Projekt Arbeit

Robert Mrasek

mrro1011@hs-karlsruhe.de

Auftraggeber: Tassilo Kienle, o/ZB Stuttgart
Betreut von: Prof. Dr. Frank Schaefer, Hochschule Karlsruhe
Erstellt im: Sommersemester 2012

Inhaltsverzeichnis

1. Aufgabenstellung

Die o/ZB Stuttgart (ohne Zins Bewegung Stuttgart) möchte ihre bestehende Webapplikation durch eine modernere Webapplikation in Ruby on Rails ersetzten. Auf Basis des Datenmodells der alten Anwendung hat Okunevych Dmytro für ein Master-Projekt[1] ein neues Datenmodell entwickelt und dafür eine Java-Programm geschrieben, dass die Daten des alten Modells in das neue Modell überführt. Zudem haben Martin Briewig und Michael Leibel[2] ein Grundgerüst für die o/ZB Stuttgart in Ruby on Rails programmiert. Außerdem hat sich Philipp Henschel [3] mit der Thematik der temporalen Datenhaltung und ihrer Anwendungen für das Projekt beschäftigt.

Meine Aufgabe war es, ein einheitliches Datenmodell zu entwickeln, dass die Erkenntnisse zur Temporalen Datenhaltung von Philipp Henschel berücksichtigt. Zudem muss dafür die Migrationssoftware von Okunevych Dmytro erweitert und angepasst werden. Schließlich mussten noch die Daten in die Ruby on Rails Software von Martin Briewig und Michael Leibel eingebaut werden.

2. Temporale Datenhaltung

2.1 Arten der Temporalen Datenhaltung

Die Temporalen Datenhaltung ist die Technik zum Festhalten von zeitlichen Veränderungen beim Speichern von Daten in der Informationstechnik. In der Arbeit von Philipp Henschel[4] wird zwischen 3 Arten unterschieden :

- Valid-Time

  Es wird ein Zeitraum definiert, in dem das Objekt gültig ist. Dies erlaubt es für jeden Zeitpunkt die zu diesem Zeitpunkt gültige Daten zu bestimmen.

- Transaction-Time

  Bei dieser Technik wird der Zeitpunkt der Änderung gespeichert. Auch werden keine Daten gelöscht sondern nur als gelöscht markiert. Dadurch lassen sich Änderungen am Datenbestand nachvollziehen.

- Bitemporal

  Werden sowohl Valid-Time als auch Transaction-Time verwendet, bezeichnet man dies als Bitemporal.

Nach einer Besprechung mit Prof. Dr. Frank Schaefer und Herr Kienle wurde entschieden, dass für das Projekt der o/ZB Stuttgart nur die Valid-Time verwendet werden soll.

1 vgl (1)
2 Vgl (3)
3 Vgl (2)
4 Vgl (2) , Seite 4

2.3 Umsetzung der Temporalen Datenhaltung

2.3.1 Erweiterung der Tabellen

Für die Umsetzung im Projekt wird eine „Valid-Time-Tabelle“ verwendet[5]. Die bestehenden Tabellen werden um die Felder „GueltigVon“ sowie „GueltigBis“ erweitert. Da Ruby on Rails mehre Datenbanksystem neben MySQL unterstützt wurde das Datenformat „datetime“ gewählt. Dies Beschreibt einen Zeitpunkt auf ein Sekunde genau und ist somit für die Ziele der o/ZB Stuttgart ausreichend. Zudem wird sie von den Ruby on Rails unterstützten Datenbanksystem angeboten.

2.3.2 Definition der Intervalle

Für die Implemenierung der Valid-Time in dem Projekt wird der Intervall als rechtsseitig halboffenes Intervall angeben ([GueltigVon,GueltigBis[). Dies bedeutet dass GueltigVon den Startzeitpunkt beschreibt, welcher ein Teil des Intervalls ist. Der Endzeitpunkt GueltigBis ist nicht mehr Teil des Intervalls. Dies Ermöglicht einen lückenlosen Übergang zwischen den Intervallen[6].

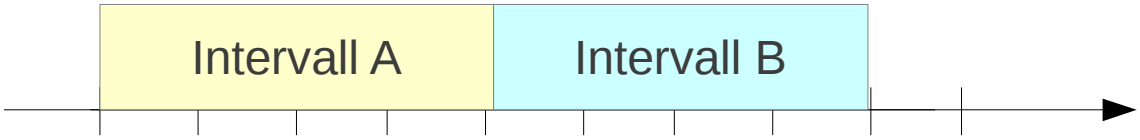

Abbildung 1: 2 Intervalle ohne Zwischenpuffer

Durch diese Wahl der Definition eines Intervall ist es Möglich das Puffer zwischen den Zeitintervallen entstehen bzw. das sich Intervalls überschneiden. Sollte dies unerwünscht sein ist es Aufgabe des Ruby on Rails Anwendung dies zu verhindern.

Abbildung 2: 2 Intervalle mit Zwischenpuffer

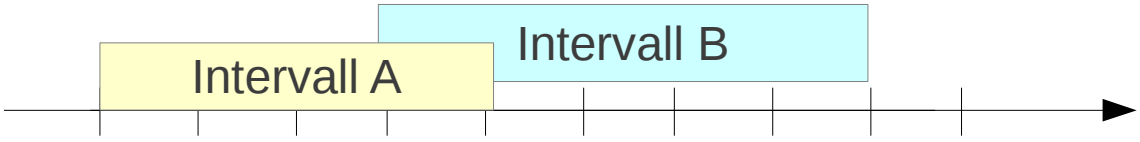

Abbildung 3: 2 überschneidende Intervalle

5 Vgl (2) , Seite 7
6 Vgl (2) , Seite 11

2.3.3 Zugriff auf die Einträge

Um zu signalisieren, dass ein Eintrag auf unabsehbare Zeit gültig bleibt, wird der Wert für GueltigBis auf „9999-12-31 23:59:59“ gesetzt. Somit lässt sich das Aktuell Gültige Datum zu einem Eintrag mit dem SQL-Befehl:

```
SELECT `Person`.* FROM `Person`
WHERE `Person`.`GueltigBis` > '9999-01-01 00:00:00')
```

Eine Möglichkeit den gültigen Datenbestand zu einem beliebigen Zeitpunkt zu erfahren ist:

```
SELECT `Person`.* FROM `Person`
WHERE `Person`.`GueltigVon` <= '2011-07-26 13:51:04'
  AND `Person`.`GueltigBis` > '2011-07-26 13:51:04'
```

Beim Anlegen eines neuen Datensatzes muss darauf geachtet werden dass der Wert für die Spalte GueltigVon gesetzt wird. Falls keine genauere Daten bekannt sind ein guter Standartwert das aktuelle Datum und die Uhrzeit.

Beim Ändern von einem bestehenden Eintrag muss man darauf achten dass das Enddatum der aktuell gültige Eintrag aus der Datenbank gesetzt wird. Für einen lückenlosen Übergang des alten Eintrages zum neuen verwendet man als Endzeitpunkt des alten Eintrages den Startzeitpunkt des neuen Eintrages.

3. Datenmodel

In diesem Kapitel wird das Datenmodel beschrieben wie es von der überarbeiteten Java-Applikation erstellt wird. Grundlage für das Datenmodel war die Ausarbeitung von Okunevych.

3.1 Implementierung der Valid-Time

Die Implementierung erfolgte wie in Kapitel 2 beschrieben durch das hinzufügen von den Spalten GueltigVon und GueltigBis. Für die Überführung des alten Datenbestandes werden der Beginn der Gültigkeit auf den aktuellen Zeitpunkt gesetzt. Als Enddatum wird „9999-12-31 23:59:59“ verwendet.

Auf Wunsch von Herr Kienle wurden folgende Tabellen um die Valid-Time erweitert:

- Personen
- Adresse
- Gesellschafter
- Foerdermitglied
- Partner
- Bankverbindung
- OZBKonto
- EEKonto
- ZEKonto

3.2 Sachbearbeiter

Für den Sacharbeiter wurden ausgewählte Tabellen um die Spalte „SachPNR“ erweitert. Diese Spalte ist von Typ Integer(10) und ist der Verweis auf die Pnr des Sachbearbeiters aus der Tabelle Person.

