SS2010
BAI2-DBP Gruppe 1
Lösung zu Übungsblatt 3

R. C. Ladiges, D. Fast

29. April 2010

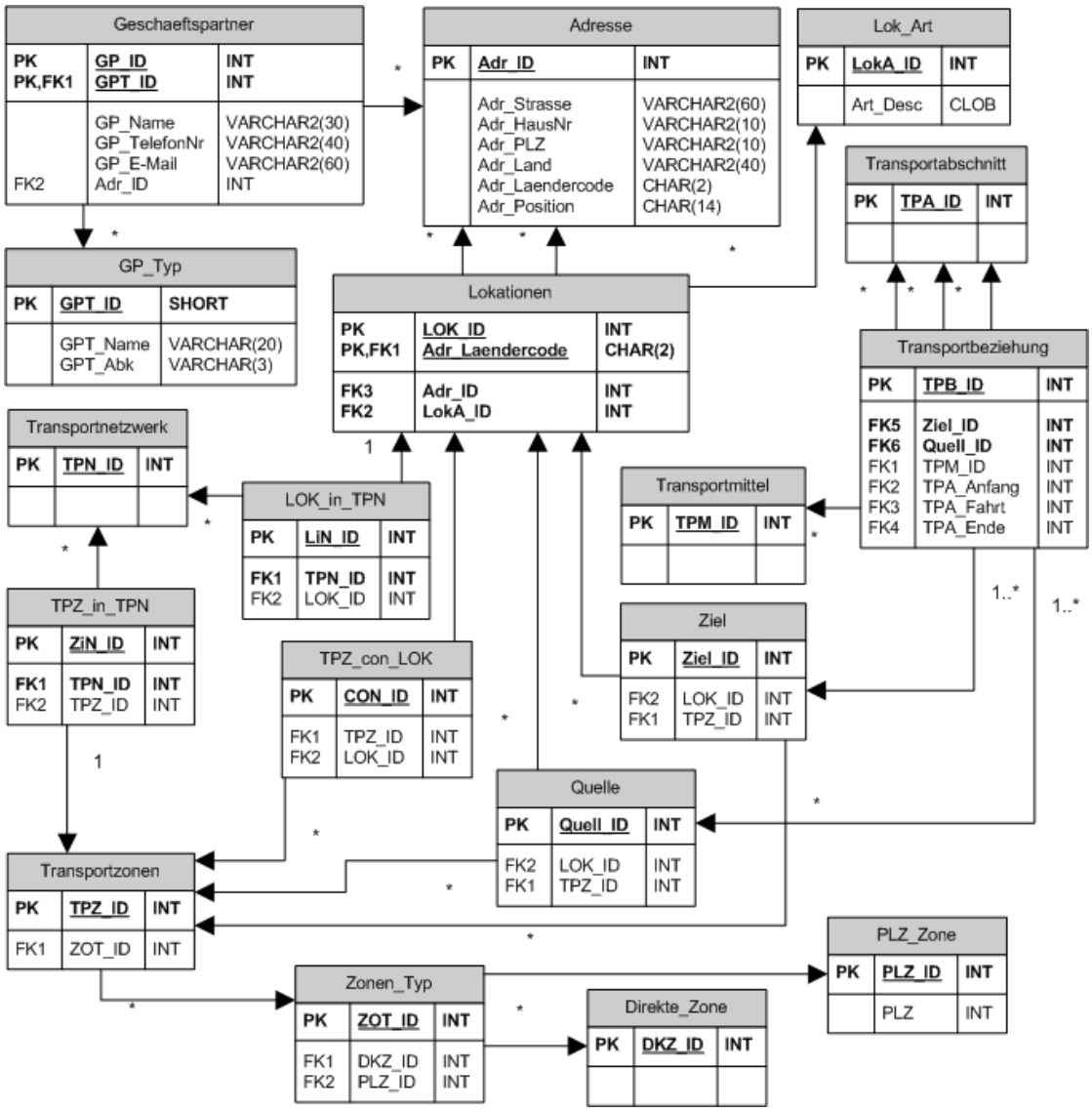
Inhaltsverzeichnis

6 Aufgabe 6 (Abbildung ERM -> RM)

6.0 Aufgabenstellung

Bilden Sie die ERM aus dem letzten Praktikum ins RM ab. Hinweis: achten Sie insbesondere auf die Abbildungsregeln!!! Erstellen Sie danach ein SQL-Skript, um die entstandenen Tabellen in Oracle zu manifestieren. Abzugeben ist ein PDF-Dokument, das die Transformation beschreibt und zusätzlich das SQL-Skript umfasst. Die erfolgreichen Implementierungen werden am Platz abgenommen!

