**Rückwertssalto**

Inhalt

[Angabe 2](#_Toc411500192)

[Requirementsanalyse 3](#_Toc411500193)

[UML – Diagramm 4](#_Toc411500194)

[GitHub - Repository 5](#_Toc411500195)

## Angabe

Erstelle ein Java-Programm, das 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)

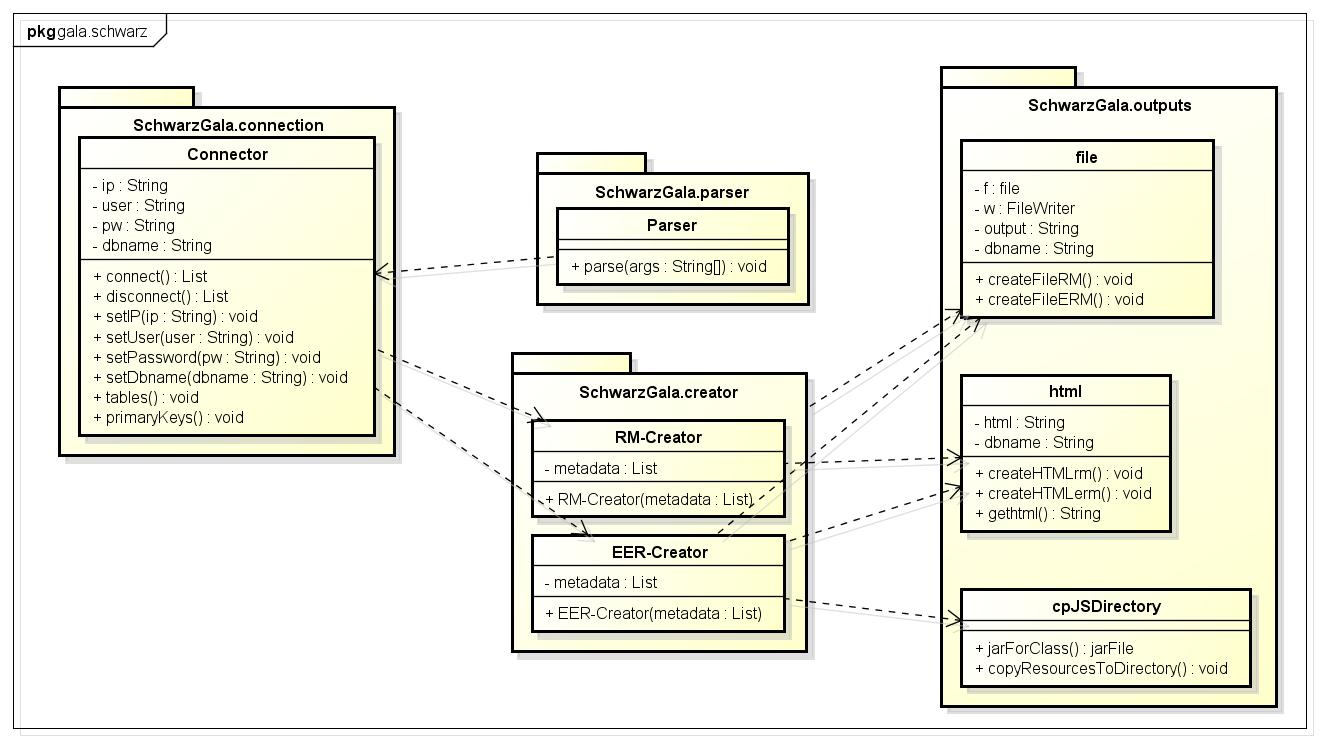
optimierte Beziehungen z.B. zwei schwache Beziehungen zu einer m:n zusammenfassen (ev. mit Attributen)