Folgende Tabelle wurden um den Eintrag „SachPNR“ erweitert:

- Person
- OZBPerson
- Student
- Foerdermitglied
- Partner
- OZBKonto
- EEKonto
- ZEKonto
- Buergschaft

3.3 Weitere Änderungen am Datenmodell

- Die Adressdaten der Tabelle Person wurde in eine eigene Tabelle „Adresse“ ausgelagert.
- Die teilweise redundante Informationen zu einer Bank einer Bankverbindung wurden aus der Tabelle „Bankverbindung“ in die Tabelle „Bank“ ausgelagert.
- Die Spalten „Antragsdatum“, „Aufnahmedatum“ und „Austrittsdatum“ wurden von der Tabelle „Person“ in die Tabelle „OZBPerson“ verschoben. Zusätzlich kam das „Schulungsdatum“ hinzu.
- Die Tabelle „Gesellschafter“ wurde die Spalte „FAIdNr“ hinzugefügt.

3.4 Aufbau der Tabellen

Folgende Abbildung zeigt das ER-Model des Datenmodells. Eine größere Version befindet sich im Anhang. Die Tabellen 1 bis 28 liefern eine genauere Beschreibung der einzelnen Tabellen.

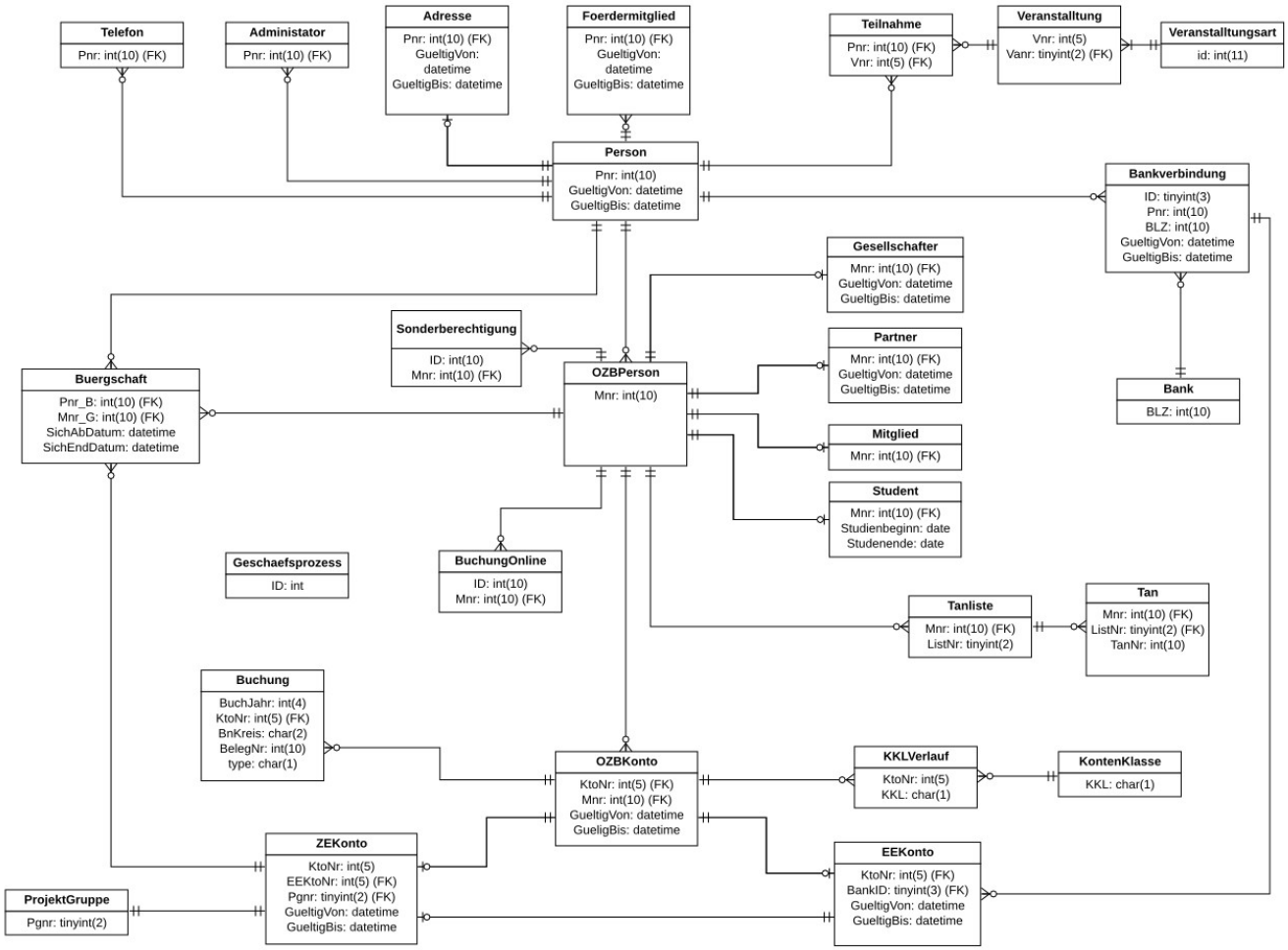

Abbildung 4: ER-Model

Spalte	Schlüssel	DatenTyp	Default	Null erlaubt?
Pnr	PK	int(10)		nein
GueltigVon		datetime		nein
GueltigBis		datetime		nein
Rolle		enum('G', 'M', 'P', 'S', 'F')	NULL	ja
Name		varchar(20)		nein
Vorname		varchar(15)	''	nein
Geburtsdatum		date	NULL	ja
Vermerk		varchar(100)	NULL	ja
Email		varchar(40)	NULL	ja
SperrKZ		tinyint(2)	'0'	nein
SachPNR	FK	int(10)	NULL	ja

Table 1: Tabelle Person

Spalte	Schlüssel	DatenTyp	Default	Null erlaubt?
Pnr	PK, FK	int(10)		Person(`Pnr`)
GueltigVon		datetime		nein
GueltigBis		datetime		nein
Strasse		varchar(50)	NULL	ja
Nr		varchar(10)	NULL	ja
PLZ		int(5)	NULL	ja
Ort		varchar(50)	NULL	ja

Table 2: Tabelle Adresse

Spalte	Schlüssel	DatenTyp	Default	Null erlaubt?
Pnr	PK, FK	int(10)		Person(`Pnr`)
AdminPW		varchar(50)		nein
AdminEmail		varchar(10)	NULL	ja

Table 3: Tabelle Administrator

Spalte	Schlüssel	DatenTyp	Default	Null erlaubt?
Mnr	PK, FK	int(10)		Person(`Pnr`)
UeberPnr	FK	int(10)		Person(`Pnr`)
Passwort		varchar(35)	NULL	ja
PWAendDatum		date	NULL	ja
Antragsdatum		date	NULL	ja
Aufnahmedatum		date	NULL	ja
Austrittdatum		date	NULL	ja
Schulungsdatum		date	NULL	ja
Gesperrt		tinyint(2)	'0'	nein
SachPNR		int(10)	NULL	ja

Table 4: Tabelle OZBPerson

Spalte	Schlüssel	DatenTyp	Default	Null erlaubt?
Mnr	PK, FK	int(10)		nein
RVDatum		date	NULL	ja

Table 5: Tabelle Mitglied

Spalte	Schlüssel	DatenTyp	Default	Null erlaubt?
Mnr	PK, FK	int(10)		nein
GueltigVon		datetime		nein
GueltigBis		datetime		nein
FALfdNr		char(20)		ja
FASteuerNr		char(15)		ja
FAIdNr		char(15)		ja
Wohnsitzfinanzamt		varchar(50)		ja
NotarPnr	FK	Int(10)		ja
BeurkDatum		date		ja
SachPNR	FK	int(10)	NULL	ja

