

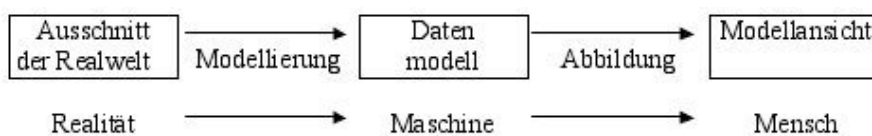
## 2 Die Datenbank Buchausleihe

In diesem Kapitel wirst du die einzelnen Schritte von der Problemstellung bis zum fertigen Datenmodell kennen lernen. Es enthält ziemlich viel Theorie, ich versuche jedoch immer wieder auf die bekannten Tatsachen aus Kapitel 1:

*Vertraut werden, Probleme erkennen* hinzuweisen.

### 2.1 Das Klassendiagramm

Für den Aufbau einer Datenbank muss zuerst ein Datenmodell definiert werden, also ein möglichst exaktes Abbild eines realen Weltausschnitts.



**Abbildung 2.1:** Von der Realwelt zur Datenbank

Du als Datenbankentwickler musst eine genaue Analyse der Realwelt durchführen. Dabei wirst du wichtige Dinge von den unwichtigen trennen, um die Prozessabläufe so weit wie möglich zu vereinfachen und zu verstehen. Dieser Vorgang wird als Modellierung bezeichnet. Bei dieser Analyse werden die Dinge der Realwelt als Objekte und Prozessabläufe als Objektbeziehungen beschrieben. Du entwickelst ein Datenmodell und implementierst dieses in einem geeigneten Programmierwerkzeug. In der Regel ist dieses Datenmodell recht abstrakt und nicht benutzerfreundlich. Du wirst also zum Schluss noch eine geeignete Benutzeroberfläche entwerfen müssen, so genannte Formulare zur Eingabe und Berichte zur Ausgabe der gesammelten Daten.

Um den in Abbildung 2.1 dargestellten Prozess zu verstehen, wirst du nun eine kleine Datenbank entwickeln. Die Fachschaft Informatik am Gymnasium Königsbrunn baut eine kleine Bücherei aus und möchte, dass Schüler die Bücher für einen begrenzten Zeitraum sich ausleihen dürfen. Du hast nun die Aufgabe, die Bücherei und den Ausleihvorgang in einer Datenbank zu erfassen.

#### 2.1.1 Ermittlung der Informationsstruktur

Im ersten Schritt musst du dich mit dem Ausleihvorgang vertraut machen. In einem Gespräch mit dem Auftraggeber erfährst du folgendes:

1. In der Bücherei gibt es (in der Regel) von jedem Buch nur ein Exemplar. Dieses Buch wird erfasst mit seinem Titel, seinen Autoren und der ISBN und dem Verlag.

2. Eine Person darf mehrere Bücher ausleihen. Die maximale Ausleihzeit beträgt eine Woche.
3. Jede Person wird mit Nachnamen, Vornamen, Adresse des Hauptwohnsitz und Telefonnummer registriert.
4. Im Laufe der Zeit hat sich jedoch herausgestellt, dass einige bestimmte Bücher recht häufig verlangt wurden. Diese sind nun doppelt vorhanden. Zur Verwaltung wird hinter dem Buchtitel die Zahl 1 bzw. 2 angehängt.

### Übung 2.1:

Erfasse zusammen mit deinem Nachbarn auf einem Blatt Papier die oben aufgeführten Daten

- a) aller Bücher, die hier im Computerraum den Schülern zur Verfügung gestellt werden,
- b) von acht beliebigen Mitschüler.

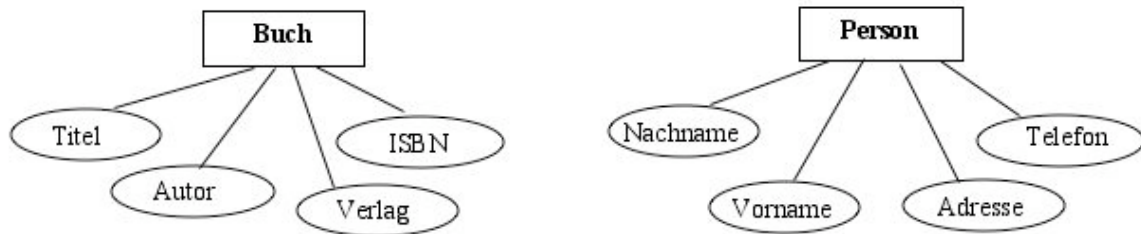
Simuliere den Ausleihvorgang. Hierzu kannst du beispielsweise geeignete Karteikarten, Briefumschläge, usw. entwerfen.

Versuche den Vorgang möglichst effizient zu gestalten. Bedenke, dass in einer realen Bücherei einige tausend Bücher und Personen erfasst werden müssen.