Erkennung von Sub/Supertyp-Beziehungen

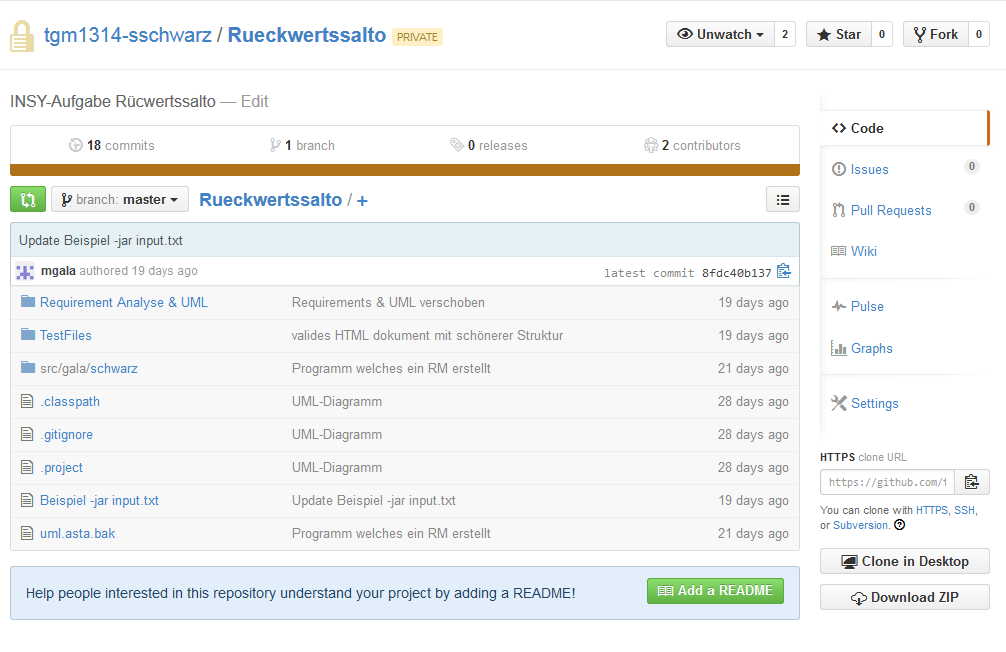
## Requirementsanalyse

|  |  |  |  |
| --- | --- | --- | --- |
| **Requirements** | erledigt | getestet | von |
|
| Verbindung zur Datenbank | ja | ja | G, S |
| Eingabe von Parametern zum Verbinden zur Datenbank | ja | ja | G, S |
| Auslesen von Metadaten mit ResultSetMedaData-Interface | ja | ja | G, S |
| Erstellung eines RM | ja | ja | S |
| korrektes anzeigen von Primary Keys | ja | ja | S |
| korrektes anzeigen von Foreign Keys | ja | ja | S |
| korrektes anzeigen von Beziehungen | ja | ja | G |
| Erstellung eines ERR | ja | ja | S |
| alle Tabellen der Datenbank als Entitäten | ja | ja | S |
| alle Datenfelder der Tabellen als Attribute | ja | ja | S |
| Primärschlüssel der Datenbanken entsprechend gekennzeichnet | ja | ja | S |
| 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 | Ja | ja | S |
| Kardinalitäten | Teilw. |  |  |
| 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 |  |  |  |

## UML – Diagramm



## GitHub - Repository



## Auslesen von MetaDaten

Das Auslesen von MetaDaten erfolgt mittels:

DatabaseMetaData md = con.getMetaData.

Nun kann man die gewünschten MetaDaten in ein mittels eines Result sets auslesen:

resultTables = md.getTables(String catalog[may be null], String schemaPattern[may be null], String tableNamePattern['%' for all], String[] types[may be null]);

Mittels Cursor ließt man dann alles aus:

while (resultTables.next()) {

…

}

Datenbankname:

dbname = resultTables.getString(1);

Alle Tabellenname:

tabnames = resultTables.getString(3);

PK's mittels:

md.getPrimaryKeys("", "", tabnames);

FK's mittels:

md.getImportetKeys(null, null, tabnames);

## Relationship Model

Das RM wird in der Klasse createRM erzeugt. Die Attribute sind "database name, alle tabellen, alle attribute, alle fk's, alle pk's und die tabellen von denen pk's bezogen werden":

rm = new createRM(dbname, tables, att, fk, pk, pktable);

Im RM selbst wird der output für ein html dokument erzeugt. Dafür wird die Klasse html aufgerufen, welche die Struktur eines validen html dokuments erzeugt:

html = new html(true, dbname);