Table 6: Tabelle Gesellschafter

Spalte	Schlüssel	DatenTyp	Default	Null erlaubt?
Mnr	PK, FK	int(10)		nein
AusbildBez		Varchar(30)		ja
InstitutName		Varchar(30)		ja
Studienort		Varchar(30)		ja
Studienbeginn		Date		ja
Studienende		Date		ja
Abschluss		Char(20)		ja
SachPNR		int(10)	NULL	ja

Table 7: Tabelle Student

Spalte	Schlüssel	DatenTyp	Default	Null erlaubt?
Pnr	PK, FK	int(10)		nein
GueltigVon		datetime		nein
GueltigBis		datetime		nein
Region		Varchar(30)		ja
Foerderbeitrag		Decimal(5,2)		ja
SachPNR		int(10)	NULL	ja

Table 8: Tabelle Foerdermitglied

Spalte	Schlüssel	DatenTyp	Default	Null erlaubt?
id	PK	int(11)		nein
VABezeichnung		varchar(30)	NULL	ja

Table 9: Tabelle Veranstalltungart

Spalte	Schlüssel	DatenTyp	Default	Null erlaubt?
Vnr	PK	int(11)		nein
vid	FK	int(11)		nein
VADatum		date		nein
VAOrt		varchar(30)	NULL	ja

Table 10: Tabelle Veranstaltung

Spalte	Schlüssel	DatenTyp	Default	Null erlaubt?
Pnr	PK, FK	int(10)		Person(`Pnr`)
Vnr	PK, FK	int(11)		nein
TeilnArt		enum(a,e,u,l,m)		nein

Table 11: Tabelle Teilnahme

Spalte	Schlüssel	DatenTyp	Default	Null erlaubt?
Mnr	PK, FK	int(10)		nein
GueltigVon		datetime		nein
GueltigBis		datetime		nein
Mnr0	FK	int(10)		nein
Berechtigung		char(1)	'1'	nein
SachPNR	FK	int(10)	NULL	ja

Table 12: Tabelle Partner

Spalte	Schlüssel	DatenTyp	Default	Null erlaubt?
Pnr	PK, FK	int(10)		nein
LfdNr	PK	tinyint(2)		nein
TelefonNr		varchar(15)	NULL	ja
TelefonTyp		char(6)	NULL	ja

Table 13: Tabelle Telefon

Spalte	Schlüssel	DatenTyp	Default	Null erlaubt?
Mnr	PK, FK	int(10)		nein
ListNr	PK,FK	tinyint(2)		nein
Status		enum(n,d,a)		nein

Table 14: Tabelle Tanliste

Spalte	Schlüssel	DatenTyp	Default	Null erlaubt?
Mnr	PK, FK	int(10)		nein
ListNr	PK, FK	tinyint(2)		nein
TanNr	PK, FK	int(10)		nein
VerwendetAm		date	NULL	ja
Status		enum(o,x)		nein

Table 15: Tabelle Tan

Spalte	Schlüssel	DatenTyp	Default	Null erlaubt?
BLZ	PK	int(10)		nein
BIC		char(10)		ja
BankName		varchar(35)	NULL	ja

Table 16: Tabelle Bank

Spalte	Schlüssel	DatenTyp	Default	Null erlaubt?
ID	PK	tinyint(3)		nein
GueltigVon		datetime		nein
GueltigBis		datetime		nein
Pnr	FK	int(10)		nein
BankKtoNr		varchar(12)	NULL	ja
IBAN		char(20)		ja
BLZ	FK	int(10)		nein

Table 17: Tabelle Bankverbindung

Spalte	Schlüssel	DatenTyp	Default	Null erlaubt?
KtoNr	PK	int(5)		nein
GueltigVon		datetime		nein
GueltigBis		datetime		nein
Mnr	FK	int(10)		nein
KtoEinrDatum		date	NULL	ja
Waehrung		char(3)	'STR'	nein
WSaldo		decimal(10,2)	NULL	ja
PSaldo		int(11)	NULL	ja
SaldoDatum		date	NULL	ja
SachPNR		Int(10)	NULL	ja

Table 18: Tabelle OZBKonto

Spalte	Schlüssel	DatenTyp	Default	Null erlaubt?
KtoNr	PK, FK	int(5)		nein
GueltigVon		datetime		nein
GueltigBis		datetime		nein
BankID	FK	tinyint(3)		ja
Kreditlimit		decimal(5,2)	'0.00'	ja
SachPNR	FK	int(10)	NULL	ja

Table 19: Tabelle EEKonto

Spalte	Schlüssel	DatenTyp	Default	Null erlaubt?
Pgnr	PK	tinyint(2)		ja
ProjGruppeBez		varchar(50)		ja

Table 20: Tabelle Projektgruppe

Spalte	Schlüssel	DatenTyp	Default	Null erlaubt?
KtoNr	PK, FK	int(5)		nein
GueltigVon		datetime		nein
GueltigBis		datetime		nein
EEKtoNr	FK	int(5)		nein
Pgnr	FK	tinyint(5)		ja
ZENr		char(10)		ja
ZEAbDatum		date	NULL	ja
ZEEndDatum		date	NULL	ja
ZEBetrag		decimal(10,2)	NULL	ja
Laufzeit		tinyint(4)		nein
ZahlModus		char(1)	'M'	ja
TilgRate		decimal(10,2)	'0.00'	nein
AnsparRate		decimal(10,2)	'0.00'	nein
KDURate		decimal(10,2)	'0.00'	nein
RDURate		decimal(10,2)	'0.00'	nein
ZEStatus		char(1)	'A'	nein
SachPNR	FK	int(10)	NULL	ja

Table 21: Tabelle ZEKonto

Spalte	Schlüssel	DatenTyp	Default	Null erlaubt?
Pnr_B	PK, FK	int(10)		nein
Mnr_G	PK, FK	int(10)		nein
KtoNr	FK	int(5)		nein
SichAbDatum		date	NULL	ja
SichEndDatum		date	NULL	ja
SichBetrag		decimal(10,2)	NULL	ja
SichKurzbez		varchar(200)	NULL	ja
SachPNR	FK	int(10)	NULL	ja

Table 22: Tabelle Buergschaft

Spalte	Schlüssel	DatenTyp	Default	Null erlaubt?
KKL	PK	char(1)		nein
KKLAbDatum		date		nein
Prozent		decimal(5,2)	'0.00'	nein

Table 23: Tabelle Kontenklasse

Spalte	Schlüssel	DatenTyp	Default	Null erlaubt?
KtoNr	PK, FK	int(5)		nein
KKLAbDatum	PK	date		nein
KKL	FK	char(1)		nein

Table 24: Tabelle KKLVerlauf

Spalte	Schlüssel	DatenTyp	Default	Null erlaubt?
BuchJahr	PK	int(4)		nein
KtoNr	PK, FK	int(5)		nein
BnKreis	PK	char(2)		nein
BelegNr	PK	int(10)		nein
Typ	PK	char(1)		nein
Belegdatum		date		nein
BuchDatum		date		nein
Buchungstext		varchar(50)	''	nein
Sollbetrag		decimal(10,2)	NULL	ja
Habenbetrag		decimal(10,2)	NULL	ja
SollKtoNr		int(5)	'0'	nein
HabenKtoNr		int(5)	'0'	nein
WSaldoAcc		decimal(10,2)	'0.00'	nein
Punkte		int(10)	NULL	ja
PSaldoAcc		decimal(10,2)	'0'	nein

Table 25: Tabelle Buchung

Spalte	Schlüssel	DatenTyp	Default	Null erlaubt?
ID	PK, FK	int(10)		nein
Mnr	FK	int(10)		nein
UeberwDatum		date		nein
SollKtoNr		int(5)	'0'	nein
HabenKtoNr		int(5)	'0'	nein
Punkte		int(10)		nein
Tan		int(5)		nein
BlockNr		tinyint(2)	'-1'	nein

Table 26: Tabelle BuchungOnline

Spalte	Schlüssel	DatenTyp	Default	Null erlaubt?
ID	PK, FK	int(10)		nein
Mnr	FK	int(10)		nein
Email		varchar(40)		nein
Berechtigung		Enum('IT', 'MV', 'RW', 'ZE', 'OeA')		nein

Table 27: Tabelle Sonderberechtigung

Spalte	Schlüssel	DatenTyp	Default	Null erlaubt?
ID	PK, FK	int(10)		nein
Beschreibung		varchar(200)		nein
IT		tinyint(1)		nein
MV		tinyint(1)		nein
RW		tinyint(1)		nein
ZE		tinyint(1)		nein
OeA		tinyint(1)		nein

Table 28: Tabelle Geschaeftsprozesse

4. Anpassung der Ruby on Rails-Models

In diesem Kapitel möchte ich erklären wir ich die temporale Datenhaltung in ActiveRecord eingebaut habe. ActiveRecord ist ein Bestandteil von Ruby on Rails und kümmert sich um das Mapping von Objekt und der Datenbank[7]. Auf Basis der Ausarbeitung von Herrn Briewig und Leibel werde ich erklären wie ein Modell in Ruby on Rails aufgebaut ist und welche Besonderheiten für die temporale Datenhaltung notwendig waren. Stellvertretend für alle Klassen werde ich das Modell für die Klasse Personen betrachten. Der komplette Sourcecode befindet sich im Anhang 9.1.