Im Prinzip hast du nun eine kleine Datenbank entworfen. In dieser wird mit Karteikarten gearbeitet und vieles muss mit der Hand geschrieben bzw. sortiert werden. Hierbei musst du aufpassen, dass die Karteikarten nicht aus Versehen in die falsche Ablage geschoben werden.

Ist eine Datenbank korrekt implementiert, kann diese die Sortieraufgaben übernehmen und menschliche Fehler erkennen. Deshalb ist es wichtig, dass du gleich zu Beginn die Struktur einer Datenbank genauestens erkennst. Eine spätere Umstrukturierung gestaltet sich äußerst schwierig.

Die Übung 2.1 zeigt, dass es sinnvoll, zwei verschiedene Formulare zu verwenden. Das erste Formular enthält Felder, in die alle Daten eines Buchs eingetragen werden können. Das zweite Formular enthält alle Daten der entsprechenden Person. Du hast also die beiden Klassen *Buch* und *Person* erstellt mit ihren jeweiligen Datenfeldern. Abbildung 2.2 zeigt ein noch vorläufiges Modell der beiden Klassen.



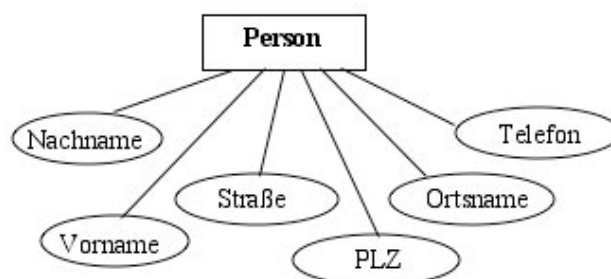
**Abbildung 2.2:** Vorläufige Klassen *Buch* und *Person*

Das Modell dieser beiden Klassen bereitet noch einige Probleme, mit denen du dich noch beschäftigen musst, bevor die Datenbank implementiert werden kann.

1. In der Klasse *Person* muss das Attribut *Adresse* weiter untergliedert werden. Somit kann man in der Datenbank gezielt nach Personen suchen, die in Königsbrunn wohnen.
2. Bei vielen Personen ist es recht mühselig, immer wieder den gleichen Ortsnamen und die gleiche PLZ aufzuschreiben. Und was passiert, wenn du dich einmal verschrieben hast?
3. Du wirst festgestellt haben, dass einige Bücher von mehreren Autoren geschrieben wurden. In der oben dargestellten Klasse *Buch* kann dieser zweite Autor nicht erfasst werden.
4. Sinnvoll ist es, das Ausleihdatum zu erfassen und vom Computer den Rückgabetermin berechnen zu lassen. Du wirst jedoch festgestellt haben, dass das Ausleihdatum weder ein Attribut der Klasse *Buch* noch der Klasse *Person* ist.

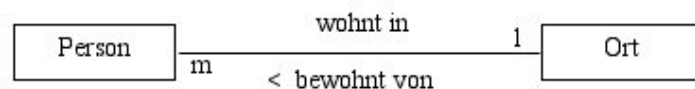
### 2.1.2 Erstellung des Klassendiagramms

Problem 1 ist recht einfach zu lösen. Das Attribut *Adresse* wird ersetzt durch die Attribute *Straße*, *PLZ* und *Ortsname*.



**Abbildung 2.3:** Erweitertes Modell der Klasse *Person* (1. Normalform)

Problem 2 ist etwas komplizierter zu lösen. Sehr wahrscheinlich musstest du in Übung 2.1 häufig den Namen „Königsbrunn“ schreiben, was an sich schon lästig ist. Nimm an, du hättest dich bei der Dateneingabe einer bestimmten Person verschrieben, also an Stelle von „Königsbrunn“ den falschen Ortsnamen „Känigsbrunn“. Diese Person wird in einer Auflistung aller Königsbrunner nicht mehr erscheinen. Und in einer großen Datenbank mit tausenden Einträgen wird diese Person kaum noch zu finden sein. Sinnvoll ist es deshalb, die Ortsnamen nicht als Attribut der Klasse *Person* zu führen, sondern eine eigenständige Klasse *Ort* zu erstellen.



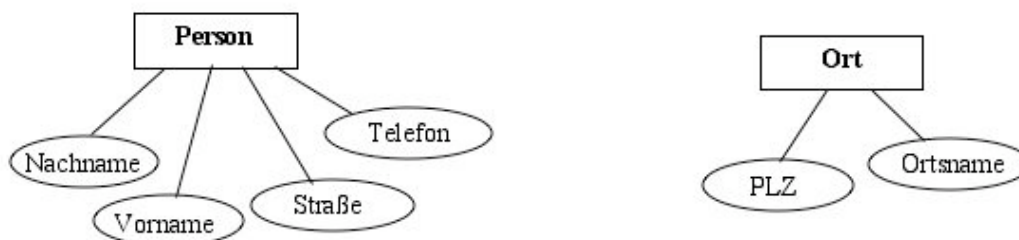
**Abbildung 2.4:** Beziehung zwischen *Person* und *Ort* (3. Normalform)

Wie Abbildung 2.4 darstellt, besteht zwischen den Klassen *Person* und *Ort* eine 1:m-Beziehung.

