INSY – A05 - Rückwärtssalto

Kalauner, Ritter 4AHIT

28.12.2014

Inhaltsverzeichnis

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

GitHub-Repolink

https://github.com/pkalauner-tgm/insy-a05-rueckwaertssalto

Zeitaufzeichnung

Schätzung

Kalauner Paul

Beschreibung	Zeitaufwand (Min.)
Design	30
Auslesen der Metadaten aus der DB	90
Schreiben des RMs in eine Datei	20
Schreiben des ERDs in eine Datei (Pair Programming)	120
Testen	100
Gesamt	360 Minuten

Ritter Mathias

Beschreibung	Zeitaufwand (Min.)
Design	30
Decorator für Attribute	100
Kommandoverarbeitung	30
Schreiben des ERDs in eine Datei (Pair Programming)	120
Testen	100
Gesamt	380 Minuten

Tatsächlich benötigte Zeit

Kalauner Paul

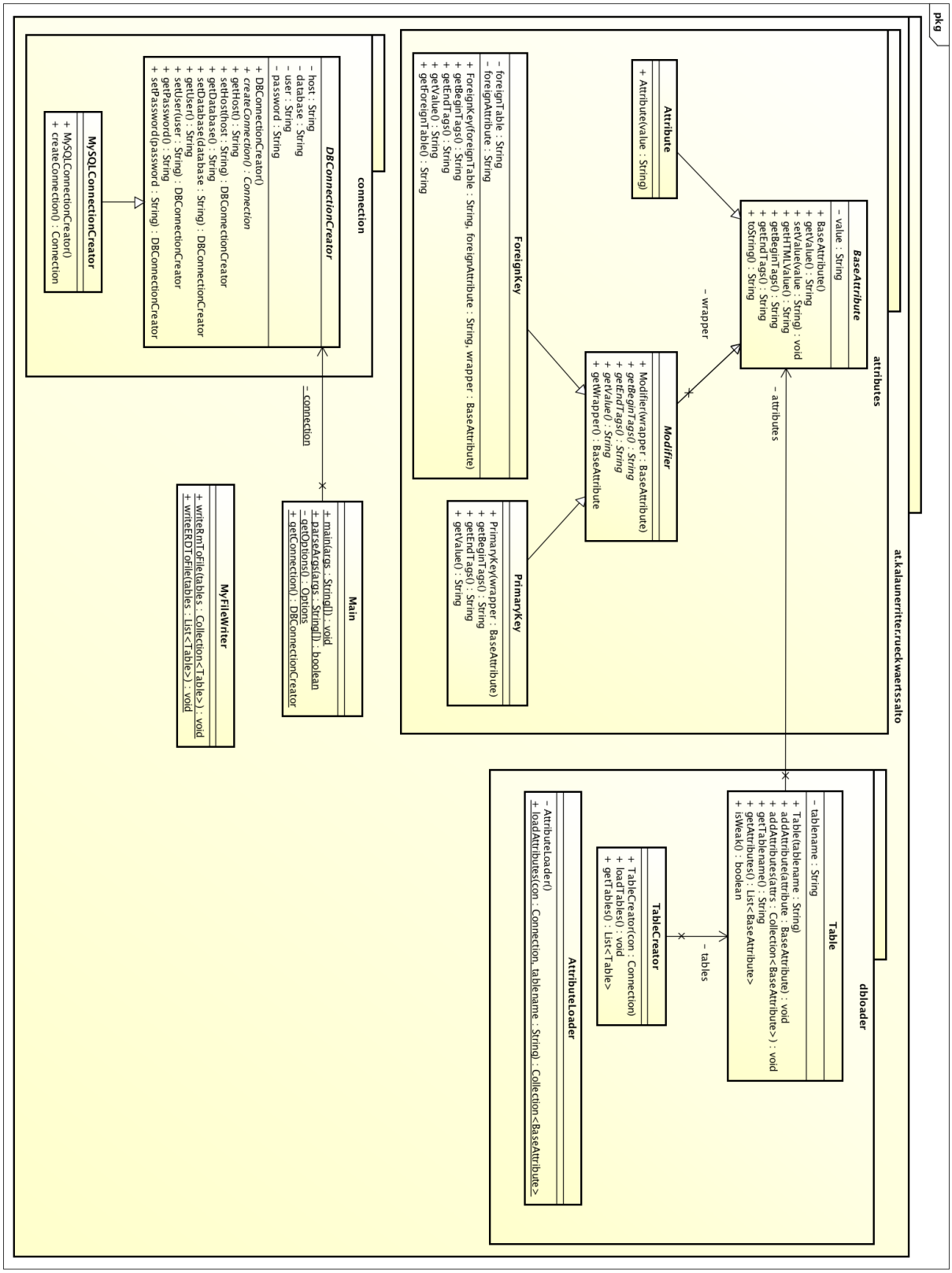
Datum	Beschreibung	Zeitaufwand (Min.)
	Design	50
26.12.2014	Auslesen der Metadaten aus der DB	140
26.12.2014	Schreiben des RMs in eine Datei	10
28.12.2014	Schreiben des ERDs in eine Datei (Pair Programming)	150
28.12.2014	Testen	145
28.12.2014	UML-Redesign	25
	Gesamt	520 Minuten