4.1 Definiton eines Modells

Ein Modell in Ruby on Rails ist die Beschreibung wie eine Datenbanktabelle interpretiert und behandelt werden soll. Dazu wird in einer Textdatei mittels einer an Ruby angelegten Sprache beschrieben. Die Datei befinden sich in Ruby on Rails im Ordner „app/models/“. Ein Modell leitet sich immer von der Klasse ActiveRecord::Base ab. Eine gültige minimale Konfiguration ist folgende:

```
class Partners < ActiveRecord::Base
end
```

Nach der Namenskonversion von ActiveRecord wird die Pluralform als Tabellenname für eine Klasse verwendet. In diesem Fall würde ActiveRecord die Tabelle „Partners“ verwenden. Im OZB-Modell hat die Tabelle aber den Namen „Person“, somit muss die beschrieben werden welche Tabelle zu verwenden ist:

```
class Partners < ActiveRecord::Base
   set_table_name "Partner"
end
```

4.2 Definitionen von Attribute

ActiveRecord verwendet für den Zugriff auf den Inhalt einer Spalte der Datenbank Symbole. Ein Symbol in Ruby besteht aus einem „:“ sowie ein String. Symbole, die eine Spalte in der Datenbank repräsentieren, haben den Spaltennamen als String. Für die Spalte „Pnr“ ist dieser „:Pnr“. Die Symbole werden zur Laufzeit als Methoden angeboten. Dadurch kann man durch ein Ruby-Objekt p der Klasse Person durch „p.Pnr“ den Wert lesen bzw. setzten.

Solange nicht genaueres beschrieben ist, sind alle Spalten außerhalb des Objektes sichtbar. Dies entspricht einer public Variablen in Programmiersprachen wie C oder Java. Mit dem Befehlt „attr_accessible“ lässt sich eine Liste von Symbolen definieren, die nach Außen sichtbar sind[8]. Alle anderen sind dann nur noch im Objekt selber verwendbar.

```
attr_accessible :Pnr, :Rolle, :Name, :Vorname, :Geburtsdatum, :Vermerk ...
```

Auch gibt es die Möglichkeit einen Alias für ein Attribut zu setzen. Dieser Alias ist vom Prinzip eine Verknüpfung unter einem anderen Namen.

```
alias_attribute :pnr, :Pnr
```

7 Vgl. (4)
8 Vgl. (5)

4.3 Setzten des Primärschlüssels

Standardmäßig geht ActiveRecord davon aus, dass der Primärschlüssel einer Tabelle den Namen der Tabelle hat mit dem Zusatz „_id“. Dies ist bei dem Datenmodell der OZB nicht der Fall. Für eine Model mit nur einem Primärschlüssel wird folgende Definition verwendet[9]:

```
set_primary_key :Pnr
```

Bei der temporale Datenhaltung muss neben dem Primärschlüssel auch noch der Gültigkeitsbereich beachtet werden. Dafür wird das Plug-in „Composite_Primary_Keys“ benötigt[10]. Dies Plug-in bietet die Methode „set_primary_keys“ an, welche mehre Spalten als Primärschlüssel zulässt. Das führt da zu das der Primärschlüssel im Model definiert wird. In ActiveRecord wird dies durch den folgenden Eintrag realisiert:

```
set_primary_keys :Pnr, :GueltigVon, :GueltigBis
```

4.4 Lesen aus der Datenbank

Für den Zugriff auf ein Objekt aus der Datenbank ist durch die temporale Datenhaltung nicht nur der Primärschlüssel wichtig, sondern auch der Zeitpunkt. Um ein Objekt zu einem gewünschten die Zeitpunkt zu erhalten habe ich der Klasse Person eine Klassenmethode programmiert.

```
def Person.get(pnr, date = Time.now)
   Person.where(:pnr => pnr).where(["GueltigVon <= ?", date]).where(["GueltigBis
> ?",date]).first
end
```

Die Methode sucht das Objekt mit der gegebenen Pnr, das zum gegeben Zeitpunkt date gültig war. Dazu wird überprüft ob der Zeitpunkt größer-gleich des Startzeitpunktes sowie kleiner als der Endzeitpunkt ist. Falls es zu dem Zeitpunkt kein gültiges Objekt gab wird NULL zurückgegeben. Wenn die Methode ohne Zuweisung eines Wertes für date aufgerufen wird wird die aktuelle Zeit und Datum verwendet.

Bei einigen Klassen wie z.B. der Klasse Partner kann auch sein das zu einem Zeitpunkt mehrer Einträge gültig für eine ID gültig sind. So liefert die Methode get_all() alle Partner-Objekte für die ID und den Zeitpunkt.

```
def Partner.get_all(mnrO, date = Time.now)
  Partner.where(:MnrO => mnrO).where
    (["GueltigVon <= ?", date]).where(["GueltigBis > ?",date])
end
```

9 Vgl. (6)
10 http://compositekeys.rubyforge.org/

4.5 Schreiben in die Datenbank

ActiveRecord bietet mehrere Call-back-Methoden an, die beim Schreiben in der Datenbank aufgerufen werden[11]. Für das setzten des Gültigkeitsbereiches sind die Methoden „before_save“ sowie „before_update“ wichtig.

4.5.1 before_save

Bei deinem Call-back wird Aufgerufen, bevor ein neuer Eintrag in die Datenbank geschrieben wird. Dies habe ich dafür genutzt um den Gültigkeitsbereich für ein neu erstellten Eintrag zu setzten:

```
before_save do
   self.GueltigVon = Time.now
   self.GueltigBis = Time.zone.parse("9999-12-31 23:59:59")
end
```

GueltigVon wird auf den aktuellen Zeitpunkt gesetzt und GueltigBis wird auf das Enddatum gesetzt.

4.5.2 before_update

Dieser Call-back wird dann Aufgerufen, wenn ein schon bestehendes Objekt aus der Datenbank neugeschrieben wird. Bei der Temporalen Datenhaltung muss hier dafür gesorgt werden, dass die Gültigkeit der alten Einträge begrenzt werden.

```
before_update do
   if(self.Pnr && self.GueltigBis > "9999-01-01 00:00:00")
      copy = Person.where(:Pnr => self.Pnr).where(:GueltigBis =>
self.GueltigBis).first
      copy = copy.dup
      copy.Pnr = self.Pnr
      copy.GueltigVon = self.GueltigVon
      copy.GueltigBis = Time.now
      copy.save!
      self.GueltigVon = Time.now
      self.GueltigBis = Time.zone.parse("9999-12-31 23:59:59")
   end
end
```

Nach der Überprüfung ob der Eintrag der Aktuellste ist wird eine Kopie mit den gespeicherten Daten aus der Datenbank geladen. Anschließen werden der Gültigkeitsbereich auf den aktuellen Zeitpunkt begrenzt und anschließend in die Datenbank geschrieben. Wie beim Call-back für eine neue Person wird nun auch der Startzeitpunkt auf die aktuelle Uhrzeit und Datum gesetzt sowie das Enddatum gesetzt. Damit ist der Call-back abgeschlossen, und ActiveRecord speichert den Eintrag in der Datenbank.