Jeder PK wird mit "<u>PK</u>" gekennzeichnet:

for(int k = 0; k < pk.length; k++){

if(feld[j].equals(pk[k])){

output += "<u>";

pktf = true;

}

}

Jeder FK wir mit "<i>FKTable.FK</i> gekennzeichnet:

for(int k = 0; k < fk.length; k++){

if(feld[j].equals(fk[k])){

output += "<i>" + fkt[k] + ".";

fktf = true;

}

}

Danach werden die restlichen attribute eingefügt:

output += feld[j];

Es wird überprüft ob es das letzte Attribut pro Tabelle war, damit ein "," am Ende vermieden wird.

if(j != feld.length - 1){

output += ", ";

}

Am ende wird die Klasse file aufgerufen, welche mittels dem übergebenen output ein .html file im Verzeichnis des .jar Files erzeugt.

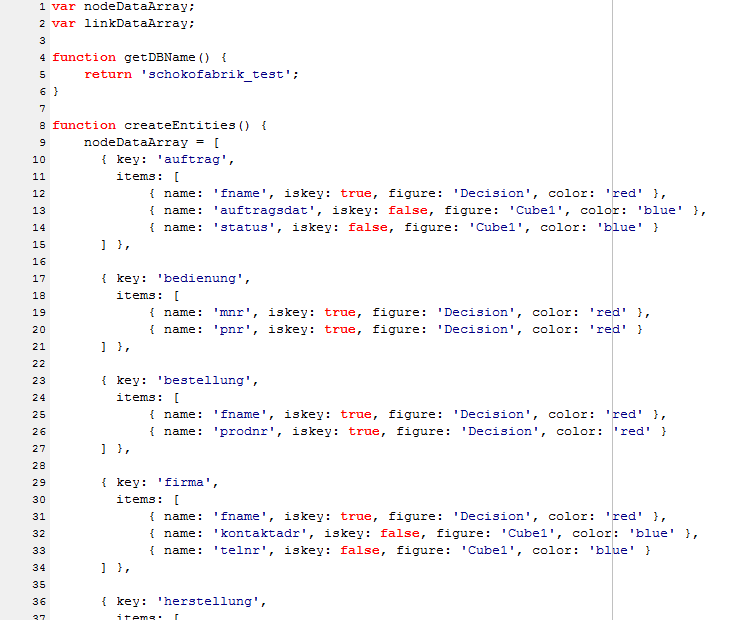
f = new file(true, output, null, dbname);

## Entity Relationship Model

Das ERM wird in der Klasse createERM erzeugt. Die Attribute sind dieselben wie beim RM, nur das noch die tabellen von denen fk's bezogen werden dazu kommen:

erm = new createERM(dbname, tables, att, fk, pk, pktable, fktable);

Im ERM wird der output für ein .js file erzeugt, welches als ERM.js im Verzeichnis /js gespeichert wird. In diesem js file befinden sich in einigen Arrays die Daten, welche die js libary goJS(<http://gojs.net/latest/index.html>, <http://gojs.net/latest/samples/entityRelationship.html>) für ein ERM benötigt:

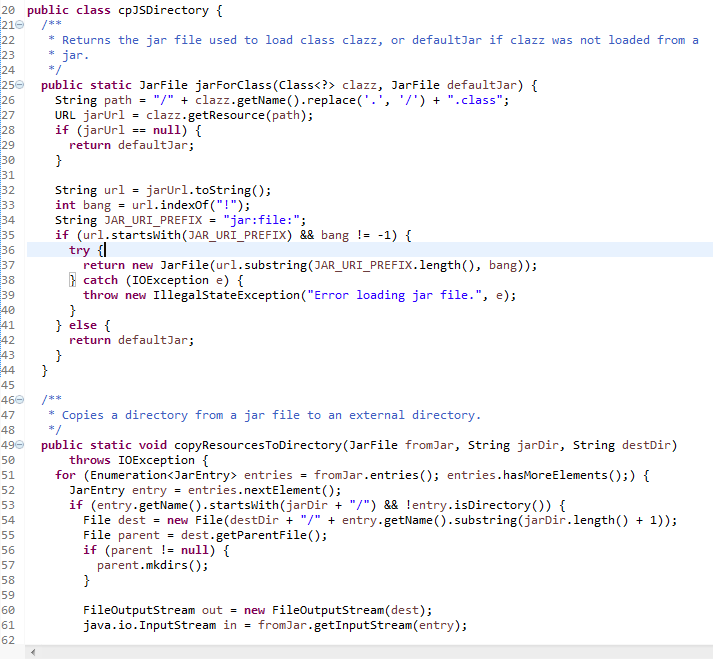


Desweiteren befindet sich eine layout.js und die goJS libary selbst als go-debug.js in dem /js Verzeichnis.