Ritter Mathias

Datum	Beschreibung	Zeitaufwand (Min.)
26.12.2014	Design	50
26.12.2014	Decorator für Attribute	135
26.12.2014	Kommandoverarbeitung	25
28.12.2014	Schreiben des ERDs in eine Datei (Pair Programming)	150
28.12.2014	Testen	110
28.12.2014	Bearbeiten des Protokolls	75
	Gesamt	545 Minuten

Designüberlegungen

Das Klassendiagramm befindet sich auch zusätzlich als PNG-Datei im Jar-File.



Umsetzung

Laden der Tabellen/Attribute

Zuerst werden die Tabellen inkl. aller Attribute aus der Datenbank ausgelesen. Die Tabellen erhält man über die Meta-Daten einer Connection:

```
this.dbmd = con.getMetaData();
ResultSet rs = dbmd.getTables(null, null, "%", null);
```

Mit rs.next() kann man nun die Tabellen nach der Reihe durchgehen. Zu jeder Tabelle werden alle Attribute hinzugefügt:

```
while (rs.next()) {
        String tablename = rs.getString(3);
        Table t = new Table(tablename);
        t.addAttributes(AttributeLoader.loadAttributes(con, tablename));
}
```

In AttributeLoader.loadAttributes(...) werden die Attribute hinzugefügt. Diese erhält man wieder über die Meta-Daten:

```
ResultSet rs = dbmd.getColumns(null, null, tablename, null);

while (rs.next()) {
        ...
        map.put(columnName, new Attribute(columnName));
}
```

Nun werden die Datenbank-Attribute, wenn es sich um einen Primary-Key und/oder einen Foreign-Key handelt, mit einem Decorator dekoriert.

```
ResultSet pks = dbmd.getPrimaryKeys(con.getCatalog(), null, tablename);
while (pks.next()) {
    ...
    BaseAttribute attr = map.get(columnName);
    map.put(columnName, new PrimaryKey(attr));
}

ResultSet foreignKeys = dbmd.getImportedKeys(con.getCatalog(), null, tablename);
while (foreignKeys.next()) {
    ...
    BaseAttribute attr = map.get(fkColumnName);
    map.put(fkColumnName, new ForeignKey(pkTableName, pkColumnName, attr));
}
```

Der Decorator verfügt über eine Methode getHTMLValue, wodurch die Beginn-HTML-Tags, die Value (Namen) des Datenbank-Attributs und die End-HTML-Tags zurückgegeben werden.

Der Primary-Key-Decorator fügt HTML-Tags für unterstrichenen Text hinzu. Die Value (Namen) des Datenbank-Attributs bleibt gleich:

```
public String getBeginTags() {
   return "<u>" + getWrapper().getBeginTags();
}

public String getEndTags() {
   return  getWrapper().getEndTags() + "</u>";
}

public String getValue() {
   return getWrapper().getValue();
}
```

Der Foreign-Key-Decorator fügt HTML-Tags für kursiven Text hinzu. Die Value (Namen) des Datenbank-Attributs wird ebenfalls geändert.

```
public String getBeginTags() {
   return "<i>" + getWrapper().getBeginTags();
}

public String getEndTags() {
   return getWrapper().getEndTags() + "</i>";
}

public String getValue() {

   //Die Value wird geandert, entweder auf attr4: RelY.attrZ oder nur auf RelY.attrZ
            bei gleichnamigen Attributen
   if (getWrapper().getValue().contains(foreignAttribute))
      return foreignTable + "." + getWrapper().getValue();
   else
      return getWrapper().getValue() + ": " + foreignTable + "." +
            foreignAttribute;
}
```

RM Generierung

Der Decorator dekoriert die Attribute automatisch mit HTML-Tags. Aus diesem Grund kann durch die Tabellen und dessen Attribute durchiteriert werden und in ein File geschrieben werden.

```
for (Table cur : tables) {
    writer.print(cur.getTablename() + " (");
    writer.println(listToString(cur.getAttributes()) + ")<br>");
}
```

ERD Generierung

Die Generierung des ERDs ist etwas aufwendiger. Hierzu haben wir DOT aus dem GraphViz-Paket [1] in Kombination mit einer GraphViz Java API [2], welche aus einem DOT-File eine Bilddatei erzeugt, verwendet.