11 (7)

4.6 Beziehungen zwischen den Tabellen

ActiveRecord erlaubt es Methoden in einem Modell zu definieren, um auf Objekte aus anderen Tabellen zuzugreifen. Dabei wird zwischen 1:1 (has_one) und 1:n (has_many).

4.6.1 has_one

Für eine 1:1 Beziehung wird über den Befehl has_one definiert. Hier wird der Modellname als erster Parameter verwendet, sowie der Schlüssel. Das Objekt p der Klasse Person verfügt nun über die Methode „p.Bankverbinung“, die das Objekt der Methode Bankverbinung liefert. Die Definition hierfür ist:

```
has_one :Bankverbindung, :foreign_key => :pnr
```

Für eine Tabelle mit der temporalen Datenhaltung muss nun auch das Datum berücksichtigt werden. Hierfür sollten automatisch der letzte gültige Eintrag im Gültigkeitszeitraum des Objektes gewählt werden. Die Definition dafür ist wie folgend:

```
has_one :Adresse, :foreign_key => :Pnr, :conditions =>
proc { "GueltigVon >= '#{self.GueltigVon.to_formatted_s(:db)}' AND " +
       "GueltigBis <= '#{self.GueltigBis.to_formatted_s(:db)}' " },
:order => "GueltigBis DESC",
```

Der SQL-Befehlt für einen Zugriff auf das Objekt Adresse einer Person ist wie folgender:

```
SELECT `Adresse`.* FROM `Adresse` WHERE `Adresse`.`Pnr` = 7 AND (GueltigVon >=
'2012-07-26 10:29:25' AND GueltigBis <= '9999-12-31 23:59:59' ) ORDER BY
GueltigBis DESC LIMIT 1
```

Bei der Definition wird aus den Adressen die mit der gleichen Pnr wie das Objekt gewählt. Für die :conditions wird ein proc erzeugt. Ein Proc ist in Ruby ein Stück Programmcode, der erst bei der Ausführung für das Objekt kompiliert. In dem Proc wird ein String generiert, der die Gültigkeit auf die des Personenobjektes limitiert. Schließlich wird die Ergebnissliste noch nach dem größten GueltigBis Wert sortiert und das erste Element gewählt.

Die Beziehung eines Objektes durch zum Sachbearbeiter ist auch durch eine has_one Beziehung gelöst. Hierfür wurde die Methode „sachbearbeiter“ definiert. Diese liefert das aktuellste Personen-Objekt mit der Pnr, die im Feld SachPNR gespeichert ist. Die vollständige Definiton lautet:

```
has_one :sachbearbeiter, :class_name => "Person", :foreign_key => :Pnr,
:primary_key => :SachPNR, :order => "GueltigBis DESC"
```

4.6.2 has_many

Die Definition einer 1:n Beziehung ändelt der Definiton einer 1:1 sehr. Der Hauptunterschied ist das nicht eine einzelnes Objekt zurückgeliefert wird, sondern ein Array mit alles gefundenen Ergebnissen.

```
has_many :partner, :foreign_key => :mnr0
```

Bei dieser Definition liefert für jeden Gültigkeitsbereich eine eigenes Objekt zurück. Sind nur die Einträge zu einem bestimmten Zeitpunkt gewünscht lässt sich dies über die Methode get_all des Klasse erledigen.

5. Rechteverwaltung

Für der Verwaltung der Rechte der Benutzer wurde eine flexible Lösung gewünscht, die auf Rollen basiert und ohne Anpassung des Programmcodes geändert werden kann.

5.1 Implementierung der Rechteverwaltung

Zunächst wurden 5 Rollen definiert

- Admin (AG-IT)
- Mitglieder (AG-MV)
- Finanz (AG-RW)
- Projekte (AG-ZE)
- Öffentlichkeit (AG-ÖA)

In der Tabelle Sonderberechtigung kann nun für einer Mitgliedsnummer eine Rolle zugewiesen werden. Auch können durch mehrer Einträge eine Mitgliedsnummer verschiedene Rollen zugewiesen bekommen. Zudem kann jeder Eintrag auch eine Emailadresse enthalten, die als Rollenspezifische Kontaktadresse dienen können.

Die bekannten Geschäftsvorfälle wurden numberiert. Es wurde eine Standartbelegung erstellt, die von der Migrationssoftware in der Tabelle Geschaeftsprozesse erstellt wird.

ID	Beschreibung	IT	MV	RW	ZE	ÖA
1	Alle Mitglieder anzeigen	√	√	√	√	√
2	Details einer Person anzeigen	√	√	√	√	√
3	Mitglieder hinzufügen	√	√			
4	Mitglieder bearbeiten	√	√			
5	Mitglieder löschen	√	√			
6	Rolle eines Mitglieds zum Gesellschafter ändern	√	√			
7	Mitglied Administratorrechte hinzufügen	√				
8	Kontenklassen hinzufügen	√		√		
9	Kontenklassen bearbeiten	√				
10	Kontenklassen löschen	√				
11	Alle Konten anzeigen	√	√	√	√	√
12	Details eines Kontos anzeigen	√	√	√	√	√
13	Einlage/Entnahmekonten hinzufügen	√		√		
14	Einlage/Entnahmekonten bearbeiten	√		√		
15	Zusatzentnahmekonten hinzufügen	√		√		
16	Zusatzentnahmekonten bearbeiten	√		√		
17	Bürgschaften anzeigen	√	√	√	√	√
18	Bürgschaften hinzufügen	√			√	
19	Bürgschaften bearbeiten	√			√	

Table 29: Berechtigung der Rollen

5.2 Rechteverwaltung in Rails

Für Rails wurde die Klasse Sonderberechtigung entwickelt. Mittels dieser Klassen lassen sich leicht Berechtigungen für eine Mitglied setzten.

```
s = Sonderberechtigung.new
s.Mnr = 5
s.Email = "test@domain.org"
s.Berechtigung = "IT"
s.save
```

Zudem wurde die Methode is_allowed geschreiben. Diese Methode nimmt als Parameter eine Mitgliedernummer sowie die Nummer eines Geschäftsvorfalles. Nun wird überpfüft ob eine Rolle er Mitgliedernummer den Geschäftsvorfall ausführen darf. In diesem Fall wird true zurückgegeben, ansonsten false.

```
def is_allowed(ozb_person, gescheaftsvorfallnr)
if ozb_person && ozb_person.Mnr
  if prozess = Geschaeftsprozess.where(:ID => gescheaftsvorfallnr).first
        return true if prozess.IT && Sonderberechtigung.where(:Mnr =>
ozb_person.Mnr).where(:Berechtigung => "IT").first
        return true if prozess.MV && Sonderberechtigung.where(:Mnr =>
ozb_person.Mnr).where(:Berechtigung => "MV").first
        return true if prozess.RW && Sonderberechtigung.where(:Mnr =>
ozb_person.Mnr).where(:Berechtigung => "RW").first
        return true if prozess.ZE && Sonderberechtigung.where(:Mnr =>
ozb_person.Mnr).where(:Berechtigung => "ZE").first
        return true if prozess.OeA && Sonderberechtigung.where(:Mnr =>
ozb_person.Mnr).where(:Berechtigung => "OeA").first
      end
    return false
end
```

6. Migrationssoftware

Die Java-Applikation zur Migration der Datenbank aus dem PHP-System in das neue Datenmodel von Okunevych Dmytro wurde an das neue Datenmodel angepasst. Zudem wurde die Bedienung des Programmes erweitert. Neben der grafischen Oberfläche gibt es nun auch eine Kommandozeilenbasierte Anwendung. Diese ist dafür gedacht, um auf entfernten Rechnern ausgeführt zu werden (z.B. auf einem Server). Neben der Java-Applikation „migration.jar“ wird auch noch die Text-Datei „create_tables.txt“ benötigt. Diese Datei beschreibt das Datenmodell.