6.1 Relationenmodell



6.2 Beschreibung

Bei der Umsetzung ist darauf zu achten, dass jede Tabelle einen Primärschlüssel bekommt.

Wir beginnen mit der Deklaration der **Geschäftspartner** und **Adressen** Tabelle, mit den dazugehörigen Attributen und dem passenden Datentypen. Die **Geschäftspartner** Tabelle besitzt einen zusammengesetzten Primärschlüssel aus einer fortlaufenden Nummer und der `GPT_ID`, welches ein Fremdschlüssel auf **GP_Typ** ist.

Die speziellen Klassen der generellen **Geschäftspartner** Klasse wird mit der Tabelle **GP_Typ** dargestellt. Jeder Datensatz in ihr repräsentiert einen anderen spezialisierten **Geschäftpartner** mit der jeweiligen Abkürzung (für die eindNr).

Bei den speziellen Klassen der **Lokationen** gehen wir genauso vor. Die **Lokationen** Tabelle besitzt dazu auch wieder einen zusammengesetzten Primärschlüssel, welcher aus der Lokationen ID und dem Ländercode der **Adressen** Tabelle zusammengesetzt ist.

Um die n zu m Beziehung zwischen den **Transportzonen** und **Lokationen** darzustellen, implementieren wir die Zwischentabelle **TPZ_con_LOK** in welcher die beiden Fremdschlüssel zu einem Primärschlüssel zusammenfasst werden, mit jeweils einer 1 zu n Verknüpfung. So ähnlich fasst die **Transportnetzwerk** Tabelle die **Lokationen** und **Transportzonen** auch über Zwischentabellen auf, um die 1 zu n Beziehungen wie eine Komposition umzusetzen.

Die beiden Spezialisierungen (**Postleitzahlenzone** und **Direkte Zone**) der **Transportzonen**, fassen wir nicht so zusammen, wegen dem `PLZ` Attribut, welches sonst irrelevant in der **Direkten Zone** enthalten wäre, sowie in evtl. zukünftigen Spezialisierungen.

Zur Umsetzung der besonderen Zusammensetzungsmöglichkeiten von **Transportbeziehungen** aus **Transportzonen** und **Lokationen** (siehe UB2 A13) haben wir zwei Zwischentabellen namens **Quelle** und **Ziel** eingeführt. Diese enthalten Kontravalent (siehe UB2 A12) **Lokationen** und **Transportzonen**. Die Fremdschlüssel dürfen hier Null sein, nur nicht beide zugleich.

Die dreifache Implementierung eines **Transportabschnittes** in der **Transportbeziehungs** Relation setzen wir einfach mit den drei Fremdschlüsseln `TPA_Anfang`, `TPA_Fahrt` und `TPA_Ende` um, welche alle auf unterschiedliche ID's der **Transportabschnitte** zeigen.

6.3 SQL-Skript

```
CREATE TABLE database.GP_Typ (
  GPT_ID int PRIMARY KEY,
  GPT_Name Varchar(20),
  GPT_Abk Varchar(3)
);

INSERT INTO database.GP_Typ VALUES (1,'Auftraggeber','AG');
INSERT INTO database.GP_Typ VALUES (2,'Empfänger','EMP');
INSERT INTO database.GP_Typ VALUES (3,'Versender','VER');
INSERT INTO database.GP_Typ VALUES (4,'Frachtführer','FRA');
```

```
CREATE TABLE database.Adresse (
  Adr_ID int PRIMARY KEY,
  Adr_Strasse Varchar(60) NOT NULL,
  Adr_HausNr Varchar(10) NOT NULL,
  Adr_PLZ Varchar(10) NOT NULL,
  Adr_Land Varchar(50) NOT NULL,
  Adr_Laendercode Char(2) NOT NULL UNIQUE,
  Adr_Position Char(14) NOT NULL
);

CREATE TABLE database.Geschaeftspartner (
  GP_ID Int,
  GPT_ID int REFERENCES database.gp_typ(GPT_ID),
  Adr_ID int REFERENCES database.adresse(Adr_ID),
  GP_Name Varchar(30) NOT NULL,
  GP_TelefonNr Varchar(20),
  GP_eMail Varchar(60),
  PRIMARY KEY (GP_ID, GPT_ID)
);

CREATE TABLE database.Lok_Art (
  LokA_ID int PRIMARY KEY,
  Art_Desc Clob
);

CREATE TABLE database.Lokationen (
  LOK_ID int UNIQUE,
  Adr_Laendercode Char(2) REFERENCES database.adresse(Adr_Laendercode),
  Adr_ID int REFERENCES database.adresse(Adr_ID),
  LokA_ID int REFERENCES database.Lok_Art(LokA_ID),
  PRIMARY KEY (LOK_ID,  Adr_Laendercode)
);

CREATE TABLE database.Transportnetzwerk (
  TPN_ID int PRIMARY KEY
);

CREATE TABLE database.LOK_in_TPN (
  LiN_ID int PRIMARY KEY,
  TPN_ID int NOT NULL REFERENCES database.Transportnetzwerk(TPN_ID),
  LOK_ID int UNIQUE REFERENCES database.Lokationen(LOK_ID)
);

CREATE TABLE database.Direkte_Zone (
  DKZ_ID int PRIMARY KEY
);
```

```
CREATE TABLE database.PLZ_Zone (
  PLZ_ID int PRIMARY KEY,
  PLZ int NOT NULL
);

CREATE TABLE database.Zonen_Typ (
  ZOT_ID int PRIMARY KEY,
  DKZ_ID int REFERENCES database.Direkte_Zone(DKZ_ID),
  PLZ_ID int REFERENCES database.PLZ_Zone(PLZ_ID)
);

CREATE TABLE database.Transportzonen (
  TPZ_ID int PRIMARY KEY,
  ZOT_ID int REFERENCES database.Zonen_Typ(ZOT_ID)
);

CREATE TABLE database.Transportabschnitt (
  TPA_ID int PRIMARY KEY
);

CREATE TABLE database.Transportmittel (
  TPM_ID int PRIMARY KEY
);

CREATE TABLE database.TPZ_in_TPN (
  ZiN_ID int PRIMARY KEY,
  TPN_ID int NOT NULL REFERENCES database.Transportnetzwerk(TPN_ID),
  TPZ_ID int UNIQUE REFERENCES database.Transportzonen(TPZ_ID)
);

CREATE TABLE database.TPZ_con_LOK (
  CON_ID int PRIMARY KEY,
  TPZ_ID int REFERENCES database.Transportzonen(TPZ_ID),
  LOK_ID int REFERENCES database.Lokationen(LOK_ID)
);

CREATE TABLE database.Quelle (
  Quelle_ID int PRIMARY KEY,
  LOK_ID int REFERENCES database.Lokationen(LOK_ID),
  TPZ_ID int REFERENCES database.Transportzonen(TPZ_ID)
);

CREATE TABLE database.Ziel (
  Ziel_ID int PRIMARY KEY,
  LOK_ID int REFERENCES database.Lokationen(LOK_ID),
  TPZ_ID int REFERENCES database.Transportzonen(TPZ_ID)
);
```

```
CREATE TABLE database.Transportbeziehung (
  TPB_ID int PRIMARY KEY,
  Ziel_ID int not null REFERENCES database.Ziel(Ziel_ID),
  Quell_ID int not null REFERENCES database.Quelle(Quelle_ID),
  TPM_ID int REFERENCES database.Transportmittel(TPM_ID),
  TPA_Anfang int REFERENCES database.Transportabschnitt(TPA_ID),
  TPA_Fahrt int REFERENCES database.Transportabschnitt(TPA_ID),
  TPA_Ende int REFERENCES database.Transportabschnitt(TPA_ID)
);
```

7 Aufgabe 7 (UML- und ER-Modell)

7.0 Aufgabenstellung

In Moodle wurde eine aktuelle Version des Lastenheftes eingestellt. Überprüfen Sie die Anforderungen A01 bis A15 bzgl. Änderungen und führen Sie diese ggf. in Ihrer Modellierung nach. Modellieren Sie die Anforderungen [A16] bis [A22] des HAW Logistics- Lastenheftes in Form von UML-Klassendiagrammen. Transformieren Sie nachfolgend die entstandenen UML-Klassendiagramme in ERModelle. Abzugeben ist ein PDF-Dokument mit den UML-Modellen und den ER-Modellen inkl. Dokumentation.

7.1 A02 neu

7.1.1 Anforderung

Geschäftspartner sind durch eine Nummer der Form „GP-<Art des Geschäftspartners>-n“ (n ist eine natürliche Zahl) eindeutig identifizierbar. Die Art des Geschäftspartners ist wie folgt definiert:

a. Auftraggeber: „AG“
b. Frachtführer: „FRA“
c. Empfänger: „EMP“
d. Versender: „VER“

7.1.2 Beschreibung

Die einzige Änderung an den bisherigen Anforderungen zwischen Version 0.5 und 1.01 des Lastenheftes ist in [A02], durch die Entfernung der vorigen Definition c.

Da wir c nicht umgesetzt haben, da es offensichtlich fehlerhaft war, brauchen wir nichts verändern.

Die folgenden Diagramme sind auf die Darstellung der neuen Anforderungen gekürzt.

7.2 A16

7.2.1 Anforderung

Das Erfassen von Sendungsanfragen muss durch das System unterstützt werden. Eine Sendungsanfrage stellt den Kundenauftrag zum Transport ein oder mehrerer Waren (beschrieben durch Sendungspositionen [A21]) dar, für die HLS den Transport organisieren soll.

7.2.2 UML-Diagramm

7.2.3 Beschreibung

Wir erstellen drei Klassen, **Sendungsanfragen**, **Sendungspositionen** und **Waren**.

Zwischen **Sendungsanfragen** und **Sendungspositionen** haben wir eine Komposition, weil **Sendungspositionen** von einer Bestimmten **Sendungsanfrage** abhängig sind, und nicht ohne zugehöriger **Sendungsanfrage** existieren sollten. Außerdem muss jede **Sendungsanfrage** mindestens eine **Sendungsposition** enthalten.

Jede **Sendungsposition** aggregiert genau eine **Ware**, die in beliebig vielen **Sendungspositionen** vorkommen darf. Dadurch haben wir eine n:m Beziehung zwischen **Sendungsanfragen** und **Waren**, für die **Sendungspositionen** die jeweiligen Zuordnungen darstellen.

Eleganter könnte man es auch mit **Sendungspositionen** als Assoziationsklasse zwischen **Sendungsanfragen** und **Waren** darstellen.

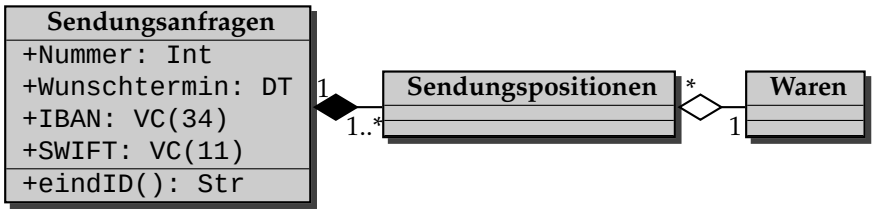
7.3 A17

7.3.1 Anforderung

Jede Sendungsanfrage enthält

a. eine eindeutige Identifikation der Form „HLS-SNA-<Nummer>“, wobei „Nummer“ eine mit dem Wert „1“ beginnende, fortlaufende natürliche Zahl ist. Beispiel: „HLS-SNA-1“
b. einen Wunschtermin für die Abfahrt, bestehend aus einem Datum und eines Zeitfensters von 4 Stunden.
c. Angaben zum Zahlungsweg (IBAN und SWIFT-BIC)

7.3.2 UML-Diagramm



7.3.3 Beschreibung

Sendungsanfragen bekommen die neue Methode `eindID():Str`, die wie `eindNr():Str` aus [A02] und [A08] „HLS-SNA-“ mit dem eigenem Attribut `Nummer:Int` konkateniert.

Zusätzlich bekommen Sendungsanfragen noch ein Attribut namens `Wunschtermin` vom Datentyp DATETIME (kurz: DT), das aus einem Datum und einer Uhrzeit besteht, welches den Zeitpunkt des Beginns des vierstündigen Zeitfensters darstellt.

Das Attribut `IBAN:VC(34)` ist eine 34 Zeichen lange Zeichenkette, da laut [ISO 13616-1:2007] die IBAN bis zu 34 Zeichen enthalten kann. Deutsche IBAN nutzen z.B. nur 22 Zeichen.

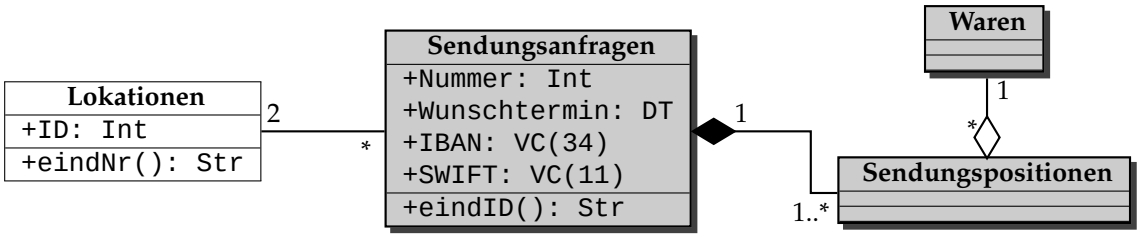
Ähnlich ist das auch für das Attribut `SWIFT:VC(11)`, da ein SWIFT-BIC nach [ISO 9362:2009] aus 8 oder 11 Zeichen bestehen muss.

7.4 A18

7.4.1 Anforderung

Einer Sendungsanfrage ist genau einem Abgangs- und einem Zielort in Form von Lokationen [A07] zugeordnet.

7.4.2 UML-Diagramm



7.4.3 Beschreibung

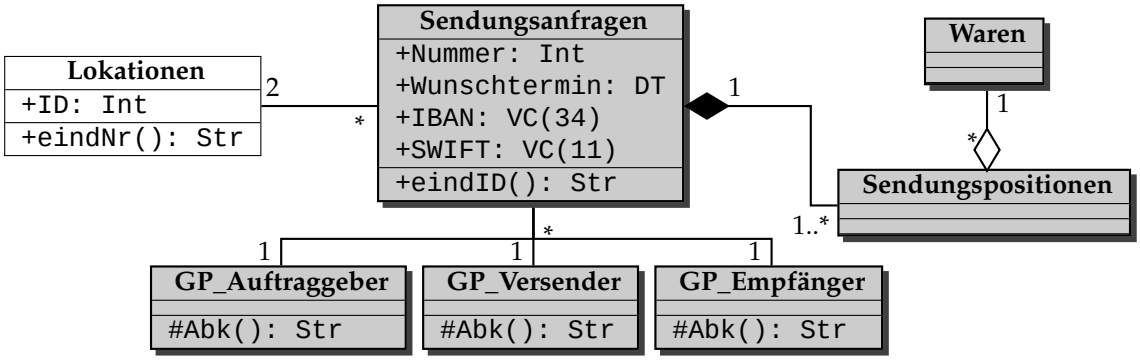
Zugeordnet fassen wir als einfache Assoziation zwischen unseren bereits vorhandenen **Lokationen** und **Sendungsanfragen** auf. Es wäre auch eine Aggregation denkbar. Eine **Sendungsanfrage** besteht aus genau zwei **Lokationen** (*Abgangs* und *Zielort*), die in beliebig vielen **Sendungsanfragen**(, **Transportzonen** und **Transportbeziehungen**) stehen können.

7.5 A19

7.5.1 Anforderung

Einer Sendungsanfrage sind Auftraggeber, Empfänger und Versender [A01] zugeordnet.

7.5.2 UML-Diagramm



7.5.3 Beschreibung

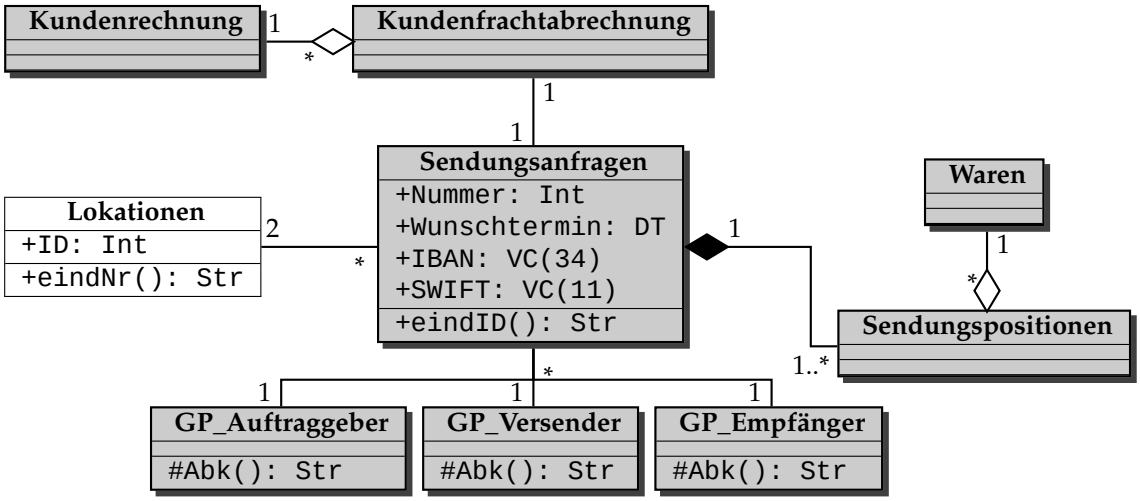
Eine **Sendungsanfrage** steht in Assoziation zu jeweils genau einer unserer bereits vorhandenen Klassen **GP_Auftraggeber**, **GP_Empfänger** und **GP_Versender**. Die Geschäftspartner können in beliebig vielen **Sendungsanfragen** auftauchen. Dies sind drei voneinander unabhängige Assoziationen, auch wenn sie im Diagramm nur einen Knoten an der Klasse **Sendungsanfragen** haben.

7.6 A20

7.6.1 Anforderung

Zu jeder Sendungsanfrage gehört eine Kundenfrachtabrechnung (nach deren Erstellung, siehe [A70]), die die Kundenrechnung enthält.

7.6.2 UML-Diagramm



7.6.3 Beschreibung

Zu jeder **Sendungsanfrage** gehört (Assoziation) eine **Kundenfrachtabrechnung** und umgekehrt. Man könnte sich überlegen ob das „gehören“ durch eine Komposition besser dargestellt wäre.

Eine **Kundenfrachtabrechnung** ist in genau einer **Kundenrechnung** enthalten. Für den Fall, dass eine **Kundenrechnung** für mehrere Sendungen ausgestellt ist, kann sie in mehreren **Kundenfrachtabrechnungen** enthalten sein.

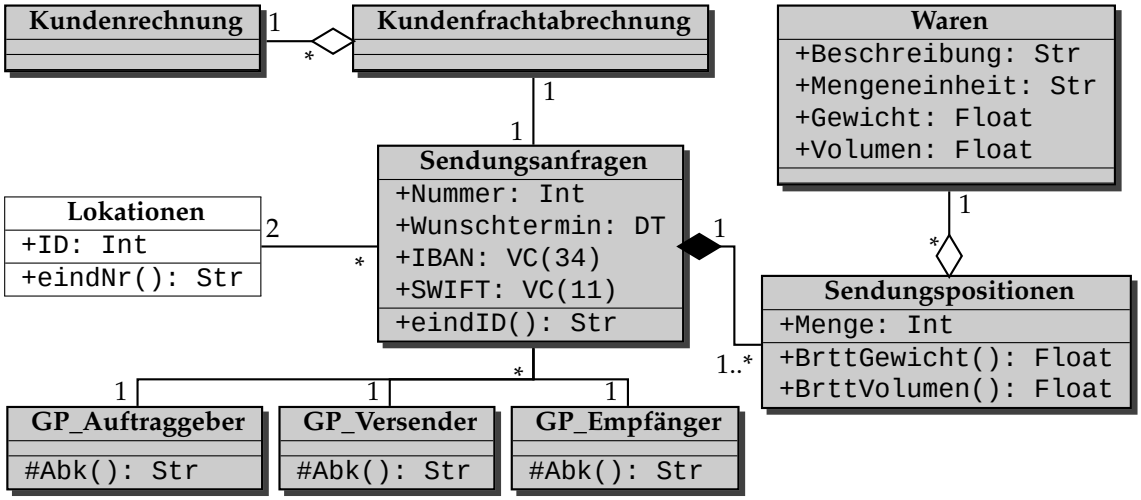
7.7 A21

7.7.1 Anforderung

Eine Sendungsanfrage besteht aus ein oder mehreren Sendungspositionen. Eine Sendungsposition stellt eine beliebige Ware dar (z. B. „Milchkarton“, „Schraube“, „Gurkenglas“, „Computer“). Jede Sendungsposition enthält:

a. eine Warenbeschreibung
b. eine Menge
c. die Mengeneinheit (Stück, Palette, Kolli, TEU, FEU)
d. das Bruttogewicht der Sendungsposition
e. das Bruttovolumen der Sendungsposition

7.7.2 UML-Diagramm



7.7.3 Beschreibung

Wir erweitern die Klasse **Waren** um die Attribute `Beschreibung:Str`, `Mengeneinheit:Str`, `Gewicht:Float`[1] und `Volumen:Float`. Die Klasse **Sendungspositionen** bekommt das Attribut `Menge:Int` womit die Methoden `BrttGewicht():Float` und `BrttVolumen():Float` ihre Werte entsprechend berechnen.

Dass das Attribut `Mengeneinheit` in der Klasse **Waren** und nicht in der Klasse **Sendungspositionen** steht hat den Nachteil, dass man die gleichen **Waren** für unterschiedliche `Mengeneinheiten` redundant erstellt. Dafür spricht, dass größere Mengen häufig zusammengefasst werden. So kann z.B. eine Palette als eigenständige **Ware** betrachtet werden mit einem Verpackungsgewicht der von **Ware** zu **Ware** unterschiedlich ist. Andernfalls bräuchten **Sendungspositionen** einiges mehr an Attributen, um korrekt das Gewicht und Volumen zu errechnen.

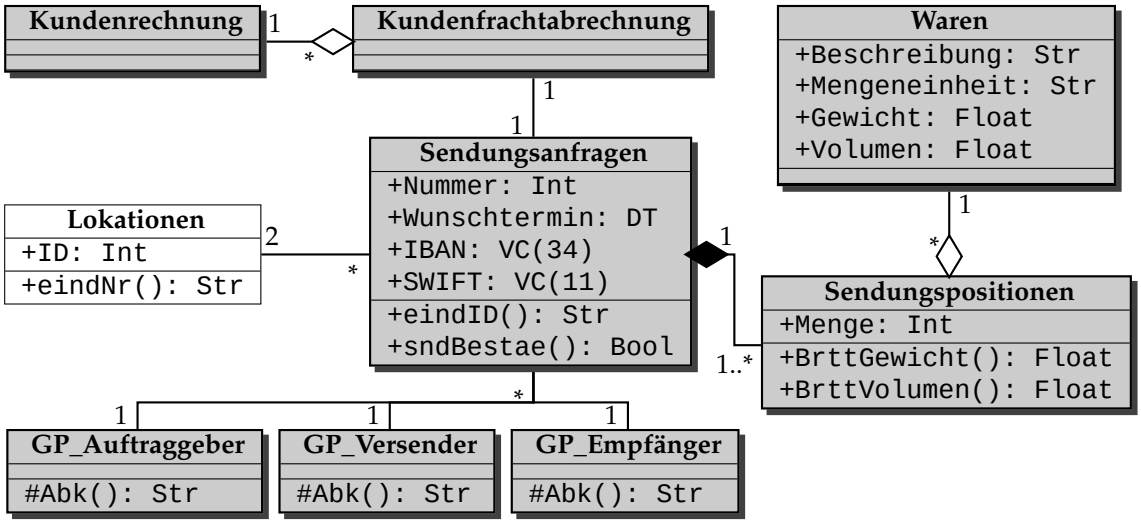
[1] Gemeint ist ein Datentyp für Fließkommazahlen. Die Genauigkeit spielt hierbei keine Rolle. Benötigt bei Gewicht/Volumen Einigkeit über die darzustellende Maßeinheit wenn so implementiert.

7.8 A22

7.8.1 Anforderung

Das System schickt dem Auftraggeber eine Auftragsbestätigung per E-Mail und per Post (Angaben in der Adresse, siehe [A04]).