Wie in createRM wird auch hier noch zusätzlich der output für ein .html file an die Klasse file übergeben und dieses wird dann auch von der Klasse file im Verzeichnis des .jar Files erstellt:

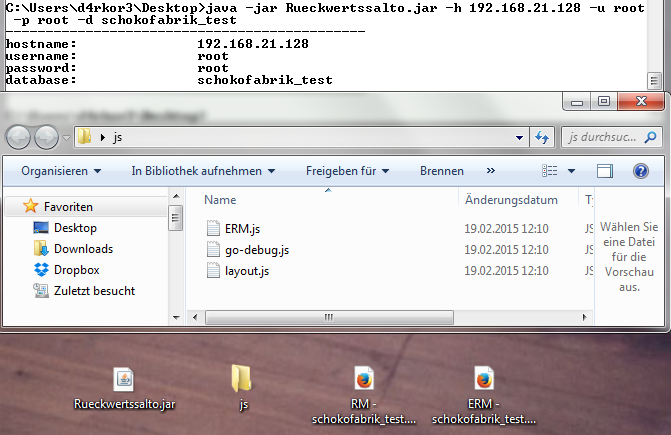
f = new file(false, output, html.gethtml(), dbname);

Das nun enstandene Problem war, dass sich der Ordner js nur im .jar File befinden konnte. Aber somit das .html file die js files nicht importieren konnte, da er nicht über das .jar File darauf zugreifen konnte. Die Lösung war eine Klasse cpJSDirectory zu schreiben, welche das JSDirectory aus dem .jar File in das Verzeichnis des .jar Files exportiert.

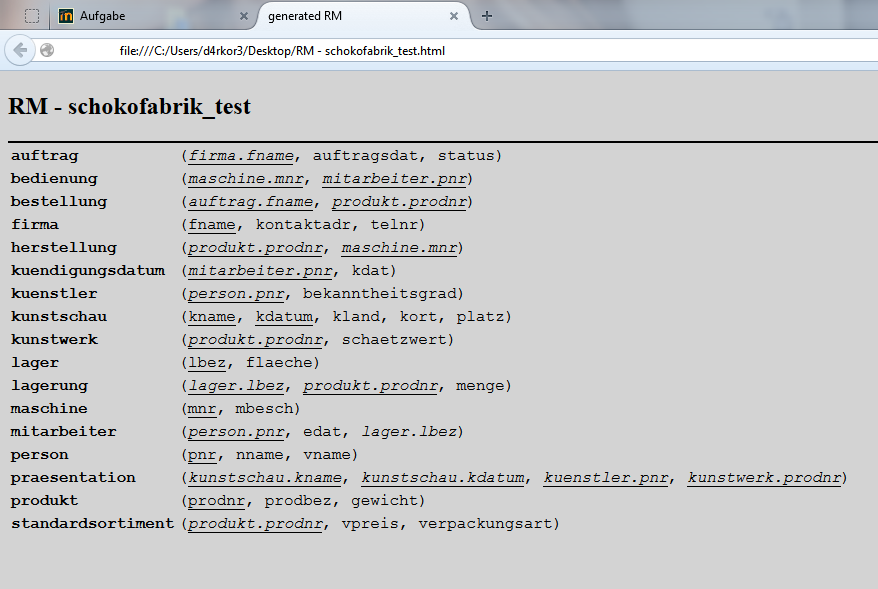
<http://www.java2s.com/Code/Android/File/Copiesadirectoryfromajarfiletoanexternaldirectory.htm>

## Endergebnisse

Folgende Files werden erstellt:



Das RM:



Das ERM:

