INSY – A05 - Rückwärtssalto

Kalauner, Ritter 4AHIT

28.01.15

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 & Requirementsanalyse

Must-Have-Anforderungen						
	Name	**Umgesetzt**	**Getestet**	**gesch. Zeit**	**tats. Zeit**	**Datum**
Verarbeitung der CLI Argumente	-	-	-	-	-	-
Einlesen der Argumente vom CLI	Ritter	x	x	10	15	26.12.14
Hilfeausgabe bei falscher Eingabe	Ritter	x	x	10	5	26.12.14
Verbindungsaufbau zur DB laut angegebener Argumente	Ritter	x	x	10	10	26.12.14
Einlesen der Infos aus der DB	-	-	-	-	-	
Einlesen der vorhandenen Tabellen	Kalauner	x	x	20	10	26.12.14
Einlesen der in den Tabellen vorhandenen Attribute	Kalauner	x	x	20	20	26.12.14
Einlesen der Eigenschaften der vorhandenen Attribute	Kalauner	x	x	20	40	26.12.14
Generieren des Relationenmodells	-	-	-	-	-	
Alle Tabellen ins RM schreiben	Kalauner/Ritter	x	x	50	60	28.12.14
Alle Attribute ins RM schreiben	Kalauner/Ritter	x	x	50	60	28.12.14
Primary Key kennzeichnen	Kalauner/Ritter	x	x	15	25	28.12.14
Foreign Key kennzeichnen	Kalauner/Ritter	x	x	15	25	28.12.14
Generieren des ERD	-	-	-	-	-	
Alle Entitäten ins ERD schreiben	Kalauner	x	x	40	45	28.12.14
Schwache Entitäten kennzeichnen	Ritter	x	x	40	30	28.12.14
Alle Attribute ins ERD schreiben	Kalauner	x	x	40	30	28.12.14
Primary Key kennzeichnen	Ritter	x	x	15	20	28.12.14
Foreign Key kennzeichnen	Ritter	x	x	25	35	28.12.14
1:n Beziehungen darstellen	Kalauner	x	x	40	30	07.01.15
1:1 Beziehungen darstellen	Ritter	x	x	40	65	07.01.15

Nice-to-Have-Anforderungen						
UNIQUE im ERD kennzeichnen	Kalauner/Ritter	x	x	30	50	14.01.15
NOT NULL im ERD kennzeichnen	Kalauner/Ritter	x	x	30	50	14.01.15
m:n Beziehungen im ERD kennzeichnen						
Sub/Supertyp-Beziehungen im ERD kennzeichen						

Nicht-Funktionale Anforderungen						
Entwurf des UML-Klassendiagramms	Kalauner/Ritter	x	-	30	50	26.12.14
Einrichtung Maven	Kalauner	x	-	20	30	26.12.14
Einrichtung GIT	Ritter	x	-	10	10	26.12.14
Testen Kalauner	Kalauner	x	-	100	150	14.01.15
Testen Ritter	Ritter	x	-	100	160	14.01.15

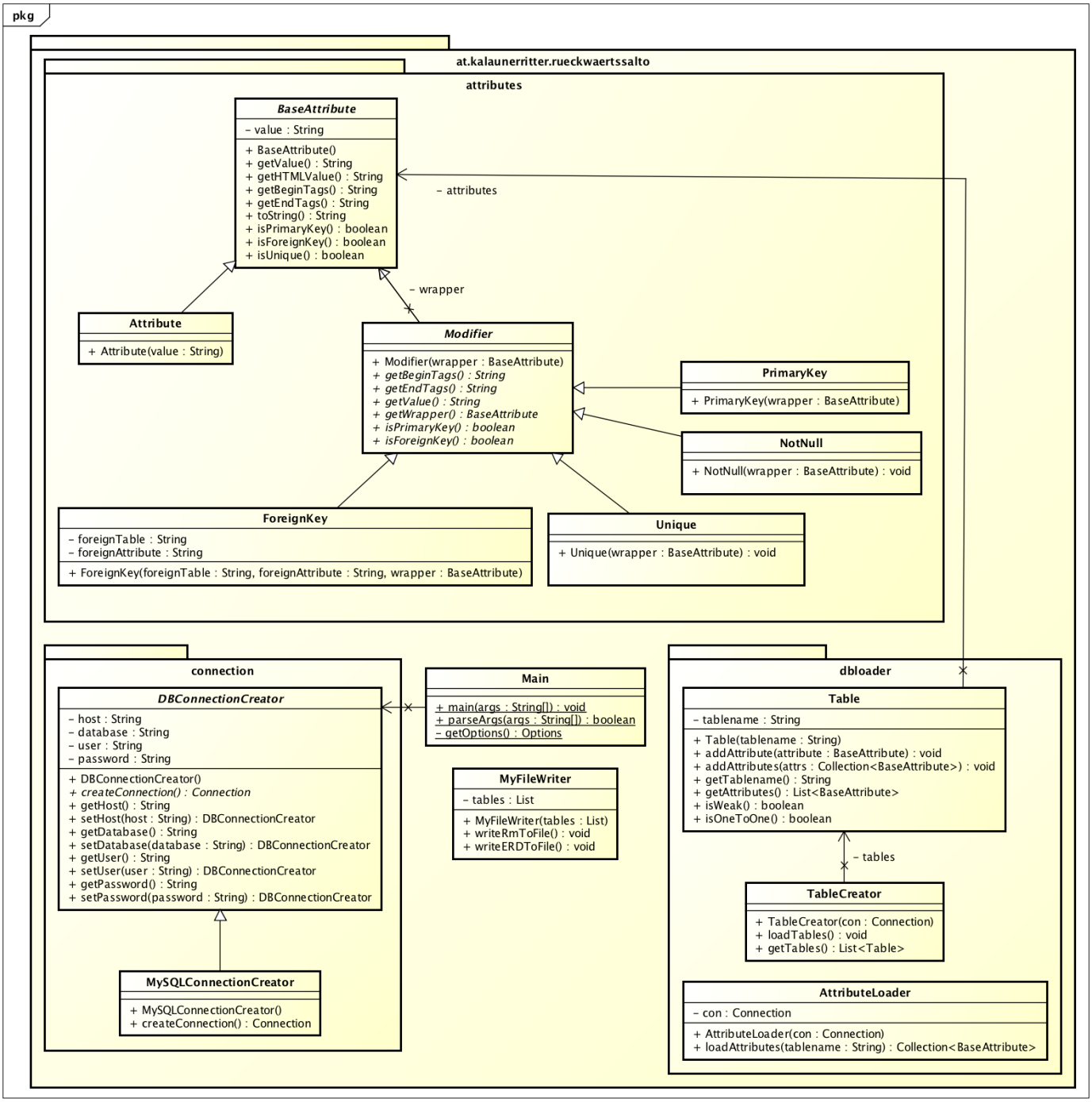
Organisatorische Anforderungen						
Recherche bezüglich ERD-Darstellung	Kalauner/Ritter	x	-	50	90	28.12.14
Recherche bezüglich anderer Darstellung im DOT	Kalauner	x	-	30	40	14.01.15

Summen			
Kalauner	805		
Ritter	760		
Gesamt	1565	26,0833333	Stunden

Alle Zeitangaben sind in Minuten (außer eine andere Einheit wurde angegeben)

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

Argumente, die beim Programmstart angegeben werden können bzw. müssen:

-d **Pflicht**: Name der Datenbank
-h Hostname des DBMS. Standard: localhost
-p Passwort. Standard: keines
-u Benutzername. Standard: Benutzername des im Betriebssystem angemeldeten Benutzers

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

Überprüfen mit dem Test-SQL-Script

Vorgegebenes SQL-Script

Das Programm wurde mit dem Test-SQL-Script von Moodle getestet. Allerdings musste das Script leicht abgeändert werden, da es sich um ein PostgreSQL-Script handelte.