DOT ist eine Beschreibungssprache für die visuelle Darstellung von Graphen, welche von Rendereren, in unserem Fall von Rendereren des GraphViz-Pakets, interpretiert wird. [3]

Damit die API eine Bilddatei erzeugen kann, muss allerdings das GraphViz Paket installiert sein, ansonsten wird nur ein .dot File generiert. GraphViz ist aber für alle Plattformen kostenlos verfügbar.

Entitäten (Rechteckte) werden in DOT folgendermaßen erzeugt:

```
node [shape=box]; entity1; entity2;
```

In unserem Code sieht dies folgendermaßen aus:

```
gv.add("node [shape=box]; ");
for (Table table : tables) {
    // Falls es sich um eine schwache Entitaet handelt, mit doppeltem Rahmen
darstellen (peripheries=2)
    gv.add(table.getTablename() + (table.isWeak() ? " [peripheries=2]" : "") + ";
");
}
gv.addln();
```

Attribute werden als Ellipse dargestellt:

```
node [shape=ellipse]; ...
```

Code-Ausschnitt:

```
for (Table table : tables) {
    boolean weak = table.isWeak();
    // durch alle Attribute jeder Tabelle iterieren
    for (BaseAttribute attr : table.getAttributes()) {
        if (!(attr instanceof ForeignKey)) {
            // Falls es sich nicht um einen Foreign Key handelt, stelle das Attribut dar
            gv.add("{node [label=<" + attr.getHTMLValue() + ">] " + attr.getValue() + counter + ";};");
            // Tabelle mit Attribut durch einen Strich verbinden
            sbConnections.append(attr.getValue()).append(counter++).append(" --
").append(table.getTablename()).append(";\n");
        } else {
            // Falls es sich schon um einen Foreign Key handelt, erzeuge Relation
            ForeignKey fk = (ForeignKey) attr;
            String relationName = "\"" + table.getTablename() + "-" + fk.getForeignTable() + "\"";
            // Falls es sich um einen Foreign Key handelt, stelle den Rahmen der Beziehung doppelt dar
            sbRelations.append(relationName).append(weak ? " [peripheries=2]" : "").append(";");
            // Relation mit Tabellen verbinden. Hierbei wird ein Set verwendet, um doppelte
Verbindungen zu vermeiden
            // Falls es sich um einen Foreign Key handelt stelle die Verbindung mit doppelten Linien
dar
            fkConnections.add(relationName + " -- " + table.getTablename() + " [label=\"1\",len=1.00" +
(weak ? ",color=\"black:white:black\"" : "") + "];\n");
            fkConnections.add(fk.getForeignTable() + " -- " + relationName + "
[label=\"n\",len=1.00];\n");
        }
    }
}
```

Hier werden auch gleichzeitig die Verbindungen zwischen Entitäten und Attributen bzw. Tabellen und Relationen dargestellt:

```
Entität -- Attribut;
```

Ausführen des Programms

Beim Ausführen des Programms werden in dem Verzeichnis, aus dem das Programm gestartet wurde, alle Dateien abgespeichert. Folgende Dateien werden generiert:

- Relationenmodell
  - rm.html
- ER-Diagramm
  - erd.dot
  - erd.png (siehe Anmerkung unten)

Achtung: Die Datei erd.png wird nur generiert, wenn Graphviz installiert ist. Dieses Programm wird benötigt, um die Datei "erd.dot" in "erd.png" umzuwandeln.

Graphviz ist eine kostenlose Open-Source-Software, welche für alle gängigen Plattformen (Fedora, Rhel, Ubuntu, Solaris, Mac OS, Windows) heruntergeladen werden kann. Der Sourcecode ist ebenfalls erhältlich.

Graphiz kann von der offiziellen Seite unter folgendem Link heruntergeladen werden: http://www.graphviz.org/Download..php

Tests und Testbericht

Die Tests befinden sich in den jeweiligen Packages im JAR-File und sind mittels JavaDoc kommentiert. Der Testbericht (testbericht.html) befindet sich im Root-Verzeichnis des JAR-Files.

Lessons Learned

- Verwendung von JDBC um Metadaten auszulesen

- Verwendung von Dot (GraphViz) [1]

Genaue Beschreibung der hier aufgezählten Punkte siehe oben unter "Umsetzung".

Quellen

[1] GraphViz. Abrufbar unter:
http://www.graphviz.org
[zuletzt abgerufen am 28.12.2014]

[2] GraphViz Java API. Abrufbar unter:
http://www.loria.fr/~szathmar/off/projects/java/GraphVizAPI/index.php
[zuletzt abgerufen am 28.12.2014]

[3] DOT, Wikipedia. Abrufbar unter:
http://de.wikipedia.org/wiki/DOT_(GraphViz)
[zuletzt abgerufen am 28.12.2014]