Objektorientierte Datenbanken (OODB)

1 Einleitende Bemerkungen

Objektorientierte Datenbanken werden oft auch als *Objektdatenbanken* bezeichnet. Sie wurden entwickelt, um komplexere Anwendungen als bisher zu ermöglichen. Als Beispiele seien genannt: Verwendung komplexer Datentypen (z. B. Tupel, Mengen), Definition anwendungsspezifischer Operationen, Einbettung von Datenbanken in objektorientierte Softwareanwendungen oder längerfristige Transaktionen, Einbindung neuer Datentypen z. B. zum Speichern von Bildern.

Die Entwicklung von Objektdatenbanken begann Mitte der 1980er Jahre. Als Beispiele für Prototypen von OODBs seien ORION (von Microelectronics and Computer Technology Corporation, Texas) oder IRIS (von Hewlett-Packard) genannt[1]. Zur Standardisierung wurde ein Consortium für Anbieter und Nutzer gebildet: die ODMG – *Object Database Management Group*.

In OODBs werden viele Konzepte der objektorientierten Programmiersprachen übernommen, wie z. B. die Definition von Objekten als Instanzen von Klassen, die Kapselung, die Vererbung oder die Polymorphie. Auf einige dieser Begriffe wird im Zusammenhang mit Datenbanken später genauer eingegangen.

2 Ein Beispiel

Folgende Beispielanwendung[2] soll eine Firma mit Informationen über Angestellte, Firmen und deren Zweigstellen modellieren.

Beispiel 1 (Firma, SQL2-Modellierung):

Wir betrachten Entity-Typen mit folgenden Attributen:

1. Firmen haben einen Namen, einen Hauptsitz, Zweigstellen und einen Chef.
2. Zweigstellen haben ebenfalls einen Namen, einen Sitz, einen oder mehrere Leiter und Angestellte.
3. Angestellte haben einen Namen (bestehend aus Familien- und Vornamen), eine Personalnummer und ein Gehalt. Sie arbeiten für genau eine Zweigstelle.
4. Leiter sind Angestellte mit einem Firmenwagen; die Benutzung der Wagen teilt sich auf mehrere Leiter auf.

Das relationale Datenmodell erweist sich für diese Anwendung als unhandlich:

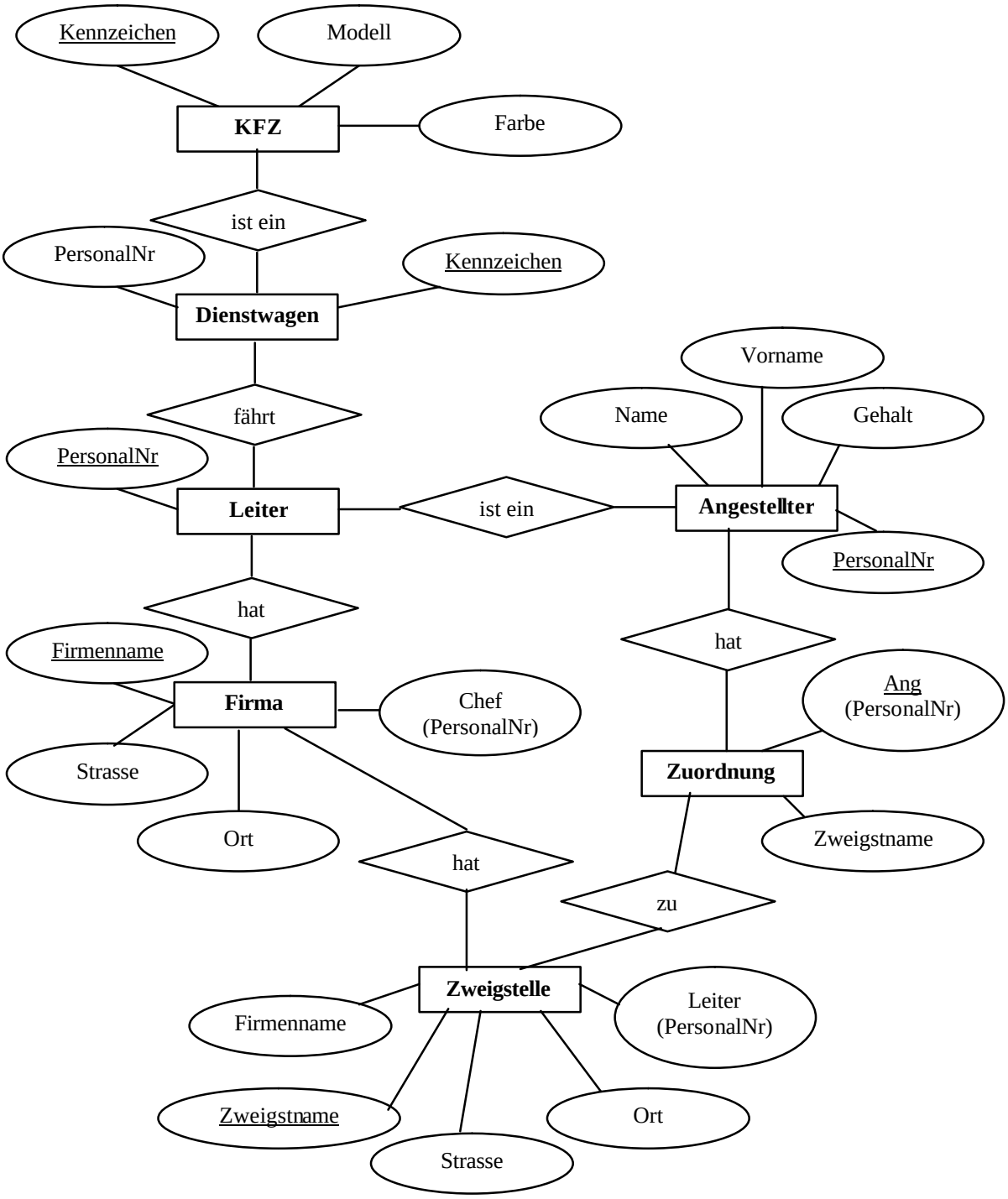
- Zusammengesetzte Attribute (wie der Name einer Person oder der Hauptsitz) müssen zerschlagen werden, d. h. die Komponenten müssen als eigene Attribute deklariert werden.
- Mengenwertige Attribute (wie die Zweigstellen einer Firma) verstoßen gegen die 1NF und müssen geeignet normalisiert werden.
- Spezialisierungen, wie z. B. Leiter als spezielle Angestellte, führen zu eigenen Relationsschemata. Der Zusammenhang zum Superentity-Typ muss durch Fremdschlüssel ausgedrückt werden.
- Künstliche Schlüssel, wie Personalnummern etc. müssen eingeführt werden, wenn z. B. der Name einer Person nicht zur eindeutigen Identifikation genügt.

---

[1] Weitere Beispiele siehe [EN] S. 390

[2] aus [KK] S. 311 ff, leicht abgewandelt

Aufgabe:
Modellieren Sie die beschriebene Datenbank mit Hilfe des Entity-Relationship-Modells.
Eine mögliche Lösung:



Eine mögliche Umsetzung dieser Firmenbeschreibung in SQL2 ist:

```
CREATE TABLE Firma
(       Firmenname VARCHAR(30) NOT NULL PRIMARY KEY,
        Strasse VARCHAR NOT NULL,
        Ort VARCHAR(30) NOT NULL,
        Chef INTEGER NOT NULL REFERENCES Leiter (PersonalNr)
);

CREATE TABLE Zweigstelle
(       Firmenname VARCHAR(30) NOT NULL   REFERENCES Firma (Firmenname),
        Zweigstname VARCHAR(30) NOT NULL PRIMARY KEY,
        Strasse VARCHAR(30) NOT NULL,
        Ort VARCHAR(30) NOT NULL,
        Leiter INTEGER NOT NULL REFERENCES Angestellter (PersonalNr)
);

CREATE TABLE Zuordnung
(       Zweigstname VARCHAR(30) NOT NULL
        REFERENCES Zweigstelle (Zweigstname),
        Ang INTEGER NOT NULL PRIMARY KEY
        REFERENCES Angestellter (PersonalNr)
);

CREATE TABLE Angestellter
(       PersonalNr INTEGER NOT NULL PRIMARY KEY,
        Name VARCHAR(30) NOT NULL,
        Vorname VARCHAR(30) NOT NULL,
        Gehalt INTEGER NOT NULL
);

CREATE TABLE Leiter
(       PersonalNr INTEGER NOT NULL  PRIMARY KEY
        REFERENCES Angestellter (PersonalNr)
        FOREIGN KEY (PersonalNr)
);

CREATE TABLE KFZ
(       Kennzeichen VARCHAR(10) NOT NULL  PRIMARY KEY
        Modell VARCHAR(30) NOT NULL,
        Farbe VARCHAR(30) NOT NULL
);

CREATE TABLE Dienstwagen
(       Kennzeichen VARCHAR(10) NOT NULL REFERENCES KFZ (Kennzeichen),
        PersonalNr INTEGER NOT NULL  PRIMARY KEY
        REFERENCES Leiter (PersonalNr)
);
```

Die komplexen Entity-Typen wurden zerschlagen. Zur Beantwortung von Anfragen müssen sie, oftmals mit der Operation Join, mühsam wieder zusammengesetzt werden.