6.1 Grafische Oberfläche

Wenn das Programm ohne zusätzliche Argumente gestartet wird erscheint das grafische Interface. Hier muss der Pfad zur „create_tables.txt“-Datei angegben werden. Standardmäßig wird angenommen, die Datei befindet sich im gleichen Ordner wie das Java-Applikation. Auch wird standardmäßig die Logdatei in den gleichen Ordner geschrieben. Zusätzlich müssen noch die Adresse und die Accountdaten zur Datenbank angegeben werden. Um einen anderen Server als der eigene Rechner zu verwendet werden muss der Zugriff von außen auf dem Server erlaubt sein.

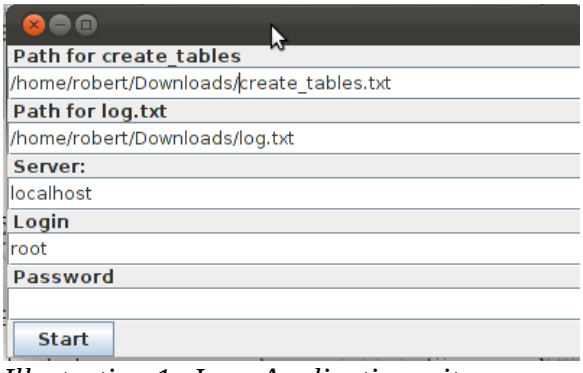

Illustration 1: Java-Application mit grafischen Interface

6.2 Kommandozeilenbasiertes Interface

Die Java-Applikation lässt sich auch auf einem entfernten Rechner ausführen. Hierfür wird ein Terminal benötigt (SSH oder Telnet). Ein guter Client für die beiden Protokolle unter Windows ist Putty[12].

Startet man die Java-Applikation mit Parameter wird auf die grafische Oberfläche verzichtet. Die benötigten Informationen werden direkt übergeben.

Option	Beschreibung
-h	Zeigt eine Hilfeseite
-i	Setzt den Pfad zur txt-Datei mit der Tabellendefiniton (Default: im gleichen Ordner wir die Jar-Datei die Datei create_tables.txt)
-s	Setzt den Server (Default: localhost)
-u	Setzt den Server (Default: localhost)
-p	Setzt das Password

Table 30: Argumente für die Java-Application

Ein Beispiel für die Ausführung auf dem selben Rechner ist:

```
java –jar migration.jar –u root –p geheim
```

6.3 Weiter Möglichkeiten

Sollte werder der Zugriff auf die Datenbank von außen noch ein Zugang über SSH/Telnet möglich sein kann man die Migration auch über PHPMyAdmin tätigen. Hierfür braucht man einen weiteren Rechner auf dem MySQL läuft. Man exporitert die Datenbank des Servers auf den auf den anderen Rechner. Nun führt man die Java-Applikation local aus. Schließlich wird die neue Datenbank exportiert und auf dem Server importiert.

7. Fazit

8. Tabellenverzeichnis

12 http://www.chiark.greenend.org.uk/~sgtatham/putty/

9. Literaturverzeichnis

1: Okunevych, Schriftliche Ausarbeitung,
2: Henschel , Temporale Datenhaltung,
3: Briewig und Leibel, Realisierung Geschäftsprozesse Ruby On Rails,
4: , ActiveRecord API, , http://ar.rubyonrails.org/
5: , , , http://apidock.com/rails/ActiveRecord/Base/attr_accessible/class
6: , Primary Key, , http://apidock.com/rails/ActiveRecord/Base/set_primary_key/class
7: , Callbacks, , http://api.rubyonrails.org/classes/ActiveRecord/Callbacks.html

10. Anhang

10.1 Model person.rb

```
class Person < ActiveRecord::Base

   set_table_name "Person"
   alias_attribute :pnr, :Pnr
   alias_attribute :rolle, :Rolle
   alias_attribute :name, :Name
   alias_attribute :vorname, :Vorname
   alias_attribute :geburtsdatum, :Geburtsdatum
   alias_attribute :vermerk, :Vermerk
   alias_attribute :email, :Email
   alias_attribute :sperrKZ, :SpeerKZ
   alias_attribute :sachPNR, :SachPNR

   set_primary_keys :Pnr, :GueltigBis

   attr_accessible :Pnr, :Rolle, :Name, :Vorname, :Geburtsdatum, :Vermerk,
:Email, :SpeerKZ, :SachPNR

   #validates_presence_of :name, :antragsdatum

   #validates :email, :EmailValidator => true

   has_many :administrator, :foreign_key => :pnr, :dependent => :delete_all
   has_many :telefon, :foreign_key => :pnr, :dependent => :delete_all
   has_many :partner, :foreign_key => :mnrO, :dependent => :delete_all
   has_many :teilnahme, :foreign_key => :pnr, :dependent => :delete_all
   has_many :buergschaft, :foreign_key => :pnrB, :dependent => :delete_all
   has_many :OZBPerson, :foreign_key => :ueberPnr, :dependent => :delete_all
   has_one :Bankverbindung, :foreign_key => :pnr, :dependent => :destroy
   has_one :foerdermitglied, :foreign_key => :pnr, :dependent => :destroy
   #has_one :Adresse, :foreign_key => :Pnr, :conditions => "GueltigBis > '9999-
01-01 00:00:00'"

   before_save do
      self.GueltigVon = Time.now
      self.GueltigBis = Time.zone.parse("9999-12-31 23:59:59")
   end

   before_update do
      if(self.Pnr)
         if(self.GueltigBis > "9999-01-01 00:00:00")
            copy = Person.where(:Pnr => self.Pnr).where(:GueltigBis =>
self.GueltigBis).first
            copy = copy.dup
            copy.Pnr = self.Pnr
            copy.GueltigVon = self.GueltigVon
            copy.GueltigBis = Time.now
            copy.save!
            self.GueltigVon = Time.now
            self.GueltigBis = Time.zone.parse("9999-12-31 23:59:59")
```

```
        end
      end
   end

   # Returns the Personobject valid at the given time
   # Returns nil if at the given time no person object was valid
   def Person.get(pnr, date = Time.now)
      Person.where(:pnr => pnr).where(["GueltigVon <= ?",
date]).where(["GueltigBis > ?",date]).first
   end
end

class EmailValidator < ActiveModel::EachValidator
  def validate_each(record, attribute, value)
    unless value =~ /\A([^@\s]+)@((?:[-a-z0-9]+\.)+[a-z]{2,})\z/i
      record.errors[attribute] << (options[:message] || "is not an email")
    end
  end
end
```