Die Foreign-Keys wurden teilweise folgendermaßen angegeben:

```
country CHAR(2) NOT NULL default '' REFERENCES countries(code)
```

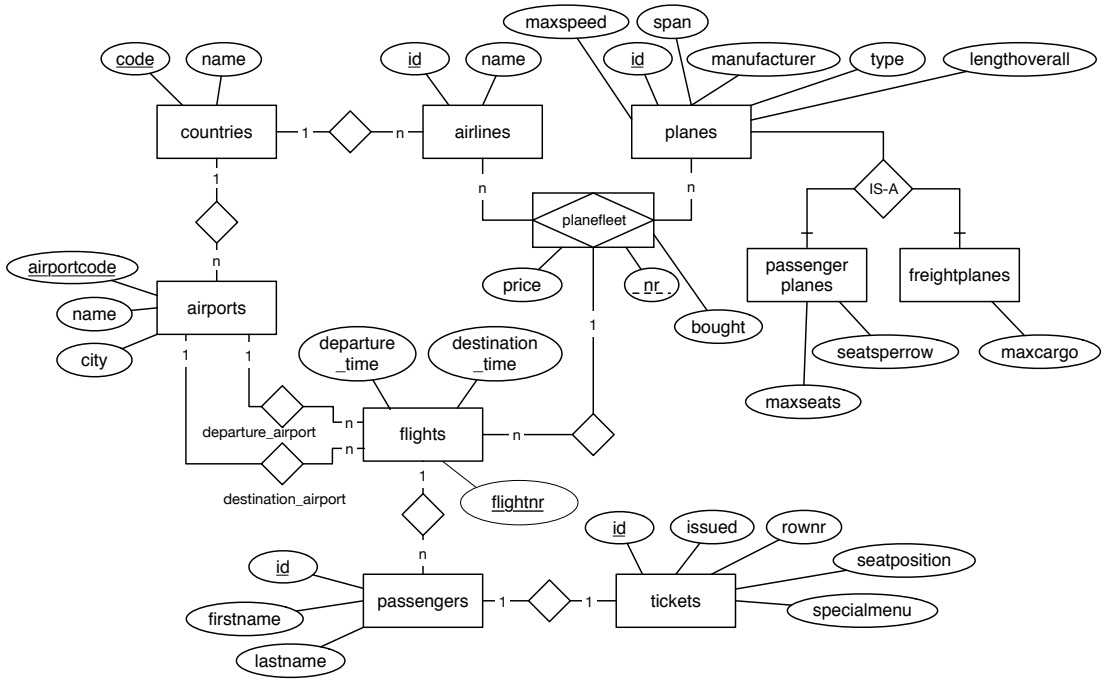
Diese Schreibweise wird in MySQL nicht erkannt und muss daher auf folgende geändert werden:

```
FOREIGN KEY (countries) REFERENCES countries(code)
```

Damit werden nun alle Foreign Keys erkannt.

Vorgegebenes ERD

Beim ERD fehlt bei der Tabelle flights der Primary Key "flightnr", den wir zusätzlich eingezeichnet haben:

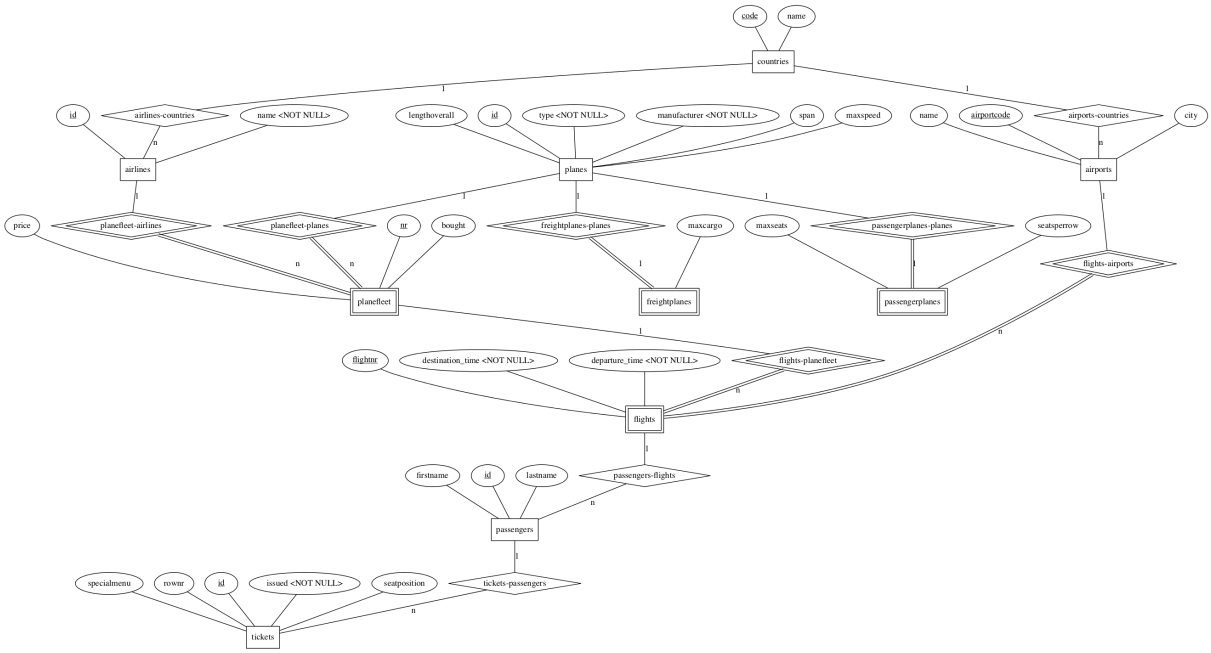


Generiertes RM

airlines (*country: countries.code <NOT NULL>*, name <NOT NULL>, id)
airports (*country: countries.code <NOT NULL>*, city, name, airportcode)
countries (code, name)
flights (*planetype: planefleet.plane*, *departure_airport: airports.airportcode*, flightnr, destination_time <NOT NULL>, *planefleet.airline*, departure_time <NOT NULL>, *destination_airport: airports.airportcode*)
freightplanes (*planes.id*, maxcargo)
passengerplanes (seatsperrow, *planes.id*, maxseats)
passengers (firstname, *flights.flightnr*, id, *flights.airline*, lastname)
planefleet (*plane: planes.id*, nr, bought, price, *airline: airlines.id*)
planes (maxspeed, lengthoverall, id, type <NOT NULL>, manufacturer <NOT NULL>, span)
tickets (specialmenu, rownr, *passenger: passengers.id <UNIQUE> <NOT NULL>*, id, issued <NOT NULL>, seatposition)

Das RM entspricht der vorgegebenen Test-DB

Generiertes ERD



Das ERD entspricht bis auf die Beziehungen zwischen "flights und airports" mit der DB überein. Diese Beziehung wird bei uns nur ein Mal dargestellt, obwohl diese eigentlich zwei Mal vorhanden sein sollte.

Tests und Testbericht

Unit-Tests

Die Unit-Tests befinden sich in den jeweiligen Packages im JAR-File und sind mittels Java-Doc kommentiert. Der Testreport befindet sich im testreport-Folder der JAR-Datei.

Acceptance-Tests

Test 1

Situation: Ungültiger Benutzername eingegeben
Erwartete Ausgabe: Es wird angezeigt, dass die Verbindung nicht aufgebaut werden konnte, inkl. fehlerhafter Eingaben

Tatsächliche Ausgabe:

```
2015-01-21 08:48:45 INFO  MySQLConnectionCreator:44 - Creating MySQLConnection: jdbc:mysql://127.0.0.1/premiere
2015-01-21 08:48:45 ERROR MySQLConnectionCreator:50 - Verbindung zu DB fehlgeschlagen. Angegebene Daten:
2015-01-21 08:48:45 ERROR MySQLConnectionCreator:51 - hostname: 127.0.0.1
2015-01-21 08:48:45 ERROR MySQLConnectionCreator:52 - database: premiere
2015-01-21 08:48:45 ERROR MySQLConnectionCreator:53 - username: sdfh
2015-01-21 08:48:45 ERROR MySQLConnectionCreator:54 - passwort:
```

Ergebnis: Test erfolgreich

Test 2

Situation: Ungültiges Passwort eingegeben
Erwartete Ausgabe: Es wird angezeigt, dass die Verbindung nicht aufgebaut werden konnte, inkl. fehlerhafter Eingaben