Beispiel 2 (SQL2-Anfrage):

„Wer ist der Firmenchef des Angestellten Rudi Rüssel, und welchen Dienstwagen fährt er?“

```
SELECT a2.Vorname, a2.Name, d.Kennzeichen
FROM Firma f, Zweigstelle zw, Zuordnung zu,
        Angestellter a1, Angestellter a2 Dienstwagen d
WHERE a1.Name = 'Ruessel'  AND a1.Vorname = 'Rudi'  AND a1.PersonalNr = zu.Ang
        AND zu.Zweigstname = zw.Zweigstname  AND zw.Firmenname = f.Firmenname
        AND f.Chef = a2.PersonalNr  AND a2.PersonalNr = d.PersonalNr;
```

Hieraus wird ersichtlich, dass ein ausdruckstärkeres Datenbankmodell zumindest folgende Eigenschaften aufweisen sollte:

- Komplexe Typdeklarationen: Insbesondere sollte die Bildung von Tupeln und von Mengen in beliebiger Schachtelung möglich sein. Eine Abkehr von der 1NF-Einschränkung ist also unumgänglich.
- Die Wiederverwendung von Information durch Referenzierung: Die Konstruktion neuer Information sollte durch das Zusammensetzen (Aggregation) bereits vorhandener Information möglich sein.
- Die Wiederverwendung von Information durch Spezialisierung mit Vererbung sollte direkt möglich sein.

Wir betrachten nun das folgende abstrakte objektorientierte Datenbankmodell für unsere Fallstudie. Als Notation wird hierbei [ ] für die Tupelbildung und { } für die Mengenbildung verwendet.

Beispiel 3 (Firma, Nicht-relationales Modell):

Firma: [
Name: string,
Hauptsitz: [Strasse: string, Ort: string],
Zweigstellen: {Zweigstelle},
Chef: Leiter]

Durch das Attribut *Chef* ist eine Komponentenbeziehung ausgedrückt, d. h. eine Entität *Firma* hat eine Komponente *Chef*. Der *Chef* selbst ist eine Referenz auf *Leiter*. Durch die Verwendung von Referenzen sollen explizite Fremdschlüssel überflüssig werden.

Zweigstelle: [
Zweigstname: string,
Sitz: [Strasse: string, Ort: string],
Leiter: Angestellter,
Angestellte: {Angestellter}]

Man beachte, dass durch die Verwendung von Referenzen die Zuordnung von Angestellten zu Zweigstellen bereits ausgedrückt ist.

Angestellter: [
PersonalNr : integer,
Name: string,
Vorname: string,
Gehalt : integer]

Die Spezialisierung von Angestellten zu Leitern kann explizit angegeben werden. Dies impliziert eine Wiederverwendung der Struktur von *Angestellte* für *Leiter*.

Leiter inherits Angestellter: [
Dienstwagen: KFZ]

KFZ: [
Kennzeichen: string,
Modell: string,
Farbe: string]

Die Referenz in *Leiter* auf *KFZ* drückt aus, dass ein Wagen mehreren leitenden Angestellten zugeordnet ist.

Bemerkung:
1. Ein verbessertes nicht-relationales Datenmodell soll zudem anstelle von SQL2 eine deklarative Objekt-Query-Sprache (OQL) unterstützen, die direkt auf komplexen Entitäten arbeiten kann.
2. Ein weiterer Mangel des relationalen Datenmodells und von SQL2 ist, dass keine speziellen Operationen für Entitäten definiert werden können (z. B. eine Operation, die die Zahl der Angestellten einer Zweigstelle bestimmt). Solche Operationen können nur in Anwendungsprogrammen, aber nicht zentral im Datenbankmodell festgelegt werden. Ein adäquates Datenbankmodell sollte diesen Mangel beheben.