10.2 create_tables.txt

```
CREATE TABLE IF NOT EXISTS `Person` (
  `Pnr` int(10) unsigned NOT NULL,
  `Rolle` enum('G','M','P','S','F') DEFAULT NULL,
  `Name` varchar(20) NOT NULL,
  `Vorname` varchar(15) NOT NULL DEFAULT '',
  `Geburtsdatum` date DEFAULT NULL,
  `Vermerk` varchar(100) DEFAULT NULL,
  `Email` varchar(40) DEFAULT NULL,
  `SperrKZ` tinyint(2) NOT NULL DEFAULT '0',
  `SachPNR` int(10) unsigned DEFAULT NULL,
  `GueltigVon` datetime NOT NULL,
  `GueltigBis` datetime NOT NULL,
  KEY `Pnr` (`Pnr`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE IF NOT EXISTS `Adresse` (
  `Pnr` int(10) unsigned NOT NULL,
  `Strasse` varchar(50) DEFAULT NULL,
  `Nr` varchar(10) DEFAULT NULL,
  `PLZ` int(5) DEFAULT NULL,
  `Ort` varchar(50)  DEFAULT NULL,
  `GueltigVon` datetime NOT NULL,
  `GueltigBis` datetime NOT NULL,
  FOREIGN KEY (`Pnr`) REFERENCES Person(`Pnr`)
          ON UPDATE CASCADE
          ON DELETE RESTRICT,
  KEY (`Pnr`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='';

CREATE TABLE IF NOT EXISTS `Administrator` (
  `Pnr` int(10) unsigned NOT NULL ,
```

```
  `AdminPW` varchar(35) NOT NULL,
  `AdminEmail` varchar(30) DEFAULT NULL,
  FOREIGN KEY (`Pnr`) REFERENCES Person(`Pnr`)
          ON UPDATE CASCADE
          ON DELETE RESTRICT,
  PRIMARY KEY (`Pnr`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='';


CREATE TABLE IF NOT EXISTS `OZBPerson` (
  `Mnr` int(10) unsigned NOT NULL ,
  `UeberPnr` int(10) unsigned ,
  `Passwort` varchar(35) DEFAULT NULL,
  `PWAendDatum` date DEFAULT NULL,
  `Antragsdatum` date DEFAULT NULL,
  `Aufnahmedatum` date DEFAULT NULL,
  `Austrittsdatum` date DEFAULT NULL,
  `Schulungsdatum` date DEFAULT NULL,
  `Gesperrt` tinyint(2) NOT NULL DEFAULT '0',
  `SachPNR` int(10) unsigned DEFAULT NULL,
  FOREIGN KEY (`Mnr`) REFERENCES Person(`Pnr`)
          ON UPDATE CASCADE
          ON DELETE RESTRICT,
  FOREIGN KEY (`UeberPnr`) REFERENCES Person(`Pnr`)
          ON UPDATE CASCADE
          ON DELETE RESTRICT,
  PRIMARY KEY (`Mnr`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='';


CREATE TABLE IF NOT EXISTS `Mitglied` (
  `Mnr` int(10) unsigned NOT NULL ,
  `RVDatum` date default NULL,
  FOREIGN KEY (`Mnr`) REFERENCES OZBPerson(`Mnr`)
          ON UPDATE CASCADE
          ON DELETE RESTRICT,
  PRIMARY KEY (`Mnr`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='';


CREATE TABLE IF NOT EXISTS `Gesellschafter` (
  `Mnr` int(10) unsigned NOT NULL ,
  `FALfdNr` char(20),
  `FASteuerNr` char(15),
  `FAIdNr` char(15),
  `Wohnsitzfinanzamt` varchar(50),
  `NotarPnr` int(10) unsigned  ,
  `BeurkDatum` date,
  `SachPNR` int(10) unsigned DEFAULT NULL,
  `GueltigVon` datetime NOT NULL,
  `GueltigBis` datetime NOT NULL,
  FOREIGN KEY (`Mnr`) REFERENCES OZBPerson(`Mnr`)
          ON UPDATE CASCADE
          ON DELETE RESTRICT,
  FOREIGN KEY (`NotarPnr`) REFERENCES Person(`Pnr`)
```

```
          ON UPDATE CASCADE
          ON DELETE RESTRICT,
  KEY (`Mnr`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='';


CREATE TABLE IF NOT EXISTS `Student` (
  `Mnr` int(10) unsigned NOT NULL ,
  `AusbildBez` varchar(30),
  `InstitutName` varchar(30),
  `Studienort` varchar(30),
  `Studienbeginn` date,
  `Studienende` date,
  `Abschluss` char(20),
  `SachPNR` int(10) unsigned DEFAULT NULL,
  FOREIGN KEY (`Mnr`) REFERENCES OZBPerson(`Mnr`)
          ON UPDATE CASCADE
          ON DELETE RESTRICT,
  PRIMARY KEY (`Mnr`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='';


CREATE TABLE IF NOT EXISTS `Foerdermitglied` (
  `Pnr` int(10) unsigned NOT NULL ,
  `Region` varchar(30),
  `Foerderbeitrag` decimal(5,2),
  `SachPNR` int(10) unsigned DEFAULT NULL,
  `GueltigVon` datetime NOT NULL,
  `GueltigBis` datetime NOT NULL,
  FOREIGN KEY (`Pnr`) REFERENCES Person(`Pnr`)
          ON UPDATE CASCADE
          ON DELETE RESTRICT,
  KEY (`Pnr`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='';


CREATE TABLE IF NOT EXISTS `Veranstaltungsart` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `VABezeichnung` varchar(30) COLLATE utf8_unicode_ci DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='';


CREATE TABLE IF NOT EXISTS `Veranstaltung` (
  `Vnr` int(11) NOT NULL AUTO_INCREMENT,
  `vid` int(11) NOT NULL,
  `VADatum` date NOT NULL,
  `VAOrt` varchar(30) COLLATE utf8_unicode_ci DEFAULT NULL,
  PRIMARY KEY (`Vnr`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='';


CREATE TABLE IF NOT EXISTS `Teilnahme` (
  `Pnr` int(10) unsigned NOT NULL ,
  `Vnr` int(5) unsigned NOT NULL ,
```

```
  `TeilnArt` enum ('a','e','u','l', 'm'),
  FOREIGN KEY (`Pnr`) REFERENCES Person(`Pnr`)
          ON UPDATE CASCADE
          ON DELETE RESTRICT,
  PRIMARY KEY (`Vnr`,`Pnr`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='';

CREATE TABLE IF NOT EXISTS `Partner` (
  `Mnr` int(10) unsigned NOT NULL ,
  `MnrO` int(10) unsigned NOT NULL ,
  `Berechtigung` char(1) NOT NULL DEFAULT '1' ,
  `SachPNR` int(10) unsigned DEFAULT NULL,
  `GueltigVon` datetime NOT NULL,
  `GueltigBis` datetime NOT NULL,
  FOREIGN KEY (`MnrO`) REFERENCES Person(`Pnr`)
          ON UPDATE CASCADE
          ON DELETE RESTRICT,
  KEY (`Mnr`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='';


CREATE TABLE IF NOT EXISTS `Telefon` (
  `Pnr` int(10) unsigned NOT NULL ,
  `LfdNr` tinyint(2) NOT NULL ,
  `TelefonNr` varchar(15)  DEFAULT NULL,
  `TelefonTyp` char(6) DEFAULT NULL,
  FOREIGN KEY (`Pnr`) REFERENCES Person(`Pnr`)
          ON UPDATE CASCADE
          ON DELETE RESTRICT,
  PRIMARY KEY (`Pnr`,`LfdNr`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='';

CREATE TABLE IF NOT EXISTS `Tanliste` (
  `Mnr` int(10) unsigned NOT NULL ,
  `ListNr` tinyint(2) unsigned NOT NULL ,
  `Status` enum ('n','d','a'),
  FOREIGN KEY (`Mnr`) REFERENCES OZBPerson(`Mnr`)
          ON UPDATE CASCADE
          ON DELETE RESTRICT,
  PRIMARY KEY (`Mnr`,`ListNr`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='';

CREATE TABLE IF NOT EXISTS `Tan` (
  `Mnr` int(10) unsigned NOT NULL ,
  `ListNr` tinyint(2) unsigned NOT NULL ,
  `TanNr` int(10) unsigned NOT NULL ,
  `Tan` int(5) NOT NULL,
  `VerwendetAm` date DEFAULT NULL,
  `Status` enum ('o','x'),
  FOREIGN KEY (`Mnr`,`ListNr`) REFERENCES `Tanliste`(`Mnr`,`ListNr`)
          ON UPDATE CASCADE
          ON DELETE RESTRICT,