Tatsächliche Ausgabe:

```
2015-01-21 08:52:05 INFO  MySQLConnectionCreator:44 - Creating MySQLConnection: jdbc:mysql://127.0.0.1/premiere
2015-01-21 08:52:05 ERROR MySQLConnectionCreator:50 - Verbindung zu DB fehlgeschlagen. Angegebene Daten:
2015-01-21 08:52:05 ERROR MySQLConnectionCreator:51 - hostname: 127.0.0.1
2015-01-21 08:52:05 ERROR MySQLConnectionCreator:52 - database: premiere
2015-01-21 08:52:05 ERROR MySQLConnectionCreator:53 - username: root
2015-01-21 08:52:05 ERROR MySQLConnectionCreator:54 - passwort: blablablabla
```

Ergebnis: Test erfolgreich

Test 3

Situation: Ungültiger Hostname eingegeben
Erwartete Ausgabe: Es wird angezeigt, dass die Verbindung nicht aufgebaut werden konnte, inkl. fehlerhafter Eingaben

Tatsächliche Ausgabe:

```
2015-01-21 08:53:06 INFO  MySQLConnectionCreator:44 - Creating MySQLConnection: jdbc:mysql://sdfhjksaf/premiere
2015-01-21 08:53:06 ERROR MySQLConnectionCreator:50 - Verbindung zu DB fehlgeschlagen. Angegebene Daten:
2015-01-21 08:53:06 ERROR MySQLConnectionCreator:51 - hostname: sdfhjksaf
2015-01-21 08:53:06 ERROR MySQLConnectionCreator:52 - database: premiere
2015-01-21 08:53:06 ERROR MySQLConnectionCreator:53 - username: Mathias
2015-01-21 08:53:06 ERROR MySQLConnectionCreator:54 - passwort:
```

Ergebnis: Test erfolgreich

Test 4

Situation: Gültige Daten eingegeben, Testdatenbank: Schokoladefabrik (aus INSY-Übung)
Erwartete Ausgabe: Es wird ein Relationenmodell generiert, dass der angegebenen Datenbank entspricht.

Tatsächliche Ausgabe:

Auftrag (datum <NOT NULL>, nummer, *firmenname: Firma.name*, status <NOT NULL>)
Auftraginfo (*auftragsnr: Auftrag.nummer*, anzahl <NOT NULL>, *produktnr: Produkt.nummer*, *Auftrag.firmenname*)
Bedienung (*mitarbeiternr: Mitarbeiter.nummer*, *maschinennr: Maschine.nummer*)
Einlagerung (anzahl <NOT NULL>, *produktnr: Produkt.nummer*, *lagerbez: Lager.bezeichnung*)
Firma (name, telnr <NOT NULL>, adr <NOT NULL>)
Kuendigung (*mitarbeiternr: Mitarbeiter.nummer*, kuendigung <NOT NULL>)
Kuenstler (bekanntheit <NOT NULL>, *Person.nummer*)
Kunstschau (datum, ort <NOT NULL>, name, land <NOT NULL>)
Kunstwerk (*kuenstlernr: Kuenstler.nummer <NOT NULL>*, wert <NOT NULL>, *Produkt.nummer*)
Lager (bezeichnung, flaeche <NOT NULL>)
Lagerverwaltung (*mitarbeiternr: Mitarbeiter.nummer*, *lagerbez: Lager.bezeichnung*)
Maschine (nummer, beschreibung <NOT NULL>)
Mitarbeiter (*Person.nummer*, einstellung <NOT NULL>)
Person (vorname <NOT NULL>, nachname <NOT NULL>, nummer)
Produkt (bezeichnung <NOT NULL>, gewicht <NOT NULL>, nummer)
Produktion (*produktnr: Produkt.nummer*, *maschinennr: Maschine.nummer*)
Standardsortiment (preis <NOT NULL>, verpackung <NOT NULL>, *Produkt.nummer*)
Vorfuehrung (*Kunstschau.datum*, platz <NOT NULL>, *Kunstschau.name*, *Kunstwerk.nummer*)

Ergebnis: Test erfolgreich

Test 5

Situation: Gültige Daten eingegeben, Testdatenbank: Schokoladefabrik (aus INSY-Übung)
Erwartete Ausgabe: Es wird ein ERD generiert, dass der angegebenen Datenbank entspricht.
Tatsächliche Ausgabe: Das ERD entspricht der DB
Ergebnis: Test erfolgreich

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]