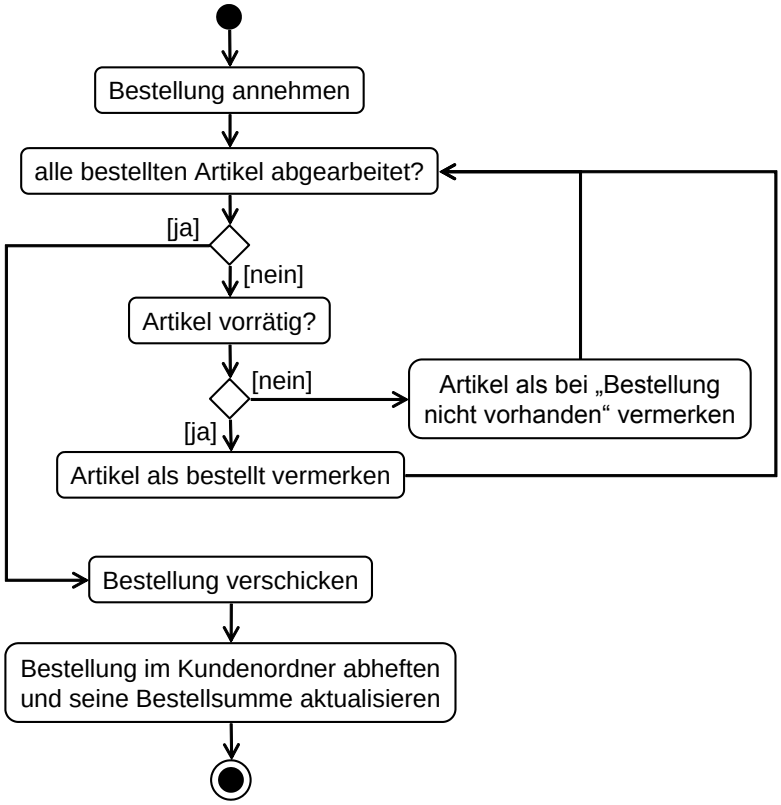
A Lösungen zu den Aufgaben

Die folgenden Lösungen zeigen jeweils eine Lösungsmöglichkeit. Häufig sind andere bzw. leicht abgewandelte Lösungen möglich.

Lösungen zu Kapitel 1

zu 1.: Für jeden Kunden und jeden Artikel gibt es jeweils eine Akte, der Ablauf kann dann wie folgt aussehen:



zu 2.: Grundsätzlich ist es wichtig, ein Dateisystem zu entwickeln und sich Strukturen für Dateinamen auszudenken, so dass die Ergänzung und Veränderung der Da-

ten erleichtert wird. Sinnvoll kann es sein, für jeden Kunden einen Ordner einzurichten und im Namen der Datei einige Kundeninformationen, wie den Kundennamen oder die Kundennummer, das Bestelldatum und den Zustand der Abarbeitung zu vermerken. Ein Dateiname könnte z. B. `24348_051223_komplett.txt` sein.

Typische Probleme sind trotzdem:

- Informationen über einen Artikel zu erhalten, z. B. wie oft wurde er verkauft und zurück gesandt.
- die Änderung des Namens eines Artikels oder der Kundenadresse, wenn alle Dokumente konsistent bleiben sollen.
- die Unsicherheit, was passiert, wenn zwei Nutzer die Datei gleichzeitig nutzen

zu 3.:

	extern	logisch	physisch	Anmerkung
a.	X			
b.			X	evtl. kleine Änderungen in anderen Ebenen
c.		X		
d.		X		
e.	X			
f.				keine Änderungen, da Standardfunktionaliät
g.			X	(*)
h.				wie f.
i.			X	(*)
j.			X	(*)

(*) eventuell ist auch die externe Ebene betroffen. Wenn sich die Speicheradresse oder Zugangsinformationen ändern, müssen diese Verbindungsdaten angepasst werden. Sonst findet keine Änderung im externen Modell statt.

zu 4.: Hier wird ein objektorientierter Ansatz gewählt, es werden die Methoden angegeben, die eine Datenbankklasse realisieren müsste.

```
public interface Datenbank {
  public void tabelleAnlegen( String tabellenname, String spalte1,
      String spalte2, String spalte3);
  public void datumEintragen( String tabellenname,
      String spalte1Wert, String spalte2Wert, String spalte3Wert);
  public void datumLoeschen( String tabellenname,
      String spalte1Wert, String spalte2Wert, String spalte3Wert);
  public void datumAendern( String tabellenname,
      String altSp1Wert, String altSp2Wert, String altSp3Wert,
```

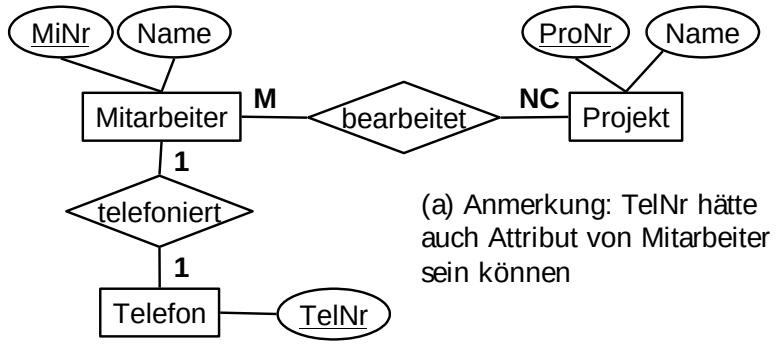
```
      String neuS1Wert, String neuSp2Wert, String neSp3Wert);
  public Collection datenLesen( String tabellenname,
      int spaltennummer, String spaltenwert);
}
```

Beim Löschen oder Ändern muss ein aktueller Eintrag aufgeführt werden. Falls es mehrere identische Einträge gibt, stellt sich die Frage, ob ein Eintrag oder alle Einträge gelöscht werden. Damit diese Frage nicht aufkommt, sollte diese Situation in der Realität vermieden werden. Wenn Daten ausgelesen werden, muss die Antwort flexibel sein. In der Lösung, kann man für eine Tabelle eine Spaltennummer und einen konkreten Wert dieser Spalte angeben, und man erhält alle Einträge, die genau diesen Wert in dieser Spalte haben. Zur Rückgabe dieses Ergebnisses wird eine flexible Datenstruktur benötigt.

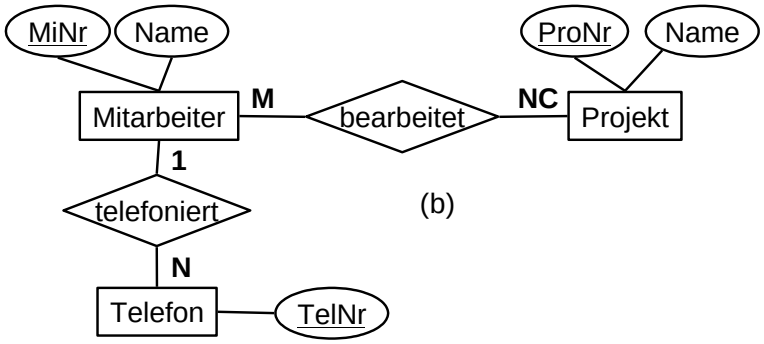
Weitere Funktionalität kann ergänzt werden, dabei muss man auf die sinnvolle Wahl der Datenstrukturen achten. Schwierig ist die Umsetzung des Mehrnutzerbetriebs, da die Software zunächst nur auf einen Nutzer ausgelegt ist. Man muss einen Weg finden, dass die Datenbank als Server verschiedene Verbindungen mit Nutzern aufbauen kann.

