu u.A. das ResultSetMetaData-Interface, das Methoden zur Bestimmung von Metadaten zur Verfügung stellt.
Zum Zeichnen des EER-Diagramms kann eine beliebige Technik eingesetzt werden für die Java-Bibliotheken zur Verfügung stehen: Swing, HTML5, eine WebAPI, ... Externe Programme dürfen nur soweit verwendet werden, als sich diese plattformunabhängig auf gleiche Weise ohne Aufwand (sowohl technisch als auch lizenzrechtlich!) einfach nutzen lassen. (also z.B. ein Visio-File generieren ist nicht ok, SVG ist ok, da für alle Plattformen geeignete Werkzeuge zur Verfügung stehen)
Recherchiere dafür im Internet nach geeigneten Werkzeugen.
Die Extraktion der Metadaten aus der DB muss mit Java und JDBC erfolgen.
Im EER müssen zumindest vorhanden sein:
korrekte Syntax nach Chen, MinMax oder IDEFIX
alle Tabellen der Datenbank als Entitäten
alle Datenfelder der Tabellen als Attribute
Primärschlüssel der Datenbanken entsprechend gekennzeichnet
Beziehungen zwischen den Tabellen inklusive Kardinalitäten soweit durch Fremdschlüssel nachvollziehbar. Sind mehrere Interpretationen möglich, so ist nur ein (beliebiger) Fall umzusetzen: 1:n, 1:n schwach, 1:1
Kardinalitäten Fortgeschritten (auch einzelne Punkte davon für Bonuspunkte umsetzbar)
Zusatzattribute wie UNIQUE oder NOT NULL werden beim Attributnamen dazugeschrieben, sofern diese nicht schon durch eine andere Darstellung ableitbar sind (1:1 resultiert ja in einem UNIQUE)
optimierte Beziehungen z.B. zwei schwache Beziehungen zu einer m:n zusammenfassen (ev. mit Attributen)
Erkennung von Sub/Supertyp-Beziehungen.

Designüberlegung

Nach längerem überlegen entschieden wir uns das wir eine Datenbank sehr simpel in Java Abbilden wollen. Dadurch wurde sobald wir die Datenbank gespeichert haben die Verarbeitung der Daten sehr einfach.

UML



Arbeitsaufteilung

Die Arbeitsteilung wurde folgendermaßen festgelegt:

Adaresh Soni und Dominik Strasser: Prototyp - Mittels JDBC zur einer DB verbinden und deren Inhalt mit den Tabellen ausgeben

Adaresh Soni: Implementieren der MÖglichkeit die DB in einem Textfile auszugeben, Implementieren der Möglichkeit ein EER als Bild auszugeben

Dominik Strasser: Dokumentation des Quellcodes, Protokoll dokumentieren

Aufwandsabschätzung

Geschätzt haben wir für den ersten Teil (Metadaten rauslesen und RM erzeugen(Prototypen)) hatten wir auf durchschnittlich auf 6 Stunden geschätzt. Und für den zweiten Teil () hatten wir durchschnittlich auf 40 Stunden geschätzt.

Endzeitaufteilung

Insgesamt brauchten wir 8 Stunden für den ersten Teil da wir anfangs Probleme hatten die Metadaten richtig aus zu lesen. Für den zweiten Teil kamen wir gut mit der geschätzten Zeit von 40 Stunden aus.

Arbeitsdurchführung

Als Grundlage wurde der Exporter () verwendet.

Um eine effiziente Arbeitsumgebung zu gewähren wurde ein GitHub Repository erstellt.

Leider wurde er nicht in vollen Zügen genutzt und das Programm wurde trotzdem öfters über mail weiter geleitet. Dies geschah so, weil wir erstens nicht so geläufig mit GitHub sind und zweitens uns zu Ohren gekommen ist das viele Mitschüler über diese open Repositorien das Programm kopieren. Da das wegen den Metaregeln auch uns schaden kann haben wir weil es verlangt wurde nur die wesentlichen Schritte auf GitHub hochgeladen.

Nach dem wir es nach ein paar Problemen geschafft hatten die notwendigen Metadaten auszulesen. Begannen wir daraus die Datenbank als RM zurück zu geben. Die gespeicherten Daten in einem RM Format gerechten String zu speichern und zurück zu geben was nicht sonderlich schwer.

```
while (rstable.next()) {
    rsimport = metadata.getImportedKeys(con.getCatalog(), con.getSchema(), rstable.getString(3));
    rspk = metadata.getPrimaryKeys(con.getCatalog(), con.getSchema(), rstable.getString(3));
    rscol= metadata.getColumns(con.getCatalog(), null, rstable.getString(3), null);
    ret=ret+"\n"+rstable.getString(3)+"(";
    while (rscol.next()) {
    while (rsimport.next()) {
        while (rspk.next()) {
            pk.add(rspk.getString(4));
        }
        fk.add(rsimport.getString(8));
    }
    if(pk.contains(rscol.getString(4))){
        if(fk.contains(rscol.getString(4))){
            ret=ret+"<pk><fk>"+rscol.getString(4)+"</fk></pk>,";
        }else{
            ret=ret+"<pk>"+rscol.getString(4)+"</pk>,";
        }
    }else{
        if(fk.contains(rscol.getString(4))){
            ret=ret+"<fk>"+rscol.getString(4)+"</fk>,";
        }else{
            ret=ret+rscol.getString(4)+",";
        }
    }
}
    ret=ret+")";
```