7.8.2 UML-Diagramm

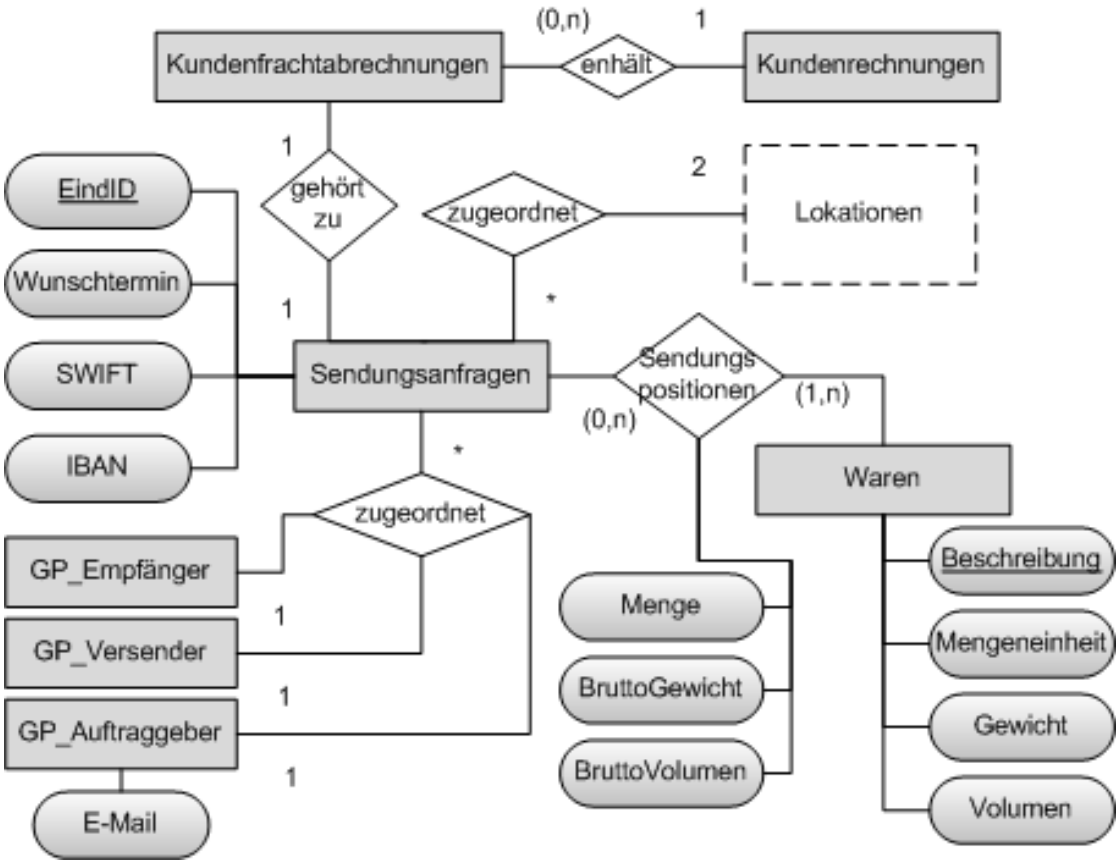


7.8.3 Beschreibung

Da wir auf das bereits vorhandene Attribut `E-Mail` vom **GP_Auftraggeber** (geerbt von **Geschäftspartner**) sowie seiner **Adresse** zugreifen können brauchen wir nichts zu ändern. Praktisch wäre jedoch eine zusätzliche Methode in **Sendungsanfragen** die dies für uns tut.

Z.B.: `sndBestae():Bool` als Boolean (Wahrheitswert: Wahr oder Falsch) der Wahr ist, wenn die Aufgaben ohne Probleme an den eigenen Mail- und Printserver weitergereicht werden konnten.

7.9 ER-Modell



7.9.1 Beschreibung

Die Assoziationen aus dem UML-Diagramm sind hier als Beziehungstyp „gehört zu“ und „zugeordnet“ modelliert.

Interessant ist das **Sendungsposition** nun keine eigene Klasse mehr ist, sondern ein Beziehungstyp zwischen **Sendungsanfragen** und **Waren**, für den immer noch die Eigenschaft der Komposition des UML-Diagrammes gilt, dass jede **Sendungsposition** einer **Sendungsanfrage** „gehört“. Also die Betrachtung von **Sendungsposition** als Assoziationsklasse.

Der Rest ist ziemlich trivial. Wie auch schon bei den UML-Diagrammen ist dies eine verkürzte Darstellung des Modells, welche nur die neuen Anforderungen illustriert.

8 Aufgabe 8 (SQL)

8.0 Aufgabenstellung

Experimentieren Sie mit den bisher erlernten SQL-Befehlen auf Ihrer Instanz der DB herum. Hier ist keine Dokumentation erforderlich.

Informationen zur Signatur

	**Unterzeichner**	EMAILADDRESS=robin.ladiges@haw-hamburg.de, CN=Robin Christopher Ladiges
	Datum/Zeit	Sun Jun 27 00:10:34 CEST 2010
	Austeller-Zertifikat	CN=CAcert Class 3 Root, OU=http://www.CAcert.org, O=CAcert Inc.
	Serien-Nr.	44727
	Methode	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signatur)