Lösungen zu Kapitel 2

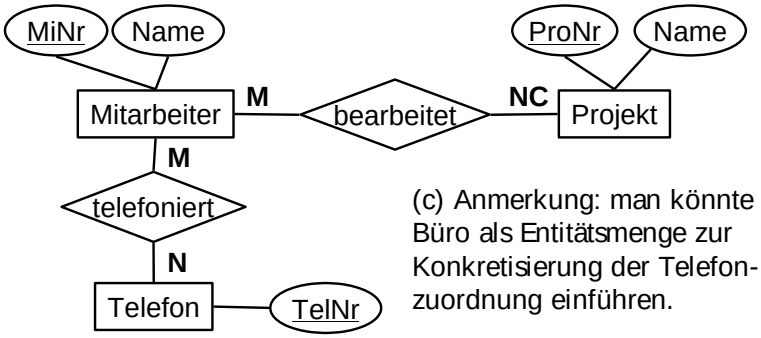
zu 1.:



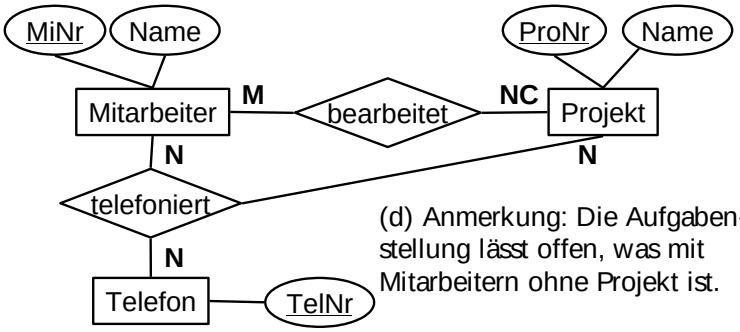
(a) Anmerkung: TelNr hätte auch Attribut von Mitarbeiter sein können



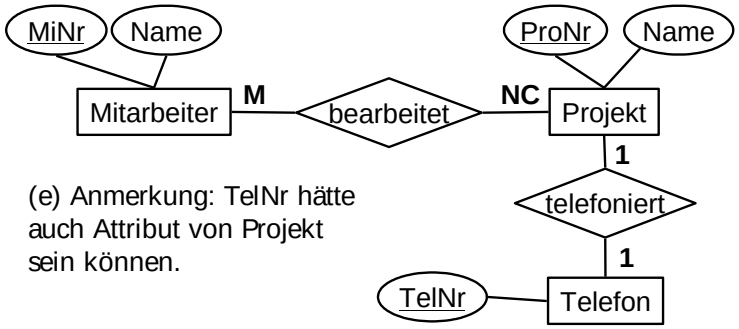
(b)



(c) Anmerkung: man könnte Büro als Entitätsmenge zur Konkretisierung der Telefonzuordnung einführen.

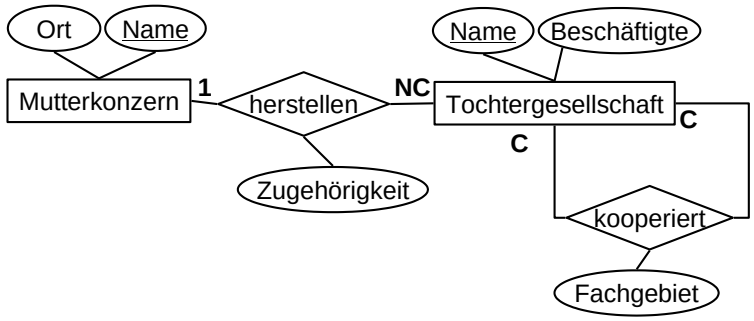


(d) Anmerkung: Die Aufgabenstellung lässt offen, was mit Mitarbeitern ohne Projekt ist.

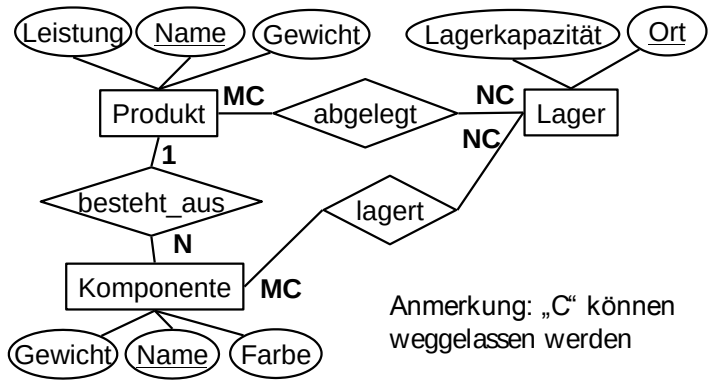


(e) Anmerkung: TelNr hätte auch Attribut von Projekt sein können.

zu 2.:

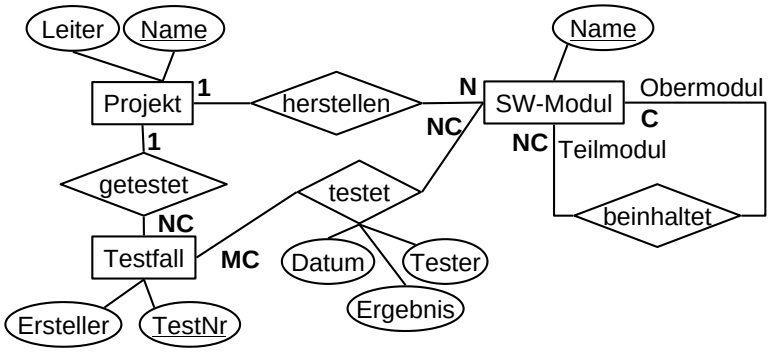


zu 3.:



Anmerkung: „C“ können weggelassen werden

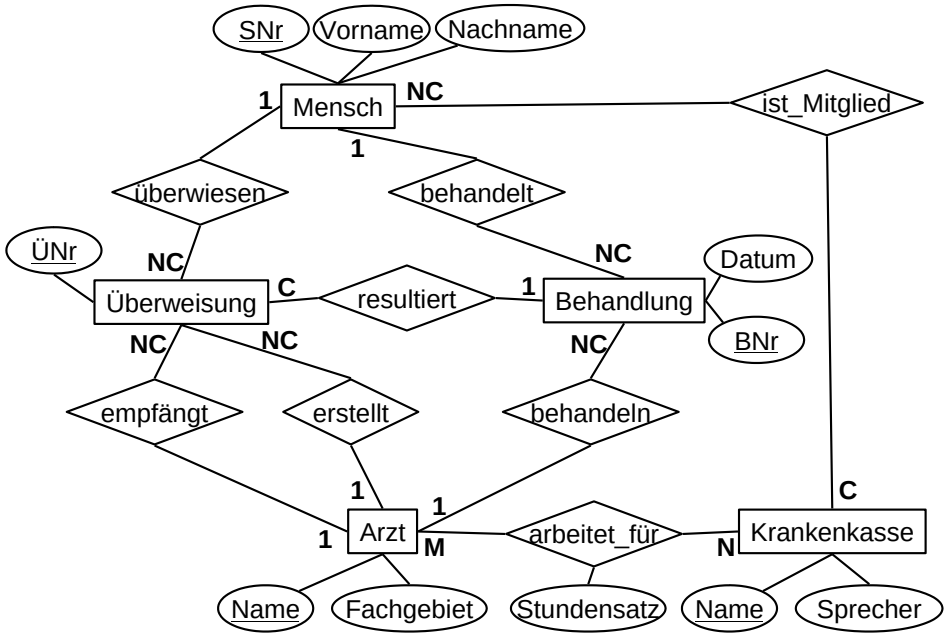
zu 4.:



zu 5.:

Anmerkung: Bei möglichen alternativen Lösungen muss überprüft werden, ob wirklich Menschen mehrfach vom gleichen Arzt behandelt und an unterschiedliche Ärzte überwiesen werden können.

Ob N oder NC als Kardinalität genutzt werden soll, ist diskutabel.



Lösungen zu Kapitel 3

zu 1.: Hinweis: Es wird die Schreibweise „Tabellename: Liste der Attribute, getrennt jeweils durch einen senkrechten Strich“ verwandt. Identifizierende Attribute sind unterstrichen.

Stadt: Name | Breite | Länge | Name_Land

Land: Einwohner | Name | Breite | Länge

Organisation: Name | Gründungsdatum

Mitgliedschaft (aus Mitglied von): NameLand | NameOrganisation | Status

Residiert (aus beheimatet): Breite | Länge | Name

Grenze (aus grenzt an): NameLand1 | NameLand2 | Länge

Die Kardinalitäten von „beheimatet“ sollten geändert werden: Jede Stadt kann beliebig viele Organisationen beheimaten. Jede Organisation ist in einer oder mehreren Städten beheimatet.

zu 2.:

für x=1, y=1: A: A1 | A2 | B1 | B2 | B3 | R1 | R2

für x=1, y=N oder y=C oder y=NC:

A: A1 | A2 | B1 | B2 | R1 | R2

B: B1 | B2 | B3

für y=1, x=N, oder x=C oder x=NC

A: A1 | A2

B: B1 | B2 | B3 | A1 | R1 | R2

sonstige Fälle:

A: A1 | A2

B: B1 | B2 | B3

REL: A1 | B1 | B2 | R1 | R2

Lösungen zu Kapitel 4

zu 1.: Bei beiden Nummern handelt es sich um Schlüsselkandidaten. Hinter der 46 kann z. B. der Mitarbeiter Heinz Meier stehen, der durch die Aussage zur 23, das ist z. B. die Herrenunterbekleidungsabteilung, kommen soll, um einen Kunden zu bedienen.

zu 2.: Im Kapitel wurde mit der Tabelle Student gezeigt, dass eine Tabelle zwei unterschiedlich große Schlüsselkandidaten haben kann. Würde der Satz gelten, hätte die Tabelle nur den einelementigen Schlüsselkandidaten. Schlüsselkandidaten müssen nur in sich minimal sein, es findet kein Vergleich der Schlüsselkandidaten untereinander statt.

zu 3.: Eine Tabelle mit drei Attributen A, B und C kann maximal einen Schlüsselkandidaten mit drei Elementen {A, B, C} oder drei Schlüsselkandidaten mit zwei Elementen {A, B}, {A, C} und {B, C} oder drei Schlüsselkandidaten mit einem Element {A}, {B} und {C} haben.

zu 4.: Folgende Rechenregeln können für die volle funktionale Abhängigkeit übernommen bzw. angepasst werden, dabei soll der Pfeil in diesem Fall vollständige funktionale Abhängigkeit bedeuten:

aus $A \rightarrow B \cup C$ folgt $A \rightarrow B$ und $A \rightarrow C$

seien A,B,C einelementig: aus $A \rightarrow B$ und $B \rightarrow C$ folgt $A \rightarrow C$ (*)

aus $A \rightarrow B$ und $A \rightarrow C$ folgt $A \rightarrow B \cup C$

(*) Man beachte, dass vollständige funktionale Anhängigkeit nicht transitiv ist, wie das folgende Beispiel zeigt.

A	B	C	D	E
10	7	17	3	20
42	1	43	41	84
12	5	17	7	24

In der Tabelle wird der Wert der Spalte C aus A+B und der Wert der Spalte D aus A-B berechnet. Damit gilt die volle funktionale Abhängigkeit {A,B} $\rightarrow$ {C,D}. Der Wert von E wird aus C+D berechnet, es gilt die volle funktionale Abhängigkeit {C,D} $\rightarrow$ {E}. Es gilt nicht die volle funktionale Abhängigkeit {A,B} $\rightarrow$ {E}, da man E aus 2*A berechnen kann und damit {A} $\rightarrow$ {E} gilt.

zu 5.:

zu a) {AuftragsNr, ProduktNr}

zu b) Umformung in die zweite Normalform ergibt:

Auftragsliste: <u>AuftragsNr</u> | <u>ProduktNr</u>

Auftrag: <u>AuftragsNr</u> | Eingangsdatum

Produkt: <u>ProduktNr</u> | ProduktName | Preis | Lieferant

Tabelle Produkt ist nicht in dritter Normalform, umgeformt:

Produkt: ProduktNr | ProduktName

Lieferant: ProduktName | Preis | Lieferant

zu c) Die Umformung in die dritte Normalform ergäbe:

Produkt: ProduktNr | ProduktName | Lieferant

Lieferant: ProduktName | Lieferant | Preis

zu 6.:

zu a) kann aus II. mit I. transitiv geschlossen werden

zu b) {MatNr, Fach} und {TelefonNr, Fach}

zu c) Umformung in zweite Normalform:

Tabelle Note: MatNr | Fach | Note [statt MatNr könnte auch TelefonNr stehen]

Tabelle Studi: MatNr | Name | TelefonNr | Studium | Firma | Betreuer [zwei Schlüsselkandidaten {MatNr} und {TelefonNr}]

zu d) nur Tabelle Studi nicht in dritter Normalform, Umformung ergibt

Tabelle Studi: MatNr | Name | TelefonNr | Studium | Firma

Tabelle Betreuung: Firma | Betreuer

Eine Tabelle MatNr | TelefonNr ist nicht notwendig (nicht Ergebnis des Verfahrens), da es sich um Schlüsselattribute handelt (es kann so auch keine redundante Information in den Tabellen entstehen).

zu 7.:

zu a) {PNr} $\rightarrow$ {TelNr}

{TelNr} $\rightarrow$ {PNr}

{Projekt} $\rightarrow$ {PBudget}

{PNr, Projekt} $\rightarrow$ {Rolle}

zu b) Gegenbeispiel mit der 2. und 6. Zeile.

zu c) {PNr, Projekt, VAbt} und {TelNr, Projekt, VAbt}

zu d) Nicht in zweiter Normalform, resultierende Tabellen:

Projektmitarbeit: PNr | Projekt | VAbt | TelNr [statt PNr könnte TelNr im Schlüsselkandidaten gewählt werden]

Projekte: Projekt | PBudget

Projektrolle: PNr | Projekt | Rolle

zu e) Tabellen sind alle in 3NF.

zu f) Nicht in Boyce-Codd-Normalform, resultierende Tabellen:

Projektmitarbeit: PNr | Projekt | VAbt

Mitarbeiter: PNr | TelNr

Projekte: Projekt | PBudget

Projektrolle: PNr | Projekt | Rolle

zu 8.: Generell gilt die volle funktionale Abhängigkeit {ProNr, MiNr} → {Arbeit}. In den folgenden Lösungen sind zusätzlich geltende volle funktionale Abhängigkeiten angegeben.

zu a) {MiNr} → {TelNr} und {TelNr} -> {MiNr}, Schlüsselkandidaten {ProNr, MiNr} und {ProNr, TelNr}, Tabelle in zweiter und dritter Normalform, nicht in Boyce-Codd-Normalform, dann folgende Aufteilung:

Projektmitarbeit: PNr | MiNr | Arbeit

Mitarbeiter: MiNr | TelNr

Statt MiNr hätte auch TelNr in der ersten Tabelle stehen bleiben können. In der unteren Tabelle wäre dann TelNr Primärschlüssel.

zu b) {MiNr} → {TelNr}, Schlüsselkandidat {ProNr, MiNr}, Tabelle nicht in zweiter Normalform, Aufteilung, wie bei a) beschrieben.

zu c) {ProNr, MiNr} → {TelNr} und {TelNr} → {MiNr} sowie {TelNr} → {ProNr}, Schlüsselkandidaten {ProNr, MiNr} und {TelNr}, Tabelle in zweiter, dritter und Boyce-Codd-Normalform.

zu d) {ProNr, MiNr} → {TelNr}, Schlüsselkandidat {ProNr, MiNr}, Tabelle in zweiter, dritter und Boyce-Codd-Normalform.

zu e) {ProNr} -> {TelNr}, {TelNr} -> {ProNr}, Schlüsselkandidaten {ProNr, MiNr} und {TelNr ,MiNr}. Tabelle in zweiter und dritter Normalform, nicht in Boyce-Codd-Normalform, deshalb Aufteilung:

Projektmitarbeit: ProNr | MiNr | Arbeit

Projekt: ProNr | TelNr

ProNr und TelNr können in der Aufteilung auch vertauscht werden.

Lösungen zu Kapitel 5

zu 1.: Die resultierende Elementanzahl hängt davon ab, wie viele Elemente in beiden Relationen vorkommen. Da in Relationen keine doppelten Elemente vorhanden sind, kann das Ergebnis zwischen vier und acht Elemente haben.

zu 2.:

zu a) Proj(Aufgabe, [Arbeit])

zu b) Proj(Sel(Projekt×Aufgabe, Projekt.ProNr=Aufgabe.ProNr), [Name, Arbeit])

zu c) Proj(Sel(Aufgabe×Maschine, Aufgabe.AufNr=Maschine.AufNr

AND Aufgabe.Arbeit='knicken'),[Mname])

zu d) Proj(Sel(Projekt×Aufgabe×Maschine,

Projekt.ProNr=Aufgabe.ProNr

AND Aufgabe.AufNr=Maschine.AufNr), [Name,Mname])

zu e) Hier ist der Trick, dass zwei Informationen aus der Relation Aufgabe benötigt werden. Aus diesem Grund wird eine Kopie dieser Relation angelegt.

Proj(Sel(Projekt×Aufgabe×Ren(Aufgabe,A2), Projekt.ProNr=Aufgabe.ProNr

AND Projekt.ProNr=A2.ProNr

AND Aufgabe.AufNr<>A2.AufNr),[Name])

Lösungen zu Kapitel 6

zu1.:

```
CREATE TABLE Projekt(
 PrNr INTEGER,
 PrName VARCHAR(20),
 PrLeiter VARCHAR(8),
 PRIMARY KEY(PrNr)
);

CREATE TABLE Arbeitspaket(
 PrNr INTEGER NOT NULL,
 PakNr INTEGER,
 PakName VARCHAR(18) NOT NULL,
 PakLeiter VARCHAR(8) NOT NULL,
 PRIMARY KEY(PakNr),
```

```
 CONSTRAINT A1 FOREIGN KEY(PrNr) REFERENCES Projekt(PrNr)
    ON DELETE CASCADE,
 CONSTRAINT A2 CHECK(PakLeiter<>'Winzig'),
 CONSTRAINT A3 CHECK(NOT(PakName='Analyse')
                 OR NOT(PakLeiter='Hack')),
 CONSTRAINT A4 CHECK(NOT(PakName='Implementierung')
                 OR (PakLeiter='Mittel' OR PakLeiter='Hack'))
);

CREATE TABLE Arbeit(
 PakNr INTEGER,
 MiName VARCHAR(10),
 Anteil Number,
 PRIMARY KEY(PakNr,MiName),
 CONSTRAINT B1 FOREIGN KEY(PakNr)
    REFERENCES Arbeitspaket(PakNr) ON DELETE CASCADE
);

INSERT INTO Projekt VALUES(1,'Notendatenbank','Wichtig');
INSERT INTO Projekt VALUES(2,'Adressdatenbank','Wuchtig');
INSERT INTO Projekt VALUES(3,'Fehlzeitendatenbank','Wachtig');

INSERT INTO Arbeitspaket VALUES(1,1,'Analyse','Wichtig');
INSERT INTO Arbeitspaket VALUES(1,2,'Modell','Wuchtig');
INSERT INTO Arbeitspaket VALUES(1,3,'Implementierung','Mittel');
INSERT INTO Arbeitspaket VALUES(2,4,'Modell','Durch');
INSERT INTO Arbeitspaket VALUES(2,5,'Implementierung','Mittel');
INSERT INTO Arbeitspaket VALUES(3,6,'Modell','Schnitt');
INSERT INTO Arbeitspaket VALUES(3,7,'Implementierung','Hack');

INSERT INTO Arbeit VALUES(1,'Wichtig',50);
INSERT INTO Arbeit VALUES(1,'Klein',30);
INSERT INTO Arbeit VALUES(2,'Winzig',100);
INSERT INTO Arbeit VALUES(3,'Hack',70);
INSERT INTO Arbeit VALUES(4,'Maler',40);
INSERT INTO Arbeit VALUES(4,'Schreiber',30);
INSERT INTO Arbeit VALUES(6,'Maler',30);
INSERT INTO Arbeit VALUES(6,'Schreiber',40);
INSERT INTO Arbeit VALUES(7,'Hack',50);
```

zu 2.:

```
DELETE FROM Projekt
  WHERE PrName='Notendatenbank'
```

Die erste Zeile in Projekt, die ersten drei Zeilen in Arbeitspaket und die ersten vier Zeilen in Arbeit werden wegen `ON DELETE CASCADE` gelöscht.

zu 3.:

```
ALTER TABLE Arbeitspaket DISABLE CONSTRAINT A1;
UPDATE Projekt
 SET PrName='Anwesenheitsdatenbank', PrNr=11
 WHERE PRNAME='Fehlzeitendatenbank';
UPDATE Arbeitspaket
 SET PrNr=11
 WHERE PrNr=3;
ALTER TABLE Arbeitspaket ENABLE CONSTRAINT A1
```

Lösungen zu Kapitel 7

zu 1.:

```
SELECT Student.Name
  FROM Student, Pruefung
  WHERE Student.MatNr= Pruefung.MatNr
    AND Pruefung.Fach='Wahl1'
```

zu 2.:

```
SELECT Veranstaltung.Titel, Pruefung.Note
  FROM Student, Pruefung, Veranstaltung
  WHERE Student.MatNr=Pruefung.MatNr
    AND Pruefung.Fach=Veranstaltung.Kürzel
    AND Student.Name='Simson'
```

zu 3.:

```
SELECT Veranstaltung.Titel, Pruefung.Note
  FROM Pruefung, Veranstaltung
  WHERE Pruefung.Fach=Veranstaltung.Kürzel
```

zu 4.:

```
SELECT COUNT(*)
  FROM Pruefung
  WHERE Pruefung.Fach='DB'
```

zu 5.:

```
SELECT V1.Dozent
  FROM Veranstaltung V1, Veranstaltung V2
  WHERE V1.Dozent=V2.Dozent
    AND V1.Titel<>V2.Titel
```

zu 6.:

```
SELECT AVG(Pruefung.Note)
  FROM Pruefung, Veranstaltung
  WHERE Pruefung.Fach=Veranstaltung.Kürzel
    AND Veranstaltung.Dozent='Hinz'
```

zu 7.:

```
SELECT S1.Name
  FROM Student S1, Student S2, Pruefung P1, Pruefung P2
  WHERE S1.MatNr=P1.MatNr
    AND P1.Fach='DB'      AND S2.MatNr=P2.MatNr
    AND P2.Fach='DB'      AND S2.Name='Simson'
    AND S1.Name<>'Simson' AND P1.Note<=P2.Note
```

zu 8.:

```
SELECT DISTINCT Student.Name
  FROM Student, Pruefung, Veranstaltung
  WHERE Student.MatNr=Pruefung.MatNr
    AND Pruefung.Fach=Veranstaltung.Kürzel
    AND Veranstaltung.Dozent='Hinz'
```

Lösungen zu Kapitel 8

zu 1.:

```
SELECT Veranstaltung.Titel, AVG(Pruefung.Note)
  FROM Veranstaltung, Pruefung
  WHERE Veranstaltung.Kürzel=Pruefung.Fach
  GROUP BY Veranstaltung.Titel
```

zu 2.:

```
SELECT Veranstaltung.Dozent, COUNT(*)
  FROM Veranstaltung, Pruefung
  WHERE Veranstaltung.Kürzel=Pruefung.Fach
  GROUP BY Veranstaltung.Dozent
```

zu 3.:

```
SELECT Student.Name
  FROM Student, Pruefung
  WHERE Student.MatNr=Pruefung.MatNr
  GROUP BY Student.Name
  HAVING AVG(Pruefung.Note)<2.5
```

zu 4.:

```
SELECT Produkt.Pname
  FROM Produkt
```

zu 5.:

```
SELECT Produkt.Pname
  FROM Bonposition, Produkt
  WHERE Bonposition.ProdID=Produkt.ProdID
    AND Bonposition.BonID=1
```

zu 6.:

```
SELECT Bonposition.Anzahl, Produkt.Pname
  FROM Bon, Bonposition, Produkt
  WHERE Bonposition.BonID=Bon.BonID
    AND Bonposition.ProdID=Produkt.ProdID
    AND Bon.Verkaeufer='Müller'
```

zu 7.:

```
SELECT DISTINCT Produkt.Pname
  FROM Bonposition B1, Bonposition B2, Produkt
  WHERE B1.BonID<B2.BonID
    AND B1.ProdID=B2.ProdID
    AND B1.ProdID=Produkt.ProdID
```

zu 8.:

```
SELECT Produkt.Pname, SUM(Bonposition.Anzahl)
  FROM Produkt, Bonposition
  WHERE Produkt.ProdID = Bonposition.ProdID
  GROUP BY Produkt.Pname
```

zu 9.:

```
SELECT Bon.BonID, SUM(Bonposition.Anzahl*Produkt.Preis)
  FROM Bon, Bonposition, Produkt
  WHERE Bon.BonID=Bonposition.BonID
    AND Produkt.ProdID=Bonposition.ProdID
  GROUP BY Bon.BonID
```

Lösungen zu Kapitel 9

zu 1.:

```
SELECT Kino.Name
  FROM Kino, Vorfuehrung, Film
  WHERE Kino.Name=Vorfuehrung.Kino
    AND Vorfuehrung.Film=Film.FID
    AND FILM.Titel='Die Hand'
```

zu 2.:

```
SELECT Kino.Name, Kino.Plaetze*Kino.Saele
  FROM Kino
```

zu 3.:

```
SELECT Film.Titel
  FROM Film, Vorfuehrung
  WHERE Film.FID=Vorfuehrung.Film
  GROUP BY Film.Titel
  HAVING COUNT(*)>1
```

zu 4.:

```
 SELECT Kino.Name
   FROM Kino
MINUS
 SELECT Kino.Name
   FROM Kino, Vorfuehrung, Film
   WHERE Kino.Name=Vorfuehrung.Kino
     AND Vorfuehrung.Film=Film.FID
     AND Film.Titel='Die Nase'
```

zu 5.:

```
SELECT Kino.Name, MAX(Film.Laenge)
  FROM Kino, Vorfuehrung, Film
  WHERE Kino.Name=Vorfuehrung.Kino
    AND Vorfuehrung.Film=Film.FID
  GROUP BY Kino.Name
```

zu 6.:

```
SELECT Film.Titel,SUM(Kino.Plaetze)
  FROM Kino, Vorfuehrung, Film
  WHERE Kino.Name=Vorfuehrung.Kino
    AND Vorfuehrung.Film=Film.FID
  GROUP BY Film.Titel
```

zu 7.:

```
 SELECT F.Titel, K.Name, COUNT(*) Anzahl
  FROM (SELECT Film.Titel, Film.FID FROM Film) F,
       (SELECT Kino.Name FROM Kino) K,
       Vorfuehrung
  WHERE Vorfuehrung.Film=F.FID
    AND Vorfuehrung.Kino=K.Name
  GROUP BY F.Titel, K.Name
UNION
 SELECT F.Titel, K.Name, 0
  FROM (SELECT Film.Titel, Film.FID FROM Film) F,
       (SELECT Kino.Name FROM Kino) K
  WHERE NOT EXISTS(
    SELECT *
    FROM Vorfuehrung
    WHERE Vorfuehrung.Film=F.FID
      AND Vorfuehrung.Kino=K.Name)
```

Lösungen zu Kapitel 10

zu 1.:

zu a) Für Heinz und Verena können die Alterswerte (44,34), (43,35) und (44,35) erreicht werden.

zu b) Für Heinz und Verena können die Alterswerte (44,34) und (43,35) erreicht werden.

zu 2.:

zu a) Wird nur das Schattenspeicher-Verfahren eingesetzt, können trotzdem Lost Update und Unrepeatable Read-Probleme auftreten. Beim Unrepeatable Read muss dabei eine ganze Transaktion des Nutzers 1 zwischen den lesenden Operationen des Nutzers 2 stattfinden. Dirty Read-Probleme können nicht mehr auftreten.

zu b) Wichtig ist, dass der Schattenspeicher und das Datenbank-Managementsystem über die betroffenen Tabellen oder Zeilen Informationen austauschen. Zeilen, die in Schattenspeichern bearbeitet werden oder Grundlage von Berechnungen sind, müssen markiert werden. Dabei ist es wichtig, ob nur lesend oder auch schreibend auf die Zeile zugegriffen wird. Greift ein anderer Schattenspeicher auf eine markierte Zeile zu, ist ein weiterer lesender Zugriff unkritisch. Stoßen zwei schreibende Zugriffe oder ein schreibender und ein lesender Zugriff aufeinander, muss garantiert werden, dass eine Transaktion vollständig nach der anderen durchgeführt wird. Einen kompakten Überblick über mögliche Ansätze findet man in [Schu04].

Lösungen zu Kapitel 11

zu 1.:

```
CREATE VIEW WasWo AS
 SELECT Film.Titel, Vorfuehrung.Kino
  FROM Film, Vorfuehrung
  WHERE Film.FID=Vorfuehrung.Film
```

zu 2.:

```
SELECT WasWo.Titel, COUNT(*)
 FROM WasWo
 GROUP BY WasWo.Titel
```

zu 3.:

```
SELECT Kino.Name, Kino.Saele-COUNT(*)
 FROM Kino, WasWo
 WHERE Kino.Name=WasWo.Kino
 GROUP BY Kino.Name, Kino.Saele
```

zu 4.: Der View kann nicht für Änderungen genutzt werden, da er sich auf zwei Tabellen bezieht und weiterhin kein Primärschlüssel dieser Tabellen im View enthalten ist.

zu 5.: Wird ein Werkzeug zur Ableitung der Tabellen aus dem Entity-Relationship-Modell genutzt, kann dies Rechte zur Erzeugung und Änderung von Tabellen erhalten. Inhaltliche Änderungen der Tabellen sind nicht notwendig. Gibt es Personen, die ausschließlich für die Erstellung der Tabellen verantwortlich sind, können diese die gleichen Rechte haben.

Gibt es weitere Projekte, die ihr Ergebnis mit diesem Entwicklungsprojekt verknüpfen wollen, ist es sinnvoll, wenn die Entwickler des anderen Projekts Leserechte auf den Tabellen haben.

Wird zum Testen ein Werkzeug eingesetzt, das Zufallsdaten generiert, braucht dieses Werkzeug nur Schreibrechte auf den Tabellen.

Generell nutzen viele Werkzeuge im Bereich Software-Engineering und Konfigurationsmanagement Datenbanken, um aktuelle Projektergebnisse und frühere Projektstände zu verwalten. Zur Verwaltung müssen spezielle Tabellen eingerichtet werden, auf denen diese Werkzeuge alle Rechte haben.

Lösungen zu Kapitel 12

zu 1.:

```
CREATE OR REPLACE PROCEDURE einfuegen(nr Number, vn VARCHAR,
  n VARCHAR, gesch VARCHAR, land VARCHAR) IS
BEGIN
  INSERT INTO Kunde VALUES(nr,vn,n,gesch,land);
END;
```

zu 2.:

```
CREATE OR REPLACE PROCEDURE einfuegen2(vn VARCHAR,
              n VARCHAR, gesch VARCHAR, land VARCHAR) IS
  zeilenanzahl NUMBER(7);
  maxAnzahl NUMBER(5);
BEGIN – Alternativ SELECT MAX() nutzen, Ergebnis auf NULL testen
  SELECT COUNT(*)
    INTO zeilenanzahl
    FROM Kunde;
  IF zeilenanzahl=0
    THEN
      INSERT INTO Kunde VALUES(100,vn,n,gesch,land);
    ELSE
      SELECT MAX(KNR)
        INTO maxAnzahl
        FROM Kunde;
      INSERT INTO Kunde VALUES(maxAnzahl+1,vn,n,gesch,land);
  END IF;
END;
```

zu 3.:

```
CREATE OR REPLACE PROCEDURE auftragEintragen(kunde NUMBER,
     betr NUMBER, datum VARCHAR) IS
  dat2 DATE;
BEGIN
  dat2:=TO_DATE(datum)+7;
  INSERT INTO Auftrag VALUES(kunde,datum,betr,0,dat2);
END;
```

zu 4.:

```
CREATE OR REPLACE PROCEDURE auftragEintragen2(knde VARCHA2,
      betr NUMBER, datum VARCHAR) IS
  kundennummer Kunde.KNR%TYPE;
BEGIN
  einfuegen2(NULL,knde,NULL,NULL);
  SELECT MAX(KNR)
    INTO kundennummer
    FROM Kunde;
  AuftragEintragen(kundennummer,betr,datum);
END;
```

zu 5.:

```
CREATE OR REPLACE FUNCTION anrede (nr NUMBER)
  RETURN VARCHAR IS
  k Kunde%ROWTYPE;
BEGIN  -- fehlt Pruefung ob Kunde ueberhaupt existiert
  SELECT *
    INTO k
    FROM Kunde
    WHERE Kunde.KNR=nr;
  IF k.Geschlecht='W'THEN
    IF k.Land='CHN' THEN
      RETURN 'Sehr geehrte Frau '||k.Name||' '||k.Vorname||',';
    ELSE
      RETURN 'Sehr geehrte Frau '||k.Name||',';
    END IF;
  ELSIF k.Geschlecht='M' THEN
    IF k.Land='CHN' THEN
      RETURN 'Sehr geehrter Herr '||k.Name||' '||k.Vorname||',';
    ELSE
      RETURN 'Sehr geehrter Herr '||k.Name||',';
    END IF;
  ELSE
    RETURN 'Sehr geehrte/r Kundin/Kunde,';
  END IF;
END;
```

zu 6.:

```
CREATE OR REPLACE PROCEDURE alleNamen IS
CURSOR alle IS
SELECT *
```

```
FROM Kunde;
BEGIN
 FOR p IN alle
   LOOP
     DBMS_OUTPUT.PUT_LINE('Vorname: '|| p.Vorname ||
                          '  Nachname: '|| p.Name);
   END LOOP;
END;
```

zu 7.:

```
CREATE OR REPLACE PROCEDURE auftragEintragen3(kunde NUMBER, betr
NUMBER, datum VARCHAR2) IS
  offen Number;
BEGIN
  SELECT COUNT(*)
    INTO offen
    FROM Auftrag
  WHERE Auftrag.KNR=kunde;
  IF offen<3
    THEN
      INSERT INTO Auftrag
                  VALUES(kunde,datum,betr,0,TO_DATE(datum)+7);
    ELSE
      RAISE_APPLICATION_ERROR(-20202,'Zu viele offene Aufträge');
  END IF;
END;
```

zu 8.:

```
CREATE OR REPLACE TRIGGER guteKunden
  BEFORE INSERT
  ON Auftrag
  FOR EACH ROW -- sonst :NEW nicht zugreifbar auf Tabellenebene
DECLARE
  counter INTEGER;
BEGIN
  SELECT COUNT(*)
    INTO counter
    FROM Eintreiber
    WHERE Eintreiber.KNR = :NEW.KNR;
  IF counter>0
    THEN
      RAISE_APPLICATION_ERROR(-20101,'Kunde nicht kreditwürdig');
  END IF;
END;
```

zu 9.:

```
CREATE OR REPLACE TRIGGER kritischeKunden
  BEFORE INSERT
  ON Auftrag
  FOR EACH ROW -- sonst :NEW nicht zugreifbar auf Tabellenebene
DECLARE
```

```
  counter INTEGER;
BEGIN
  SELECT SUM(Auftrag.Mahnungsanzahl)
    INTO counter
    FROM Auftrag
    WHERE Auftrag.KNR = :NEW.KNR;
  IF counter>0
    THEN
      DBMS_OUTPUT.PUT_LINE('Kunde auf bestehende Außenstände '
                           || 'aufmerksam machen!');
  END IF;
END;
```

zu 10.:

```
CREATE OR REPLACE TRIGGER neuerKunde
BEFORE INSERT ON Kunde
FOR EACH ROW
BEGIN
 INSERT INTO Kundenstatistik
   VALUES(:NEW.KNR,0,0);
END;
```

zu 11.:

```
create or replace
TRIGGER skontoberechnen
BEFORE INSERT ON Auftrag
FOR EACH ROW
DECLARE
anzahl INTEGER;
gesamtwert NUMBER;
skontowert NUMBER;
BEGIN
 SELECT COUNT(*)
   INTO anzahl
   FROM Kunde
   WHERE KNR = :NEW.KNR;
 IF anzahl > 0
   THEN
     UPDATE Kundenstatistik
       SET Gesamt= Gesamt+ :NEW.Betrag
       WHERE KNR = :NEW.KNR;
     SELECT Kundenstatistik.Gesamt
       INTO gesamtwert
       FROM Kundenstatistik
       WHERE Kundenstatistik.KNR=:NEW.KNR;
     skontowert:=gesamtwert/1000;
     IF skontowert >10
      THEN skontowert:=10;
     END IF;
     UPDATE Kundenstatistik
       SET Skonto= skontowert
```

```
          WHERE KNR = :NEW.KNR;
        DBMS_OUTPUT.PUT_LINE('Skonto gewährt: '
              ||(:NEW.Betrag-:NEW.Betrag*(1-(skontowert/100))));
        :NEW.Betrag:= :NEW.Betrag*(1-(skontowert/100));
     END IF;
   END;
```

Lösungen zu Kapitel 13

zu 1.:

```
package aufgabe13;
public class Main {
  public static void main(String[] args) {
    Verbindung v=new Verbindung();
    if(v.anmelden())
      v.anfragen();
  }
}
```

```
package aufgabe13;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Scanner;

public class Verbindung {

  private Connection conn;
  private Statement statement;

  public boolean anmelden(){
    try {
      //Oracle JDBC driver laden
      DriverManager.registerDriver(
          new oracle.jdbc.driver.OracleDriver());

      Scanner eingabe=new Scanner(System.in);
      eingabe.useDelimiter("\n");
      System.out.print("Login: ");
      String login=textEinlesen();
      System.out.print("Passwort: ");
      String passwort=textEinlesen();
      //folgende Zeile kann Verbindung zu einer lokalen DB aufbauen
      conn = DriverManager
          .getConnection("jdbc:oracle:thin:@localhost:1521:xe",
              login, passwort);
```

```
      statement=conn.createStatement();
      return true;
    } catch (SQLException es) {
      problemAusgeben(es);
      return false;
    }
  }

  private String textEinlesen(){
    Scanner eingabe=new Scanner(System.in);
    eingabe.useDelimiter("\n");
    String ergebnis=eingabe.next();
    // letztes Zeichen /n löschen
    return ergebnis.substring(0,ergebnis.length()-1);
  }

  private void problemAusgeben(SQLException es){
    while (es != null) {
      System.out.print("Meldung: "+es.getMessage());
      System.out.println("ORACLE Fehlercode: " + es.getErrorCode());
      System.out.println("SQL State: " + es.getSQLState() + "\n");
      es = es.getNextException();
    }
  }

  public void anfragen() {
    String anfrage="bla";
    while(!anfrage.toLowerCase().equals("ende")){
      System.out.println("SELECT-Befehl eingeben [ohne Return]"
          +" (Ende mit \"ende\")");
      anfrage= textEinlesen();
      if (!anfrage.toLowerCase().equals("ende"))
        ausfuehren(anfrage);
    }
    try {
      conn.close();
    } catch (SQLException es) {
      problemAusgeben(es);
    }
  }

  private void ausfuehren(String anfrage) {
    try{
      ResultSet result= statement.executeQuery(anfrage);
      ResultSetMetaData metaresult=result.getMetaData();
      int spalten=metaresult.getColumnCount();
      for(int i=1;i<=spalten;i++)
        System.out.print(metaresult.getColumnName(i)+"\t| ");
      System.out.println();
      while(result.next()){
        for(int i=1;i<=spalten;i++)
```

```
          System.out.print(result.getString(i)+"\t| ");
        System.out.println();
      }
    }catch (SQLException es) {
      problemAusgeben(es);
    }
  }
}
```

zu 2.: Bei executeQuery werden alle Befehle ausgeführt; Fehlermeldungen kommen vom „falschen“ ResultSet-Objekt

```
Login: kleuker
Passwort: kleuker
SELECT-Befehl eingeben [ohne Return] (Ende mit "ende")
CREATE TABLE AB( X INTEGER)
Meldung: ORA-01003: keine Anweisung wurde analysiert/geparst
ORACLE Fehlercode: 1003
SQL State: 72000

SELECT-Befehl eingeben [ohne Return] (Ende mit "ende")
INSERT INTO AB VALUES(42)
Meldung: ORA-00900: Ungültige SQL-Anweisung
ORACLE Fehlercode: 900
SQL State: 42000

SELECT-Befehl eingeben [ohne Return] (Ende mit "ende")
SELECT * FROM AB
X          |
42         |
SELECT-Befehl eingeben [ohne Return] (Ende mit "ende")
UPDATE AB SET X=41
Meldung: ORA-00900: Ungültige SQL-Anweisung
ORACLE Fehlercode: 900
SQL State: 42000

SELECT-Befehl eingeben [ohne Return] (Ende mit "ende")
SELECT * FROM AB
X          |
41         |
SELECT-Befehl eingeben [ohne Return] (Ende mit "ende")
DELETE FROM AB
Meldung: ORA-00900: Ungültige SQL-Anweisung
ORACLE Fehlercode: 900
SQL State: 42000

SELECT-Befehl eingeben [ohne Return] (Ende mit "ende")
SELECT * FROM AB
X          |
SELECT-Befehl eingeben [ohne Return] (Ende mit "ende")
DROP TABLE AB
Meldung: ORA-01003: keine Anweisung wurde analysiert/geparst
```

```
ORACLE Fehlercode: 1003
SQL State: 72000

SELECT-Befehl eingeben [ohne Return] (Ende mit "ende")
SELECT * FROM AB
Meldung: ORA-00942: Tabelle oder View nicht vorhanden
ORACLE Fehlercode: 942
SQL State: 42000
```

Lösungen zu Kapitel 14

zu 1.: Es werden folgende Dateien mit Testdaten genutzt.

basisdaten.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<dataset>
  <Kunde Knr="100" Vorname="Haary" Name="Hase" Geschlecht="M"
         Land="D"/>
  <Kunde Knr="101" Vorname="Lisa" Name="Lustig" Geschlecht="W"
         Land="D"/>
  <Kunde Knr="102" Vorname="Wai" Name="Li" Geschlecht="W"
         Land="CHN"/>
  <Auftrag Knr="100" Datum="2012-12-11" Betrag="100.35"
           Mahnungsanzahl="0" Mahntermin="2012-12-18"/>
  <Auftrag Knr="101" Datum="2012-12-12" Betrag="80.80"
           Mahnungsanzahl="0" Mahntermin="2012-12-19"/>
  <Auftrag Knr="101" Datum="2012-11-11" Betrag="99.99"
           Mahnungsanzahl="1" Mahntermin="2012-12-01"/>
  <Eintreiber KNr = "101" Rechnungsdatum="2012-12-11"
              Uebergabetermin="2013-01-02"/>
  <Kundenstatistik />
</dataset>
```

testEinfuegen.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<dataset>
  <Kunde Knr="100" Vorname="Haary" Name="Hase" Geschlecht="M"
         Land="D"/>
  <Kunde Knr="101" Vorname="Lisa" Name="Lustig" Geschlecht="W"
         Land="D"/>
  <Kunde Knr="102" Vorname="Wai" Name="Li" Geschlecht="W"
         Land="CHN"/>
  <Kunde Knr="103" Vorname="Marie" Name="Meier" Geschlecht="W"
         Land="CH"/>
  <Auftrag Knr="100" Datum="2012-12-11" Betrag="100.25"
           Mahnungsanzahl="0" Mahntermin="2012-12-18"/>
  <Auftrag Knr="101" Datum="2012-12-12" Betrag="80.73"
           Mahnungsanzahl="0" Mahntermin="2012-12-19"/>
  <Auftrag Knr="101" Datum="2012-11-11" Betrag="99.81"
           Mahnungsanzahl="1" Mahntermin="2012-12-01"/>
  <Auftrag Knr="102" Datum="2013-01-02" Betrag="77.71"
```

```
          Mahnungsanzahl="0" Mahntermin="2013-01-09"/>
  <Eintreiber />
  <Kundenstatistik />
</dataset>
```

testEinfuegen2.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<dataset>
  <Kunde Knr="100" Vorname="Haary" Name="Hase" Geschlecht="M"
         Land="D"/>
  <Kunde Knr="101" Vorname="Lisa" Name="Lustig" Geschlecht="W"
         Land="D"/>
  <Kunde Knr="102" Vorname="Wai" Name="Li" Geschlecht="W"
         Land="CHN"/>
  <Kunde Knr="103" Name="Meier" />
  <Auftrag Knr="100" Datum="2012-12-11" Betrag="100.25"
           Mahnungsanzahl="0" Mahntermin="2012-12-18"/>
  <Auftrag Knr="101" Datum="2012-12-12" Betrag="80.73"
           Mahnungsanzahl="0" Mahntermin="2012-12-19"/>
  <Auftrag Knr="101" Datum="2012-11-11" Betrag="99.81"
           Mahnungsanzahl="1" Mahntermin="2012-12-01"/>
  <Auftrag Knr="103" Datum="2013-01-02" Betrag="77.71"
           Mahnungsanzahl="0" Mahntermin="2013-01-09"/>
  <Eintreiber />
  <Kundenstatistik />
</dataset>
```

```
package tests;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Types;

import org.dbunit.Assertion;
import org.dbunit.DatabaseUnitException;
import org.dbunit.database.DatabaseConfig;
import org.dbunit.database.DatabaseConnection;
import org.dbunit.database.IDatabaseConnection;
import org.dbunit.dataset.IDataSet;
import org.dbunit.dataset.ITable;
import org.dbunit.dataset.SortedTable;
import org.dbunit.dataset.xml.FlatXmlDataSetBuilder;
import org.dbunit.ext.oracle.OracleDataTypeFactory;
import org.dbunit.operation.DatabaseOperation;
```

```
import org.junit.After;
import org.junit.AfterClass;
import org.junit.Assert;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;

public class KundeAuftragEintreiberTest {

  private static Connection con = null;
  private static String dbAdresse = "127.0.0.1";
  private static String dbInstanz = "XE";
  private static IDatabaseConnection connDBU;

  private static void verbinden(String nutzer, String passwort)
      throws Exception {
    DriverManager.registerDriver(
         new oracle.jdbc.driver.OracleDriver());
    con = DriverManager.getConnection("jdbc:oracle:thin:@"
        + dbAdresse
        + ":1521:" + dbInstanz, nutzer, passwort);
  }

  public static void verbindungTrennen() throws Exception {
    if (con != null) {
      con.close();
    }
  }

  @BeforeClass
  public static void setUpBeforeClass() throws Exception {
    verbinden("ich", "x");
    connDBU = new DatabaseConnection(con, null, true);

    DatabaseConfig config = connDBU.getConfig();
    config.setProperty(DatabaseConfig.PROPERTY_DATATYPE_FACTORY
                    , new OracleDataTypeFactory());
  }

  @AfterClass
  public static void tearDownAfterClass() throws Exception {
    verbindungTrennen();
  }

  @Before
  public void setUp() {
    try {
      IDataSet dataSet = new FlatXmlDataSetBuilder()
          .build(new FileInputStream(
                      ".\\testdaten\\basisdaten.xml"));
      DatabaseOperation.CLEAN_INSERT.execute(connDBU, dataSet);
```

```
    } catch (FileNotFoundException
        | DatabaseUnitException | SQLException e) {
      e.printStackTrace();
    }
  }

  @After
  public void tearDown() throws Exception {
  }


  private void tabellenvergleich(String tabelle, String datei)
         throws Exception{
    IDataSet databaseDataSet = connDBU.createDataSet();
    ITable actualTable = databaseDataSet.getTable(tabelle);
    IDataSet expectedDataSet = new FlatXmlDataSetBuilder()
                                    .build(new File(datei));
    ITable expectedTable = expectedDataSet.getTable(tabelle);
    Assertion.assertEquals(new SortedTable(expectedTable)
                         , new SortedTable(actualTable));
  }

  @Test
  public void testErfolgreichAuftragEinfuegen() throws Exception {
    con.createStatement()
        .execute("INSERT INTO Auftrag "
            + "VALUES (102, '02.01.13', 77.77, 0,'09.01.13')");
    tabellenvergleich("Auftrag", ".\\testdaten\\testEinfuegen.xml");
  }

  @Test
  public void testKeinKundeFuerAuftrag() {
    try{
    con.createStatement()
        .execute("INSERT INTO Auftrag "
            + "VALUES (103, '02.01.13', 77.77, 0,'09.01.13')");
    Assert.fail();
    } catch (SQLException e){
      String erg = e.getMessage();
      Assert.assertTrue(erg.contains("constraint")
                    && erg.contains("violated"));
    } catch (Exception e){
      Assert.fail();
    }
  }

  @Test
  public void testDoppelterAuftrag() {
    try{
    con.createStatement()
        .execute("INSERT INTO Auftrag "
```

```
          + "VALUES (100, '11.12.12', 77.77, 0,'09.01.13')");
  Assert.fail();
  } catch (SQLException e){
    String erg = e.getMessage();
    Assert.assertTrue(erg.contains("constraint")
                  && erg.contains("violated"));
  } catch (Exception e){
    Assert.fail();
  }
}

@Test
public void testNullFuerBetragInAuftrag() {
  try{
  con.createStatement()
      .execute("INSERT INTO Auftrag "
          + "VALUES (100, '02.01.13', null, 0,'09.01.13')");
  Assert.fail();
  } catch (SQLException e){
    String erg = e.getMessage();
    Assert.assertTrue(erg.contains("insert")
                  && erg.contains("NULL"));
  } catch (Exception e){
    Assert.fail();
  }
}

@Test
public void testNullFuerMahnterminInAuftrag() {
  try{
  con.createStatement()
      .execute("INSERT INTO Auftrag "
          + "VALUES (100, '02.01.13', 77.77, 0,null)");
  Assert.fail();
  } catch (SQLException e){
    String erg = e.getMessage();
    Assert.assertTrue(erg.contains("insert")
                  && erg.contains("NULL"));
  } catch (Exception e){
    Assert.fail();
  }
}

@Test
public void testVierteMahnungInAuftrag() {
  try{
  con.createStatement()
      .execute("INSERT INTO Auftrag "
          + "VALUES (100, '02.01.13', 77.77, 4,'09.01.13')");
  Assert.fail();
  } catch (SQLException e){
```

```
      String erg = e.getMessage();
      Assert.assertTrue(erg.contains("constraint")
                    && erg.contains("violated"));
    } catch (Exception e){
      Assert.fail();
    }
  }

  @Test
  public void testProcedureEinfuegen2() throws Exception{
    CallableStatement stmt=con.prepareCall(
              "BEGIN einfuegen2('Marie','Meier','W','CH'); END;");
    stmt.execute();
    tabellenvergleich("Kunde", ".\\testdaten\\testEinfuegen.xml");
  }

  @Test
  public void testProcedureAuftragEintragen2() throws Exception{
    CallableStatement stmt=con.prepareCall(
       "BEGIN auftragEintragen2('Meier',77.77,'02.01.2013'); END;");
    stmt.execute();
    tabellenvergleich("Kunde", ".\\testdaten\\testEinfuegen2.xml");
    tabellenvergleich("Auftrag"
                   , ".\\testdaten\\testEinfuegen2.xml");
  }

  private String anrede(int knr) throws SQLException{
    CallableStatement stmt=con.prepareCall(
              "{? = call anrede(?)}");
          stmt.registerOutParameter(1,Types.VARCHAR);
          stmt.setInt(2, knr);
          stmt.execute();
         return stmt.getString(1);
  }

  @Test
  public void testAnrede1() throws SQLException{
    Assert.assertEquals(anrede(100), "Sehr geehrter Herr Hase,");
  }

  @Test
  public void testAnrede2() throws SQLException{
    Assert.assertEquals(anrede(101), "Sehr geehrte Frau Lustig,");
  }

  @Test
  public void testAnrede3() throws SQLException{
    Assert.assertEquals(anrede(102), "Sehr geehrte Frau Li Wai,");
  }

  @Test
```

```
  public void testAnrede4() throws SQLException{
    CallableStatement stmt=con.prepareCall(
       "BEGIN auftragEintragen2('Meier',77.77,'02.01.2013'); END;");
    stmt.execute();
    Assert.assertEquals(anrede(103)
                      , "Sehr geehrte/r Kundin/Kunde,");
  }


  @Test
  public void testTriggerEintreiber() {
    try{
    con.createStatement()
        .execute("INSERT INTO Auftrag "
            + "VALUES (101, '02.01.13', 77.77, 0,'09.01.13')");
    Assert.fail();
    } catch (SQLException e){
      String erg = e.getMessage();
      Assert.assertTrue(erg.contains("Kunde nicht kreditwürdig"));
    } catch (Exception e){
      Assert.fail();
    }
  }

  private double skonto(int knr) throws SQLException{
    ResultSet rs = con.createStatement().executeQuery(
        "SELECT * FROM KundenStatistik "
       +"  WHERE KNR="+knr);
    rs.next();
    return rs.getDouble("Skonto");
  }

  @Test
  public void testSkotoBerechnen() throws Exception{
    CallableStatement stmt=con.prepareCall(
              "BEGIN einfuegen2('Marie','Meier','W','CH'); END;");
    stmt.execute();
    Assert.assertTrue(skonto(103)==0);
    con.createStatement().execute("INSERT INTO Auftrag "
              + "VALUES (103, '02.01.13', 2000, 0,'09.01.13')");
    Assert.assertTrue(skonto(103)==2);
    con.createStatement().execute("INSERT INTO Auftrag "
              + "VALUES (103, '03.01.13', 3000, 0,'10.01.13')");
    Assert.assertTrue(skonto(103)==5);
    con.createStatement().execute("INSERT INTO Auftrag "
              + "VALUES (103, '04.01.13', 12000, 0,'11.01.13')");
    Assert.assertTrue(skonto(103)==10);
  }
}
```

Lösungen zu Kapitel 15

```
package db;

import com.mongodb.MongoClient;
import com.mongodb.MongoClientURI;
import com.mongodb.client.AggregateIterable;
import com.mongodb.client.FindIterable;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoCursor;
import com.mongodb.client.MongoDatabase;
import com.mongodb.client.MongoIterable;
import com.mongodb.client.model.Filters;
import entities.Mitarbeit;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashSet;
import java.util.List;
import java.util.Set;
import javafx.util.Pair;
import org.bson.Document;

public class Verbindung {

  private MongoClient mongoClient;
  private MongoDatabase db;
  private MongoCollection<Document> coll;
  private MongoCollection<Document> counter;
  private final static String DATENBANK = "db01";
  private final static String ZAEHLER = "zaehler";
  private final static String SAMMLUNG = "mitarbeit";

  public Verbindung() {
    MongoClientURI connectionString
        = new MongoClientURI("mongodb://localhost:27017");
    this.mongoClient = new MongoClient(connectionString);
    this.db = mongoClient.getDatabase(DATENBANK);
    this.coll = db.getCollection(SAMMLUNG);
    this.counter = db.getCollection(ZAEHLER);
    if (this.counter.count() == 0) {
      this.counter.insertOne(new Document("wert", 10));
      this.beispieldaten();
    }
  }

  public void schliessen() {
    this.mongoClient.close();
  }

  public void allesLoeschen() {
    this.coll.drop();
```

```
  this.counter.drop();
}

private void beispieldaten() {
  Mitarbeit[] ma = {
      new Mitarbeit(this.gibID(), "DB", "Oleg", "Prog", 60)
    , new Mitarbeit(this.gibID(), "DB", "Oleg", "Design", 20)
    , new Mitarbeit(this.gibID(), "DB", "Ute", "Prog", 30)
    , new Mitarbeit(this.gibID(), "DB", "Amy", "Analyse", 20)
    , new Mitarbeit(this.gibID(), "GUI", "Ute", "Prog", 30)
    , new Mitarbeit(this.gibID(), "GUI", "Amy", "Analyse", 80)
    , new Mitarbeit(this.gibID(), "GUI", "Ben", "Use", 100)
    , new Mitarbeit(this.gibID(), "BU", "Oleg", "Analyse", 40)
    , new Mitarbeit(this.gibID(), "BU", "Ute", "Prog", 30)
    , new Mitarbeit(this.gibID(), "BU", "Yuri", "Prog", 100)
  };
  for (Mitarbeit m : ma) {
    this.hinzu(m);
  }
}

public void hinzu(Mitarbeit p) {
  this.coll.insertOne(p.asDocument());
}

public int gibID() {
  Document doc = this.counter.find().first();
  int wert = (Integer) doc.get("wert"); // oder getInteger("wert")
  doc.put("wert", wert + 1);
  this.counter.replaceOne(Filters.eq("_id", doc.get("_id")), doc);
  return wert;
}

public void ausgeben(MongoIterable<Document> it) {
  MongoCursor<Document> cur = it.iterator();
  while (cur.hasNext()) {
    System.out.println(cur.next().toJson());
  }
  cur.close();
  System.out.println("----------");
}

public FindIterable<Document> alle() {
  return this.coll.find();
}

public FindIterable<Document> projektmitarbeiten(String projekt) {
  return this.coll.find(Filters.eq("projekt", projekt));
}
```

```
  public FindIterable<Document> mitarbeitermitarbeiten(String ma) {
    return this.coll.find(Filters.eq("mitarbeiter", ma));
  }

  public FindIterable<Document> mehrAls50Programmierung() {
    return this.coll.find(Filters.and(Filters.eq("aufgabe", "Prog")
                                    , Filters.gt("anteil", 50)));
  }

  public AggregateIterable<Document> projektuebersicht() {
    return this.coll.aggregate(Arrays.asList(
           new Document("$group",
                 new Document("_id", "$projekt")
                    .append("anzahl", new Document("$sum", 1))
              .append("anteile"
                    , new Document("$sum", "$anteil")))));
  }

  public AggregateIterable<Document> ueberlasteteMitarbeiter() {
    return this.coll.aggregate(Arrays.asList(
         new Document("$group"
             , new Document("_id", "$mitarbeiter")
           .append("anteile", new Document("$sum", "$anteil")))
       , new Document("$match"
             , new Document("anteile", new Document("$gt", 100.0)))
    ));
  }

  public Set<String> mitarbeiterInMehrerenProjekten() {
    Set<String> erg = new HashSet<>();
    String javascriptMap = "function(){"
        + "  emit(this.mitarbeiter, '');" // Dummy: leerer String
        + "  emit(this.mitarbeiter, this.projekt);"
        + "} ";
    String javascriptReduce = "function(key, value){"
        + "  var erg = 0; "
        + "  print(key + ': ' + value);"
        + "  for (var i=0; i<value.length; i++){ "
        + "    for (var j=0; j<value.length; j++){ "
        + "      if (i!=j && value[i]==value[j] && value[i]!='') {"
        + "        erg = 1;"
        + "      }"
        + "    }"
        + "  } "
        + "  print(erg);"
        + "  return erg; "
        + "} ";
    this.ausgeben(this.coll.mapReduce(javascriptMap
                , javascriptReduce));
    for (Document d : this.coll.mapReduce(javascriptMap
                , javascriptReduce)) {
```

```
    System.out.println("  " + d.toJson());
    if (d.getDouble("value") == 1.0) {
      erg.add(d.getString("_id"));
    }
  }
  return erg;
}

public Set<String> mitarbeiterInMehrerenAufgaben() {
  Set<String> erg = new HashSet<>();
  String javascriptMap = "function(){"
      + "  emit(this.mitarbeiter, '');" // Dummy: leerer String
      + "  emit(this.mitarbeiter, this.aufgabe);"
      + "} ";
  String javascriptReduce = "function(key, value){"
      + "  print(key+ ': ' + value);"
      + "  for (var i=0; i<value.length; i++){ "
      + "    for (var j=0; j<value.length; j++){ "
      + "      if (i!=j && value[i]!=value[j] && value[i]!='' "
      + "               && value[j]!='') {"
      + "         return 1;"
      + "      }"
      + "    }"
      + "  } "
      + "  return 0; "
      + "} ";
  for (Document d : this.coll.mapReduce(javascriptMap
                                        , javascriptReduce)) {
    if (d.getDouble("value") == 1.0) {
      erg.add(d.getString("_id"));
    }
  }
  return erg;
}

public Set<String> nichtProgrammierendeMitarbeiter() {
  Set<String> erg = new HashSet<>();
  String javascriptMap = "function(){"
      + "  emit(this.mitarbeiter, 'X');" // Dummy-Aufgabe X
      + "  emit(this.mitarbeiter, this.aufgabe);"
      + "} ";
  String javascriptReduce = "function(key, value){"
      + "  print(key+ ': ' + value);"
      + "  for (var i=0; i<value.length; i++){ "
      + "    if (value[i]=='Prog') {"
      + "      return 1;"
      + "    }"
      + "  } "
      + "  return 0; "
      + "} ";
```

```
    for (Document d : this.coll.mapReduce(javascriptMap
                                        , javascriptReduce)) {
      if (d.getDouble("value") == 0.0) {
        erg.add(d.getString("_id"));
      }
    }
    return erg;
  }

  public Set<Pair<String, String>> nieZusammen() {
    Set<Pair<String, String>> erg = new HashSet<>();
    Set<String> mitarbeiter = new HashSet<>();
    for (Document d : this.coll.find()) {
      mitarbeiter.add(d.getString("mitarbeiter"));
    }
    List<String> mtmp = new ArrayList<>(mitarbeiter);
    for(int i=0; i < mtmp.size(); i++){
      for(int j=i+1; j < mtmp.size(); j++){
        erg.add(new Pair(mtmp.get(i), mtmp.get(j)));
      }
    }

    Set<String> projekte = new HashSet<>();
    for (Document d : this.coll.find()) {
      projekte.add(d.getString("projekt"));
    }

    Set<Pair<String, String>> loeschen = new HashSet<>();
    for (String pro : projekte) {
      for (Pair<String, String> p : erg) {
        if (this.coll.find(Filters
            .and(Filters.eq("mitarbeiter", p.getKey())
               , Filters.eq("projekt", pro))).first() != null
           && this.coll.find(Filters
            .and(Filters.eq("mitarbeiter", p.getValue())
               , Filters.eq("projekt", pro))).first() != null) {
          loeschen.add(p);
        }
      }
    }
    erg.removeAll(loeschen);
    return erg;
  }
}
```