Jedes Objekt der Klasse *Person* wohnt in genau einem (**1**) Ort.

Jedes Objekt der Klasse *Ort* wird bewohnt von einer oder mehreren (**m**) Personen.

Solche 1:m-Beziehungen sind in einer relationalen Datenbank recht einfach zu implementieren, wie du in Kapitel 2.2.2 *Implementierung der 1:m-Beziehung* erfahren wirst. Du hast nun die ursprünglich Klasse *Person* in zwei Klassen *Person* und *Ort* aufgespaltet. Abbildung 2.5 zeigt die zugehörigen Attribute.



**Abbildung 2.5:** Die Klassen *Person* und *Ort*

Problem 3 verdeutlicht, dass auch die Klasse *Buch* verändert werden muss. Hier liegt ein ähnliches Problem vor in der Datenbank *PersonSprichtSprache* (vgl. Kapitel 1: *Vertraut werden und Probleme erkennen*). Abbildung 2.6 zeigt die Beziehung zwischen den Klassen *Buch* und *Autor*.

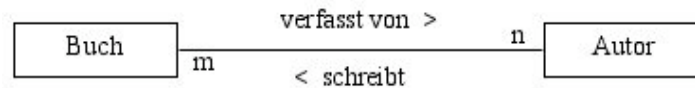


Abbildung 2.6: Beziehung zwischen *Buch* und *Autor*

Zwischen den Klassen *Buch* und *Autor* besteht eine n:m-Beziehung.:

Jedes Objekt der Klasse *Buch* kann von einem oder mehreren (**n**) Autoren verfasst worden sein.

Jedes Objekt der Klasse *Autor* kann ein oder mehrere (**m**) Bücher geschrieben haben.

Solche n:m-Beziehungen lassen sich nicht direkt in einer relationalen Datenbank implementieren. Vielmehr muss diese durch Definition einer neuen Klasse, beispielsweise *Schreiben*, in zwei 1:m-Beziehungen übergeführt werden.

### Übung 2.2:

Studiere noch einmal das Kapitel 1.3 *PersonSprichtSprache03*.

- Erstelle eine Beziehung zwischen den Klassen *Schueler* und *Sprache* ähnlich wie in Abbildung 2.6 dargestellt.
- Vervollständige die unten dargestellte Beziehung *Buch verfasstVon Autor*, indem du alle Pfeile zwischen den Klassen *Buch*, *Schreiben* und *Autor* einzeichnest.



Um das Problem *Buch verfasstVon Autor* zu lösen, hast du eine Zuordnungstabelle *tblSchreiben* erstellt, die als Datensätze die Zahlenpaare (*BuchNr* / *AutorNr*) enthält:

Ein bestimmtes Buch hat ein oder mehrere (**n**) Datensätze in der Zuordnungstabelle.

Ein bestimmter Datensatz der Zuordnungstabelle gehört zu genau einem (**1**) Buch.

Ein bestimmter Autor hat ein oder mehrere (**n**) Datensätze in der Zuordnungstabelle.

Ein bestimmter Datensatz in der Zuordnungstabelle gehört zu genau einem (1) Autor.

Somit hast du die  $n:m$ -Beziehung aufgelöst in zwei  $1:n$ -Beziehungen, die sich nun in einer relationalen Datenbank recht einfach implementieren lassen wie du in Kapitel 2.2.2 *Implementierung der 1:m-Beziehung* erfahren wirst..

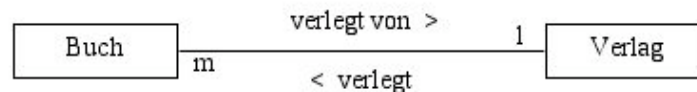
Abbildung 2.7 zeigt die Klassen *Buch* und *Autor* mit ihren zugehörigen Attributen.



**Abbildung 2.7:** Die Klassen *Buch* und *Autor*

### Bemerkung:

Eine genauere Untersuchung der Klasse *Buch* zeigt noch folgende Beziehung

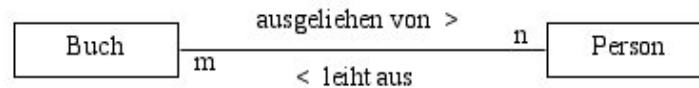


**Abbildung 2.8:** Beziehung zwischen *Buch* und *Verlag*

Ein bestimmtes Buch wird von genau einem (1) Verlag verlegt.  
Ein bestimmter Verlag verlegt ein oder mehrere (m) Bücher.

Du hast hier eine ähnliche Situation wie in der Beziehung zwischen *Person* und *Ort* (vgl. Abbildung 2.4) und könntest eine separate Klasse *Verlag* erstellen. Ich will in diesem Fall davon absehen, weil diese Klasse hier nur ein einzelnes Attribut *Verlagsname* hätte. Eine ganz andere Situation liegt jedoch vor, wenn du von diesem Verlag noch zusätzlich Adresse, Telefon, Ansprechpartner, usw. verwalten willst.