  PRIMARY KEY (`Mnr`,`ListNr`,`TanNr`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='';
```

```
CREATE TABLE IF NOT EXISTS `Bank` (
  `BLZ` int(10) DEFAULT NULL,
  `BIC` char(10),
  `BankName` varchar(35) DEFAULT NULL,
  PRIMARY KEY (`BLZ`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='';

CREATE TABLE IF NOT EXISTS `Bankverbindung` (
  `ID` tinyint(3) NOT NULL AUTO_INCREMENT ,
  `Pnr` int(10) unsigned NOT NULL ,
  `BankKtoNr`  varchar(12) DEFAULT NULL,
  `IBAN` char(20),
  `BLZ` int(10) unsigned NOT NULL,
  `GueltigVon` datetime NOT NULL,
  `GueltigBis` datetime NOT NULL,
  FOREIGN KEY (`Pnr`) REFERENCES Person(`Pnr`)
          ON UPDATE CASCADE
          ON DELETE RESTRICT,
  KEY (`ID`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='';


CREATE TABLE IF NOT EXISTS `OZBKonto` (
  `KtoNr` int(5) NOT NULL ,
  `Mnr` int(10) unsigned NOT NULL ,
  `KtoEinrDatum`  date DEFAULT NULL,
  `Waehrung` char(3) NOT NULL DEFAULT 'STR',
  `WSaldo` decimal(10,2) DEFAULT NULL,
  `PSaldo` int(11) DEFAULT NULL,
  `SaldoDatum` date DEFAULT NULL,
  `SachPNR` int(10) unsigned DEFAULT NULL,
  `GueltigVon` datetime NOT NULL,
  `GueltigBis` datetime NOT NULL,
  FOREIGN KEY (`Mnr`) REFERENCES OZBPerson(`Mnr`)
          ON UPDATE CASCADE
          ON DELETE RESTRICT,
  KEY (`KtoNr`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='';

CREATE TABLE IF NOT EXISTS `EEKonto` (
  `KtoNr` int(5) NOT NULL ,
  `BankID` tinyint(3)  ,
  `Kreditlimit`  decimal(5,2) DEFAULT '0.00',
  `SachPNR` int(10) unsigned DEFAULT NULL,
  `GueltigVon` datetime NOT NULL,
  `GueltigBis` datetime NOT NULL,
  FOREIGN KEY (`KtoNr`) REFERENCES OZBKonto(`KtoNr`)
          ON UPDATE CASCADE
          ON DELETE RESTRICT,
  FOREIGN KEY (`BankID`) REFERENCES Bankverbindung(`ID`)
          ON UPDATE CASCADE
          ON DELETE RESTRICT,
  KEY (`KtoNr`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='';
```

```
CREATE TABLE IF NOT EXISTS `Projektgruppe` (
  `Pgnr` tinyint(2) ,
  `ProjGruppez` varchar(50) ,
  PRIMARY KEY (`Pgnr`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='';

CREATE TABLE IF NOT EXISTS `ZEKonto` (
  `KtoNr` int(5) NOT NULL ,
  `EEKtoNr` int(5) NOT NULL ,
  `Pgnr`  tinyint(2),
  `ZENr`  char(10) ,
  `ZEAbDatum` date DEFAULT NULL,
  `ZEEndDatum` date DEFAULT NULL,
  `ZEBetrag` decimal(10,2) DEFAULT NULL,
  `Laufzeit` tinyint(4) NOT NULL,
  `ZahlModus` char(1) DEFAULT 'M',
  `TilgRate` decimal(10,2) NOT NULL DEFAULT '0.00',
  `AnsparRate` decimal(10,2) NOT NULL DEFAULT '0.00',
  `KDURate` decimal(10,2) NOT NULL DEFAULT '0.00',
  `RDURate` decimal(10,2) NOT NULL DEFAULT '0.00',
  `ZEStatus` char(1) NOT NULL DEFAULT 'A',
  `SachPNR` int(10) unsigned DEFAULT NULL,
  `GueltigVon` datetime NOT NULL,
  `GueltigBis` datetime NOT NULL,
  FOREIGN KEY (`KtoNr`) REFERENCES OZBKonto(`KtoNr`)
          ON UPDATE CASCADE
          ON DELETE RESTRICT,
  FOREIGN KEY (`EEKtoNr`) REFERENCES EEKonto(`KtoNr`)
          ON UPDATE CASCADE
          ON DELETE RESTRICT,
  FOREIGN KEY (`Pgnr`) REFERENCES Projektgruppe(`Pgnr`)
          ON UPDATE CASCADE
          ON DELETE RESTRICT,
  KEY (`KtoNr`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='';


CREATE TABLE IF NOT EXISTS `Buergschaft` (
  `Pnr_B` int(10) unsigned NOT NULL ,
  `Mnr_G` int(10) unsigned NOT NULL ,
  `KtoNr` int(5) NOT NULL ,
  `SichAbDatum`   date DEFAULT NULL,
  `SichEndDatum` date DEFAULT NULL,
  `SichBetrag` decimal(10,2) DEFAULT NULL,
  `SichKurzbez` varchar(200) DEFAULT NULL,
  `SachPNR` int(10) unsigned DEFAULT NULL,
  FOREIGN KEY (`Pnr_B`) REFERENCES Person(`Pnr`)
          ON UPDATE CASCADE
          ON DELETE RESTRICT,
  FOREIGN KEY (`Mnr_G`) REFERENCES OZBPerson(`Mnr`)
          ON UPDATE CASCADE
          ON DELETE RESTRICT,
  FOREIGN KEY (`KtoNr`) REFERENCES ZEKonto(`KtoNr`)
          ON UPDATE CASCADE
          ON DELETE RESTRICT,
```

```
  PRIMARY KEY (`Pnr_B`,`Mnr_G`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='';



CREATE TABLE IF NOT EXISTS `Kontenklasse` (
  `KKL` char(1) NOT NULL ,
  `KKLAbDatum` date NOT NULL ,
  `Prozent`  decimal(5,2) NOT NULL DEFAULT '0.00',
  PRIMARY KEY (`KKL`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='';


CREATE TABLE IF NOT EXISTS `KKLVerlauf` (
  `KtoNr` int(5) NOT NULL ,
  `KKLAbDatum` date NOT NULL ,
  `KKL`  char(1) NOT NULL ,
  FOREIGN KEY (`KtoNr`) REFERENCES OZBKonto(`KtoNr`)
          ON UPDATE CASCADE
          ON DELETE RESTRICT,
  FOREIGN KEY (`KKL`) REFERENCES Kontenklasse(`KKL`)
          ON UPDATE CASCADE
          ON DELETE RESTRICT,
  PRIMARY KEY (`KtoNr`,`KKLAbDatum`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='';

CREATE TABLE IF NOT EXISTS `Buchung` (
  `BuchJahr` int(4) NOT NULL ,
  `KtoNr` int(5) NOT NULL ,
  `BnKreis`  char(2) NOT NULL ,
  `BelegNr` int(10) unsigned NOT NULL ,
  `Typ`  char(1) NOT NULL ,
  `Belegdatum` date NOT NULL,
  `BuchDatum` date NOT NULL,
  `Buchungstext` varchar(50) NOT NULL DEFAULT '',
  `Sollbetrag` decimal(10,2) DEFAULT NULL,
  `Habenbetrag` decimal(10,2) DEFAULT NULL,
  `SollKtoNr` int(5) unsigned NOT NULL DEFAULT '0',
  `HabenKtoNr` int(5) unsigned NOT NULL DEFAULT '0',
  `WSaldoAcc` decimal(10,2) NOT NULL DEFAULT '0.00',
  `Punkte` int(10) DEFAULT NULL,
  `PSaldoAcc` int(10) NOT NULL DEFAULT '0',
  FOREIGN KEY (`KtoNr`) REFERENCES OZBKonto(`KtoNr`)
          ON UPDATE CASCADE
          ON DELETE RESTRICT,
  PRIMARY KEY (`BuchJahr`,`KtoNr`,`BnKreis`,`BelegNr`,`Typ`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='';


CREATE TABLE IF NOT EXISTS `BuchungOnline` (
  `ID` int(10) NOT NULL AUTO_INCREMENT,
  `Mnr` int(10) unsigned NOT NULL ,
  `UeberwDatum` date NOT NULL,
  `SollKtoNr` int(5) unsigned NOT NULL DEFAULT '0',
  `HabenKtoNr` int(5) unsigned NOT NULL DEFAULT '0',
```

```
  `Punkte` int(10) NOT NULL,
  `Tan` int(5) NOT NULL,
  `BlockNr` tinyint(2) NOT NULL DEFAULT '-1',
  FOREIGN KEY (`Mnr`) REFERENCES OZBPerson(`Mnr`)
          ON UPDATE CASCADE
          ON DELETE RESTRICT,
  PRIMARY KEY (`ID`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='';

CREATE  TABLE IF NOT EXISTS `Temp` (
  `KtoNr` int(5) NOT NULL ,
  `Kreditlimit` decimal(5,2) ,
  `BankKtoNr`  varchar(12) ,
  `BLZ` int(10),
  `BankName` varchar(30),
  PRIMARY KEY (`KtoNr`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT=''
```

10.3 ER-Modell