3 Objektorientierung und Datenbanken

Ein *Objekt* (eine Entität) besitzt eine einzigartige Identität. Jedes Objekt gehört einer *Klasse* (einem Entity-Typ) an. Durch die Klassendefinition werden die Struktur (die Attribute) und das Verhalten (die Methoden/Operationen) der Objekte festgelegt. Struktur und Methoden können privat oder öffentlich sein. Diese Zusammenfassung von Struktur und Methoden zu einem Ganzen nennt man *Kapselung*. Jedes Objekt ist eine *Instanz* (ein Exemplar) einer Klasse. Den Prozess der Erzeugung eines neuen Objektes nennt man *Instantiierung*.

Eine Objektklasse kann eine Spezialisierung von einer oder mehreren Klassen sein. Diesen Prozess nennt man *Vererbung*. Die spezialisierte (erbende) Klasse heißt *Subklasse* ihrer *Superklasse*.

3.1 Datenbankmodellierung mit ODL

Wie einleitend schon erwähnt, ist die ODMG (Object Database Management Group) um Standardisierung objektorientierter Systeme bemüht. Der Standardisierungsvorschlag umfasst ein *Objektmodell*, eine Objektdefinitionssprache ODL (*object definition language*), eine Objektabfragesprache OQL (*object query language*) und Anbindungen an C++.

Das Objektmodell ist wie folgt charakterisiert:

- Der zentrale Baustein ist das Objekt, das mit einer OID (*Objektidentität*) ausgestattet ist.
- Objekte sind von einem bestimmten Typ (einer Klasse), der die Struktur und das Verhalten der Objekte beschreibt.
- Das Verhalten der Objekte wird durch Methoden (Operationen) festgelegt, die durch ihre *Signatur* beschrieben werden. Die Signatur besteht aus dem Namen der Methode, den Namen und Typen der Eingabeparameter und des Rückgabewertes.
- Die Struktur eines Objektes wird durch seine Attributwerte und Beziehungen (Relationships) zu anderen Objekten bestimmt. Zu einer Beziehung muss immer die inverse Beziehung existieren.

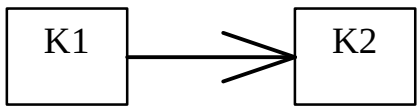
Aufgabe:
Interpretieren Sie folgenden ODL-Quellcode:

```
interface Zweigstelle
( extent alle_Zweigstellen,
  keys Zweigstname )
{
  attribute String Zweigstname;
  attribute [Strasse:String, Ort:String] Sitz;
  attribute Angestellter Leiter;
  relationship Firma von_Firma
    inverse Firma::hat_Zweigstellen;
  relationship Set<Angestellter> hat_angestellt
    inverse Angestellter::angestellt_in;
};
```

Lösung:
Interface, *extent*, *keys*, *attribute*, *relationship*, *inverse* sind Schlüsselwörter. *Zweigstelle* ist der Name der Klasse, die eine Erweiterung von *alle_Zweigstellen* ist und *Zweigstname* als Schlüsselattribut hat. Die Klasse hat drei Attribute: *Zweigstname* vom Typ Zeichenkette, *Sitz* ist ein Tupel aus *Straße* und *Ort*, *Leiter* ist ein *Angestellter*. Es bestehen zwei Beziehungen zu anderen Klassen:

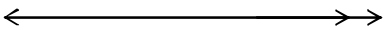
Name der Beziehung	**zu Klasse**	**inverse Beziehung**
von_Firma	*Firma*	*hat_Zweigstellen*
hat_angestellt	*Menge von Angestellten*	*angestellt_in*

Zur grafischen Veranschaulichung dienen folgende Konstrukte:

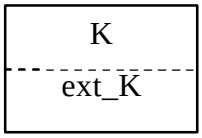


K1 ist direkte Unterklasse von K2


1 : 1 Relationship


1 : N Relationship

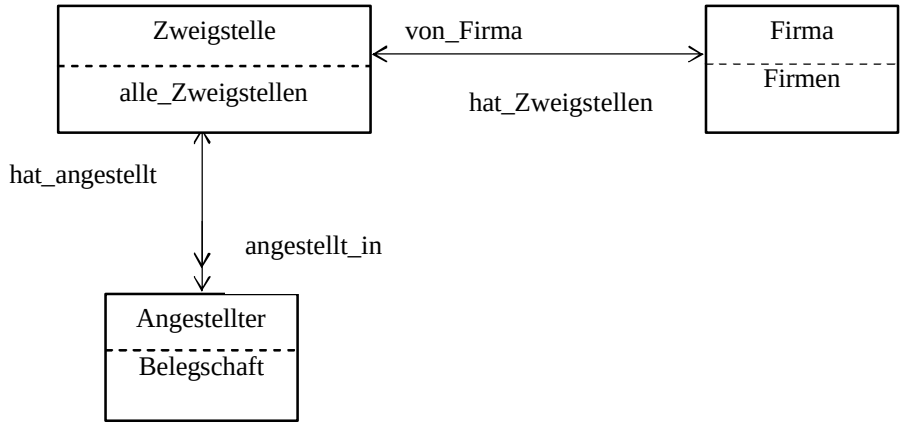

N : M Relationship



Klasse K hat Extension ext_K

Aufgabe:
Stellen Sie die im Beispiel *Zweigstelle* vorkommenden Klassen und ihre Beziehungen grafisch dar.

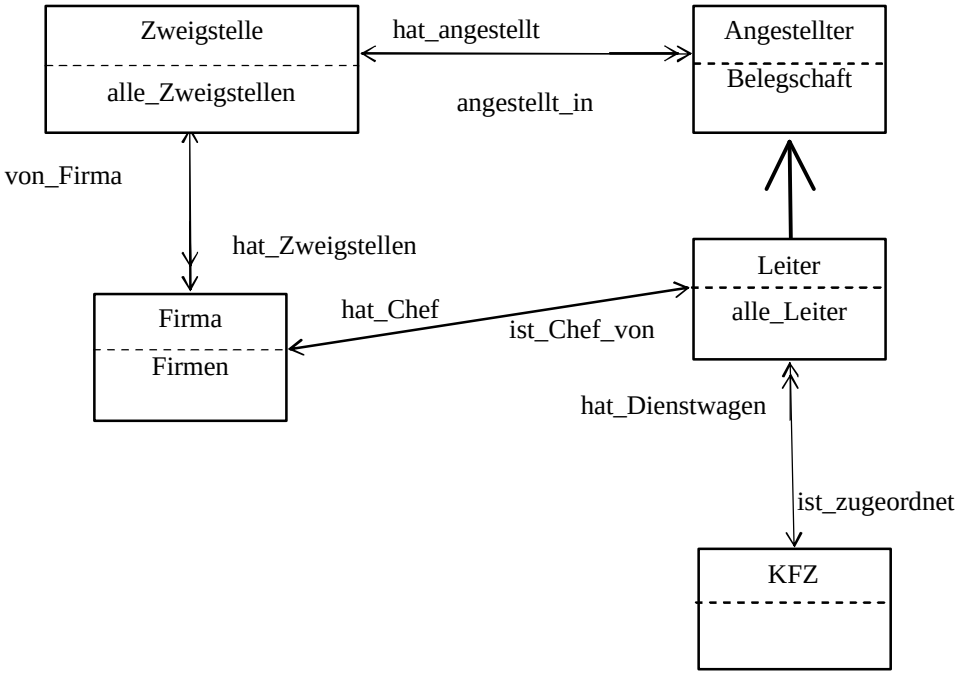
Lösung:



Zusammenfassung

- Eine Klasse ist eine Typspezifikation mit einer ihrer Implementierungen.
- Klassenhierarchien einschließlich Mehrfachvererbung werden unterstützt.
- Die Verwaltung einer Extension (Extent) ist optional. Wird ein Extent definiert, so wird er vom DBS automatisch verwaltet.
- Die Extents realisieren eine Set-Inclusion-Semantik und definieren somit eine ISA-Beziehung.
- Beziehungen zwischen Typen werden explizit behandelt und als binäre Relationships modelliert. Unterstützt werden die Funktionalitäten 1 : 1, 1 : N und N : M.
- Referentielle Integrität wird überwacht und erzeugt bei Verletzung zur Laufzeit eine Ausnahme (Exception).
- Die Methodenimplementierungen sind nicht Teil des Objektmodells von ODMG-93. Sie werden in einer Host-Sprache wie C++, Smalltalk, O2C oder auch OQL vorgenommen.

Wenden wir uns wieder dem Beispiel aus 2 zu. Das folgende Modell enthält die Klassenhierarchie und alle Beziehungen[3].