Nun bleibt noch das letzte Problem 4 mit der maximalen Ausleihzeit zu lösen. Das Datum spielt erst beim Vorgang des Ausleihens eine Rolle. Hier hilft es, die Beziehung zwischen diesen beiden Klassen *Buch* und *Person* genauer untersuchen.



**Abbildung 2.9:** Beziehung zwischen *Buch* und *Person* (2. Normalform)

Auch hier besteht eine n:m-Beziehung.

Jedes Buch kann von einem oder mehreren (**n**) Personen ausgeliehen werden (zwischenzeitliche Rückgabe vorausgesetzt).

Jede Person kann ein oder mehrere (**m**) Bücher ausleihen.

Du wirst diese n:m-Beziehung in zwei 1:n-Beziehungen auflösen müssen, indem du eine Zuordnungstabelle erstellst. Neben den Attributen *BuchNr* und *PersonNr* enthält diese zusätzlich noch das Attribut *Datum*.

### Übung 2.3:

Simuliere den Ausleihprozess auf einem Blatt Papier. Erstelle dazu ähnlich wie in Übung 2.2 die Beziehung *Buch ausgeliehetVon Person* in drei Tabellen dar.

Nun können diese verschiedenen Komponenten der Datenbank in einem Klassendiagramm dargestellt werden. Die einzelnen Klassen werden mit Linien verbunden, die die Beziehungen zwischen den einzelnen Objekten der jeweiligen Klassen erklären. Nicht dargestellt wird in diesem Klassendiagramm, dass in einer relationalen Datenbank jede n:m-Beziehung in zwei 1:m-Beziehungen aufgelöst werden muss, was jeweils zu einer weiteren Zuordnungstabelle führt.



**Abbildung 2.10:** Klassendiagramm der Datenbank *Buchausleihe*

Eine der wichtigsten Aufgaben des Klassendiagramms ist es zu zeigen, welche Beziehungen zwischen den Objekten einer Klasse bestehen. Es wird prinzipiell zwischen drei Arten von Beziehungen unterschieden.

**1:1-Beziehung:** Einem Objekt vom Typ A ist genau ein Objekt vom Typ B zugeordnet.

Bemerkung: relativ unwichtig

**1:m-Beziehung:** Jedes Objekt vom Typ A kann mit beliebig vielen Objekten des Typs B in Beziehung stehen.

Jedes Objekt vom Typ B steht mit genau einem Objekt vom Typ A in Beziehung.

Bemerkung: m steht für *multiple* oder *many*

**m:n-Beziehung:** Jedes Objekt vom Typ A bzw. vom Typ B kann mit beliebig vielen Objekten des jeweilig anderen Typs in Beziehung stehen.

Bemerkung: muss in zwei 1:m-Beziehungen zerlegt werden

## 2.2 Die 1:m-Beziehung

### 2.2.1 Das Schlüsselkonzept

In einem relationalen Datenbanksystem werden die Objekte einer Klasse in Tabellen zusammengefasst. Die Zeilen einer Tabelle repräsentieren die Objekte, während die Tabellenspalten die Attributwerte dieser Objekte enthalten.

Eine Tabelle hat folgende Merkmale:

1. Jede Tabelle hat einen eindeutigen Namen. In der Regel wird hier der Name der Klasse verwendet.
2. Jede Spalte der Tabelle hat einen eindeutigen Namen. In der Regel wird hier der Name des entsprechenden Attributs verwendet.
3. Jeder Spalte wird ein eindeutiger Wertebereich zugeordnet, z.B. Zahl, Text, usw.
4. Die Reihenfolge der Spalten und Zeilen spielt keine Rolle.
5. Der Kreuzungspunkt von Spalten und Zeilen darf nur einen einzigen Attributwert enthalten.
6. Es existieren keine zwei Zeilen, die identische Werte haben. Die Objekte müssen sich also durch eine eindeutige Identifikation (Schlüssel) unterscheiden lassen.

Der Punkt 6 gibt dir einen Hinweis, wie das Klassenmodell in ein Datenbank-Programm umgesetzt werden kann. Diese Tabellen müssen zueinander in Beziehung gesetzt werden. Dazu benötigt jede Tabelle eine spezielle Kategorie von Attributen, die **Schlüsselattribute**. Im Prinzip kann jedes Attribut einer Tabelle als Schlüsselattribut verwendet werden. Voraussetzung ist jedoch, dass die Werte des speziellen Attributs eindeutig sind, d.h. dass jeder Wert dieses Schlüsselattributs nur einmal in der Gesamtmenge vorkommt.

Unter der Voraussetzung, dass beispielsweise in der Klasse 9a keine Schüler existieren, die einen gleichen Nachnamen haben, könnte das Attribut *Nachname* als Schlüsselattribut verwendet werden. Aber jeder wird wohl einsehen, dass *Nachname* kein sinnvolles Schlüsselattribut ist. Du könntest auch die Kombination der Attribute *Nachname* / *Vorname* als Primärschlüssel verwenden. Aber bei einer genügend großen Anzahl von Schülern lässt sich auch nicht ausschließen, dass zwei verschiedene Schüler den gleichen Namen besitzen.

PersonID	Nachname	Vorname	Straße	Telefon
0	Drewnick	Robert	Haferstr. 12	0821/65897
1	Kerpl	Denise	Buchenstr. 25b	08231/5474
2	Haag	Florian	Eichenstr. 2	08231/8965
3	Schwaner	Nico	Mittelfeldstr. 6	08232/87891
4	Haag	Sebastian	Eichenstr. 2	08231/8965
5	Hermann	Sandra	Weizenstr. 5	08231/6352

Abbildung 2.11: Der Primärschlüssel *PersonID*

Sinnvoll ist deshalb, ein neues Attribut *PersonID* einzuführen mit dem Datentyp **Integer**, dessen **AutoWert** auf **Ja** gesetzt wird, der automatisch und genau einmal vergeben wird. Somit ist in der Klasse *Person* das Attribut *PersonID* der **Primärschlüssel** und jedes Objekt dieser Klasse kann eindeutig identifiziert werden.

OrtID	Ortsname	PLZ
0	Königsbrunn	86343
1	Bobingen	86322
2	Haunstetten	86729
3	Oberottmarshausen	86343

Abbildung 2.12: Der Primärschlüssel *OrtID*

Da Königsbrunn und Oberottmarshausen dieselbe *PLZ* besitzen, sollte auch hier das neue Attribut *OrtID* als Primärschlüssel eingeführt werden.

Die Personen Kerpl und Hermann wohnen in Königsbrunn, die beiden Geschwister Haag in Haunstetten, Drewnick in Oberottmarshausen und Schwaner in Bobingen. Du musst nun die beiden Tabellen in Beziehung miteinander bringen. Um zwei Tabellen miteinander zu verknüpfen, muss eine Tabelle darüber hinaus über einen **Fremdschlüssel** verfügen. Ein Fremdschlüssel definiert ein Attribut, das in einer anderen Tabelle als Primärschlüssel vorkommt. Wurde der zugehörige Primärschlüssel als AutoWert definiert, sollte der Fremdschlüssel den Datentyp **Integer** besitzen, dessen **AutoWert** auf **Nein** gesetzt wird.

Die Klasse *Person* erhält somit ein weiteres Attribut *OrtNr*, in dem der Wert der entsprechenden *OrtID* gespeichert wird.

PersonID	Nachname	OrtNr
0	Drewnick	3
1	Kerpl	0
2	Haag	2
3	Schwaner	1
4	Haag	2
5	Hermann	0

OrtID	Ortsname
0	Königsbrunn
1	Bobingen
2	Haunstetten
3	Oberottmarshausen

Abbildung 2.13: Attribut *OrtNr* verknüpft die beiden Tabellen

Wie Abbildung 2.13 zeigt, besitzt

- die Klasse *Person* den Primärschlüssel *PersonID*,
- die Klasse *Ort* den Primärschlüssel *OrtID* und
- die Klasse *Person* den Fremdschlüssel *OrtNr*.

#### Bemerkung:

Wie bereits eingeführt, werde ich den Primärschlüssel mit **ID** und den Fremdschlüssel mit **Nr** kennzeichnen. Dies ist nicht zwingend vorgeschrieben, erleichtert jedoch die Beziehungen zwischen den Tabellen.

### 2.2.2 Implementierung der 1:m-Beziehung

An Hand der folgenden Übung wirst du lernen, die beiden Tabellen *tblPerson* und *tblOrt* in Star/NeoOffice zu implementieren.

#### Übung 2.4:

- a) Erstelle in Star/NeoOffice eine neue Datenbank *Buchausleihe*.
- b) Anschließend erstelle die Tabelle *tblPerson* wie in Abbildung 2.14 gezeigt. Setze den *AutoWert* auf Ja für das Primärschlüssel-Attribut *PersonID* und den *AutoWert* auf Nein für das Fremdschlüssel-Attribut *OrtNr*.
- c) Erstelle auf analoge Weise die Klasse *Ort*.
- d) Im Fenster „Relationenentwurf“ verbindest du den Primärschlüssel *OrtID* und Fremdschlüssel *OrtNr* miteinander. (vgl. Abbildung 2.15)

The image shows two screenshots of the NeoOffice Base 'Tabellenentwurf' (Table Design) window. The left window is for 'tblPerson' and the right is for 'tblOrt'. Both windows show a table with columns for 'Feldname', 'Feldtyp', and 'Beschreibung'. In 'tblPerson', 'PersonID' is the primary key (Primärschlüssel) and 'OrtNr' is a foreign key (Fremdschlüssel). In 'tblOrt', 'OrtID' is the primary key. Below the table design, the 'Feldeigenschaften' (Field Properties) section is visible, showing settings like 'Auto-Wert', 'Eingabe erforderlich', 'Länge', 'Defaultwert', and 'Format-Beispiel'.