Danach begannen wir uns für eine passende Bibliothek oder tool umzusehen. Als ersten wollten wir astah nehmen aber nur durch Erwähnung vor dem Herr Prof. Borko wurde uns mitgeteilt das benutzten von astah ein Schuss ins Knie sei. Danach wurden wir durch Mitschüler auf Graphviz aufmerksam.

Graphviz

Graphviz ist ein von AT&T und den Bell-Labs entwickeltes plattformübergreifendes Open-Source-Programmpaket zur Visualisierung von Objekten und deren Beziehungen untereinander. Mathematisch ausgedrückt visualisiert Graphviz gerichtete und ungerichtete Graphen.
Graphviz entnimmt alle zur Erzeugung der Grafik benötigten Anweisungen einer Textdatei, die eine Beschreibung der Knoten und Kanten des Graphen enthält. Die Positionen der einzelnen Knoten sowie die Krümmungen der Kanten werden aus dieser Beschreibung automatisch berechnet und dabei so optimiert, dass die Struktur des Graphen gut erkennbar ist. Zur Beschreibung des darzustellenden Graphen wird die Auszeichnungssprache *DOT* verwendet. Sie ist syntaktisch an die Programmiersprache C angelehnt. Graphviz bietet bei Bedarf auch zusätzliche Möglichkeiten zur Veränderung des Layouts sowie der Form und Farbgebung des Graphen.
Oft genügt allein die Strukturdefinition des Graphen zur Erzeugung einer passablen Ausgabe. Daher können nicht nur Menschen, sondern auch automatische Prozesse Graphviz zur Erstellung von Visualisierungen nutzen. Auch können an vorhandenen Graphen sehr schnell Veränderungen vorgenommen werden, was mit einem Standard-Grafikprogramm nicht ohne weiteres möglich ist.
Graphviz bietet verschiedene Verfahren zur Visualisierung von Graphen an:
dot
Zur Darstellung hierarchischer Strukturen. Alle Kanten verlaufen dabei in etwa in dieselbe Richtung, von oben nach unten oder von links nach rechts. Überschneidungen der Kanten werden möglichst vermieden und die Kantenlänge wird so kurz wie möglich gehalten.
neato und fdp
Visualisiert Graphen im so genannten „spring model“ Layout. Der Startknoten wird mittig angelegt. Neato benutzt dabei den Kamada-Kawai-Algorithmus. Fdp implementiert die Fruchterman-Reingold-Heuristik für größere Graphen.
twopi
Radiales Layout, nach Graham Wills.
circo
Circuläres Layout, nach Six and Tollis.
Der DOT-Quelltext kann Graphviz z. B. als Textdatei über einen Kommandozeilenbefehl übergeben werden. Als Standard erzeugt Graphviz eine Textdatei als DOT-Quelltext in der die Attribute für die Position und Größe der Knoten und Kanten mit angegeben werden. Über Graphviz kann als Ausgabe aber auch eine Bilddatei erstellt werden. Unterstützt werden unter anderem die Dateiformate Postscript, SVG, JPEG, PNG und PDF.

Pro/Con von Graphviz

PRO	CON
Erstellen von Diagrammen sehr einfach	Es ist von Graphviz abhängig
Das Diagramm ist in sehr vielen Formaten wiedergebbar	Man muss die dot Syntax lernen
Graphviz ist gratis und führ alle verfügbar	
Dot Syntax ist sehr einfach und bietet viel Möglichkeiten	

Da es vor allem sehr einfach ist beschlossen wir Graphviz zu nehmen. Allerdings merkten wir schnell das unser jetziger Programm Design sehr unstrukturiert war Dadurch kamen wir auf unser aktuelles Design (siehe Designüberlegung). Nachdem wir auch ein zur Datenbank passendes dot File wiedergeben konnten mussten wir uns noch um das automatische erzeugen des Diagramms kümmern. Das hat uns „GraphViz Java API - Loria“ ermöglicht.

GraphViz Java API ist eine Klasse die für alle zugänglich ist und mit gewissen Angaben ein dotfile in ein Diagramm umwandeln kann.

Quellen

http://de.wikipedia.org/wiki/Graphviz

http://www.loria.fr/~szathmar/off/projects/java/GraphVizAPI/index.php

http://docs.oracle.com/javase/7/docs/api/