[3] Ein weiteres Beispiel siehe in [EN] S. 432

Aufgabe:
Vergleichen Sie obiges Modell mit folgendem zugehörigen ODL-Quellcode und beantworten Sie folgende Fragen:
Welche Attribute haben die Klassen? Von welchem Typ sind die Beziehungen und ihre Inversen? Gibt es Klassen mit Methoden? Wenn ja, welche Signaturen haben diese?

```
interface Firma
(extent Firmen,
  keys Firmenname)
{
  attribute String Firmenname;
  attribute Struct Adresse{String Strasse, String Ort}  Hauptsitz;
  relationship Leiter hat_Chef
    inverse Leiter::ist_Chef_von;
  relationship List<Zweigstelle> hat_Zweigstellen
    inverse Zweigstelle::von_Firma;
};

interface Zweigstelle
(extent alle_Zweigstellen,
  keys Zweigstname)
{
  attribute String Zweigstname;
  attribute [Strasse:String, Ort:String] Sitz;
  attribute Angestellter Leiter;
  relationship Firma von_Firma
    inverse Firma::hat_Zweigstellen;
  relationship Set<Angestellter> hat_angestellt
    inverse Angestellter::angestellt_in;
};

interface Angestellter
(extent Belegschaft,
  keys PersonalNr)
{
  attribute String PersonalNr;
  attribute String Name;
  attribute String Vorname;
  attribute Integer Gehalt;
  relationship Zweigstelle angestellt_in
    inverse Zweigstelle::hat_angestellt;
  Integer Alter();
  void erhoehe_Gehalt
    (Integer Gehaltserhoehung)
 };

interface Leiter : Angestellter
(extent alle_Leiter)
{
  relationship Firma ist_Chef_von
    inverse Firma::hat_Chef;
```

```
  relationship KFZ hat_Dienstwagen
    inverse KFZ::ist_zugeordnet;
};

interface KFZ
(keys Kennzeichen)
{
  attribute String Kennzeichen;
  attribute String Modell;
  attribute String Farbe;
  relationship Firma ist_zugeordnet
    inverse Leiter::hat_Dienstwagen
}
```

Hinweis:
Die Klasse *Angestellter* hat zwei Methoden mit folgenden Signaturen:

Name	**Name der Eingabepar.**	**Typ**	**Typ Rückgabewert**
Alter	kein	-	Integer
erhoehe_Gehalt	Gehaltserhöhgung	Integer	kein (void)

3.2 Objektabfragesprache OQL

Die OQL-Syntax für Anfragen ähnelt SQL mit zusätzlichen Features für komplexe Objekte, Operationen, Vererbung und Beziehungen.
Wie bei SQL ist der Grundaufbau einer Abfrage *SELECT … FROM … WHERE*;

Abfrage A1:
SELECT f.Adresse FROM f IN Firmen WHERE f.Firmenname = 'Unicon';

Im Allgemeinen ist für jede Anfrage ein Einstiegspunkt zur Datenbank nötig. Dabei muss es sich um ein beliebiges benanntes, persistentes (sich in der Datenbank befindendes) Objekt handeln. Für viele Anfragen ist der Einstiegspunkt der Namen des Extents der Klasse (in A1 *Firmen*). Die in A1 vorkommende Variable *f* ist eine so genannte *Iterationsvariable*.
Eine OQL-Abfrage muss aber nicht der obigen Struktur folgen.

Abfrage A2:
Firmen;

gibt eine Referenz auf die Menge aller persistenten Firmen-Objekte zurück.
Auf die Komponenten eines Objektes greift man durch Angabe des Namens des Objektes und des Komponentennamens getrennt durch einen Punkt zu. Solche Konstrukte wie z. B. *f.Firmenname* aus A1 nennt man *Pfadausdrücke*.

Kehren wir zur Anfrage von Seite 4 zurück: „Wer ist der Firmenchef des Angestellten Rudi Rüssel, und welchen Dienstwagen fährt er?“. Mit OQL ergibt sich folgende Lösung.

Abfrage A3:

```
SELECT struct (
            chef_Vorname: p.Vorname,
            chef_Name: p.Name,
            chef_Dienstwagen: p.hat_Dienstwagen.Kennzeichen)
FROM p IN
      (SELECT a.angestellt_in.von_Firma.hat_Chef
      FROM a IN Angestellter
      WHERE a.Vorname = 'Rudi' AND a.Nachname = 'Ruessel');
```

Literatur:

[EN] *Elmasri*, *Ramez*; *Navathe*, *Shamkat B.*: Grundlagen von Datenbanksystemen, 3., überarbeitete Auflage. München (Pearson Education) 2002.

[KK] *Kiesling*, *Werner*; *Köstler*, *Gerhard*: Multimedia-Kurs Datenbanksysteme, Berlin u. a. (Springer) 1998.

[R] *Rolland, F. D.*: Datenbanksysteme im Klartext. München (Pearson Education) 2003.