Feldname	Feldtyp	Beschreibung
PersonID	Integer [ INTEGER ]	Primärschlüssel
Nachname	Text [ VARCHAR ]	
Vorname	Text [ VARCHAR ]	
Straße	Text [ VARCHAR ]	
Telefon	Text [ VARCHAR ]	
OrtNr	Integer [ INTEGER ]	Fremdschlüssel

Feldname	Feldtyp	Beschreibung
OrtID	Integer [ INTEGER ]	Primärschlüssel
Ortsname	Text [ VARCHAR ]	
PLZ	Text [ VARCHAR ]	

Abbildung 2.14: Entwurf der Klassen *Person* und *Ort*

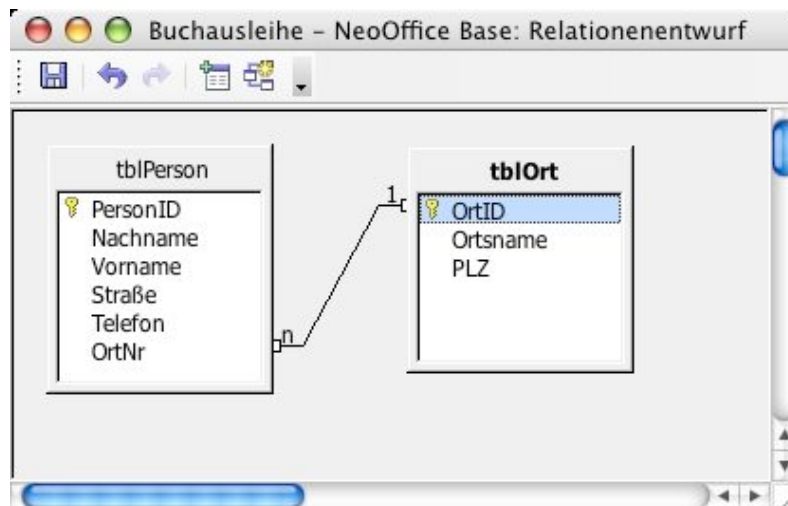


Abbildung 2.15: Die Klassen *Person* und *Ort* im Fenster „Beziehungen“

### 2.2.3 Einfache SQL-Abfragen

Du wirst nun wieder einige Abfragen an deine Datenbank stellen. Dazu musst du allerdings zuerst diese mit Daten füllen.

#### Übung 2.5:

Fülle die Tabellen *tblPerson* und *tblOrt* mit geeigneten Datensätzen. Du kannst die Vorschläge aus den Abbildungen 2.11, 2.12 und 2.13 übernehmen und weitere Mitglieder deiner Klasse in die Datenbank aufnehmen.

#### Übung 2.6:

Hinweis: Studiere noch einmal die Abfragen, die du bereits in Kapitel 1 „Vertraut werden und Problem erkennen“ durchgeführt hast.

- Suche Nachname und Vorname aller Personen.
- Suche Nachname und Vorname aller Personen sortiert nach Nachname.
- In welcher Straße wohnt Drewnick?
- In welchem Ort wohnt Hermann?
- Suche alle Personen, die in Bobingen wohnen.
- Suche alle Personen, die nicht in Königsbrunn wohnen.
- Wie viele Personen kommen aus Haunstetten?

## 2.3 Die n:m-Beziehung

### 2.3.1 Das Schlüsselkonzept

Das Klassendiagramm in Abbildung 2.10 zeigt, dass zwischen den Klassen *Buch* und *Person* ein n:m-Beziehung besteht. Solche Beziehungen lassen sich in einer relationalen Datenbank nicht direkt realisieren, da nur 1:n-Beziehungen erlaubt sind. Aber wie bereits schon erwähnt, ist es möglich, eine:n:m-Beziehung in zwei 1-n-Beziehungen zu zerlegen. Und hierbei spielen der Primärschlüssel und der Fremdschlüssel wieder eine wesentliche Rolle.

BuchID	Titel
0	Access 2000
1	Java lernen mit BlueJ
2	MySQL
3	Datenbanksysteme

PersonID	Nachname
0	Drewnick
1	Kerpl
2	Haag
3	Schwamer
4	Haag
5	Hermann

Abbildung 2.16: Primärschlüssel *BuchID* und *PersonID*

Die Person Drewnick hat sich nun die beiden Bücher Access 2000 und MySQL ausgeliehen und die Person Hermann das Buch Datenbanksysteme. Mit Hilfe einer Kombination der beiden Primärschlüssel *BuchID* und *PersonID* kannst du diese Ausleihprozesse realisieren. In einer neuen Tabelle sammelst du die Kombinationen (1/1), (3/1) und (4/6).

BuchID	Titel
0	Access 2000
1	Java lernen mit BlueJ
2	MySQL
3	Datenbanksysteme

BuchNr	PersonNr
0	0
2	0
3	5

PersonID	Nachname
0	Drewnick
1	Kerpl
2	Haag
3	Schwamer
4	Haag
5	Hermann

**Abbildung 2.17:** Neue Tabelle realisiert die Ausleihe

Du erstellst eine neue Klasse *Leihen*. Diese besitzt die Attribute *BuchNr* und *PersonNr*. Die Bezeichnung dieser Attribute deutet bereits an, dass es sich um Fremdschlüssel handelt, die sich auf die entsprechenden Primärschlüssel der Klassen *Buch* und *Person* beziehen. Allerdings benötigt auch die Klasse *Leihen* einen Primärschlüssel. Weder *BuchNr* noch *PersonNr* identifizieren einen Datensatz eindeutig. Nur die Kombination aus *BuchNr* und *PersonNr* kann somit als Primärschlüssel gewählt werden.

Wie Abbildung 2.17 zeigt, besitzt

- die Klasse *Buch* den Primärschlüssel *BuchID*,
- die Klasse *Person* den Primärschlüssel *PersonID*,
- die Klasse *Leihen* als Primärschlüssel die Kombination *BuchNr/personNr*,
- die Klasse *Leihen* den Fremdschlüssel *BuchNr* und
- die Klasse *Leihen* den Fremdschlüssel *PersonNr*

Die Informationsstruktur der Datenbank *Buchausleihe* erfordert noch ein weiteres Attribut *Datum* in der Klasse *Leihen*. Hiermit könnte später das Rückgabedatum, die Ausleihzeit, Überziehungsgebühr, usw. berechnet werden.

### 2.3.2 Implementierung der n:m-Beziehung

An Hand der folgenden Übung wirst du lernen, die beiden zusätzlichen Tabellen *tblBuch* und *tblLeihen* in der Datenbank *Buchausleihe* zu implementieren.

#### Übung 2.7:

- a) Öffne in Star/NeoOffice die Datenbank *Buchausleihe*.
- b) Anschließend erstelle die Tabelle *tblBuch* wie in Abbildung 2.14 gezeigt. Setze den *AutoWert* auf Ja für das Primärschlüssel-Attribut *BuchID* (vgl. Abbildung 2.18)
- c) Erstelle nun die Klasse *Leihen*. Hier musst du *BuchNr* und *PersonNr* **gleichzeitig** markieren und sie zum Primärschlüssel machen. (vgl. Abbildung 2.18)
- d) Im Fenster „Relationenentwurf“ verbindest du die entsprechenden Primär- und Fremdschlüssel miteinander. (vgl. Abbildung 2.19)

Feldname	Feldtyp	Beschreib...
BuchID	Integer [ INTEGER ]	Primärschlüssel
Titel	Text [ VARCHAR ]	
Verlag	Text [ VARCHAR ]	
ISDN	Text [ VARCHAR ]	

Feldname	Feldtyp	Beschreib...
BuchNr	Integer [ INTEGER ]	Primärschlüssel
PersonNr	Integer [ INTEGER ]	
Datum	Datum [ DATE ]	

Abbildung 2.18: Entwurf der Klassen *Buch* und *Leihen*

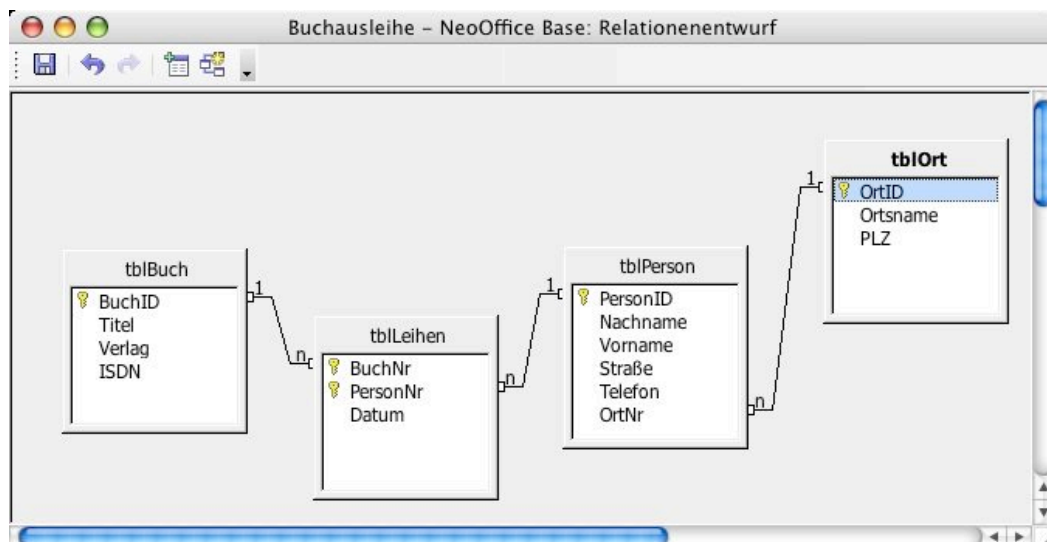


Abbildung 2.19: Die Klassen *Buch*, *Leihen*, *Person* und *Ort* im Fenster „Relationenentwurf“

### 2.3.3 Einfache SQL-Abfragen

Du wirst nun wieder einige Abfragen an deine Datenbank stellen. Dazu musst du allerdings zuerst diese mit Daten füllen.

#### Übung 2.8:

Fülle die Tabellen *tblBuch* und *tblLeihen* mit geeigneten Datensätzen. Du kannst die Vorschläge aus den Abbildungen 2.16 und 2.17 übernehmen und ergänzen. Zusätzlich solltest du weitere Bücher hinzufügen. Es sollten jedoch nicht alle Bücher ausgeliehen worden sein und nicht jede Person leiht sich ein Buch aus.

### Übung 2.9:

Hinweis: Studiere noch einmal die Abfragen, die du bereits in Kapitel 1 „Vertraut werden und Problem erkennen“ durchgeführt hast.

- Suche Titel aller Bücher.
- Suche Titel und Verlage aller Bücher, sortiert nach Verlage.
- Welche Bücher hat Hermann ausgeliehen?
- Welche Personen haben wann das Buch Access 2000 ausgeliehen?
- Wie oft wurde das Buch ACCESS 2000 ausgeliehen?
- In welchen Orten und wann war das Buch Access 2000?

## 2.4 Datenbank Buchausleihe ist komplett

Die Abbildung 2.10 zeigt noch die Klasse *Autor*, die über eine n:m-Beziehung mit der Klasse *Buch* in Verbindung steht.

### Übung 2.10:

Erstelle die Tabellen *tblBuch* und *tblSchreiben* und fülle diese mit geeigneten Datensätzen. (vgl. Abbildung 2.20)

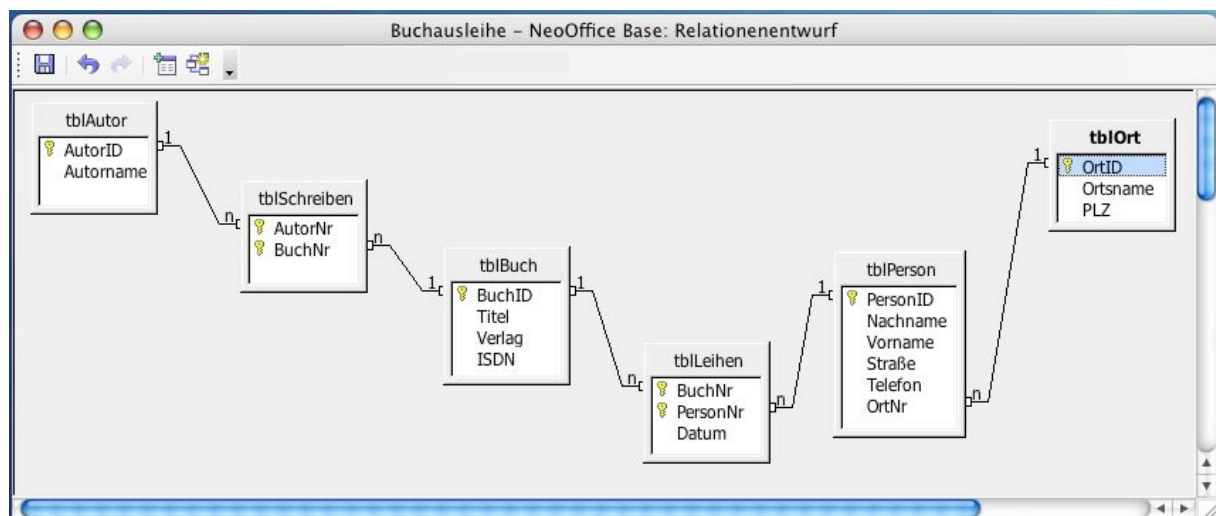


Abbildung 2.20: Die Klassen von *Buchausleihe* im Fenster „Relationenentwurf“

Du erkennst die Klassen *Autor*, *Buch*, *Person* und *Ort* wieder. Star/NeoOffice zeigt zusätzlich alle Attribute dieser Klassen. Die Primärschlüssel sind fett gedruckt.

### Übung 2.11:

Versuche selbstständig, einige SQL-Abfragen mit deinem Nachbarn zu erstellen. Komplizierte Abfragen wirst du allerdings erst im Kapitel *Abfragen mit SQL* kennen lernen.

**Bemerkung:**

Weitere Informationen hierzu findest du im Kapitel *Normalisierung*. Hier wird diese Datenbank *Buchausleihe* unter dem Gesichtspunkt der Normalformen noch einmal untersucht.