

### 3 Abfragen mit SQL

Um bestimmte Datensätze aus der Datenbank zu gewinnen, verwendest du die »Programmiersprache« SQL (Structured Query Language). Sie ist derzeit die am weitesten verbreitete Sprache für relationale Datenbanken. Auch in Zukunft wird diese Sprache keine Konkurrenz bekommen, was sich aus der Tatsache ableiten lässt, dass mittlerweile alle Hersteller relationaler Datenbanken ihr System auf diese Sprache umgestellt haben.

#### Kleine SQL-Geschichte

Anfang der siebziger Jahre wurde in den IBM Forschungslaboratorien in San Jose, Kalifornien, ein Forschungsprojekt begonnen, das sich »System R« nannte. Es sollte die Praktikierbarkeit der relationalen Theorien untersuchen. Von den IBM-Mitarbeitern R.F. Boyce und D.D. Chamberlain wurde die Sprache SEQUEL (sprich: siequel) entwickelt, die später in SQL umbenannt wurde. Man lehnte hierbei die Syntax an Begriffe der englischen Umgangssprache wie z.B. SELECT, FROM, WHERE an. Seit dieser Zeit wurden von fast allen DB-Herstellern SQL-Schnittstellen zu ihren relationalen und nichtrelationalen Datenbanksystemen entwickelt.

#### 3.1 Beispieldatenbank EDV\_Kurse

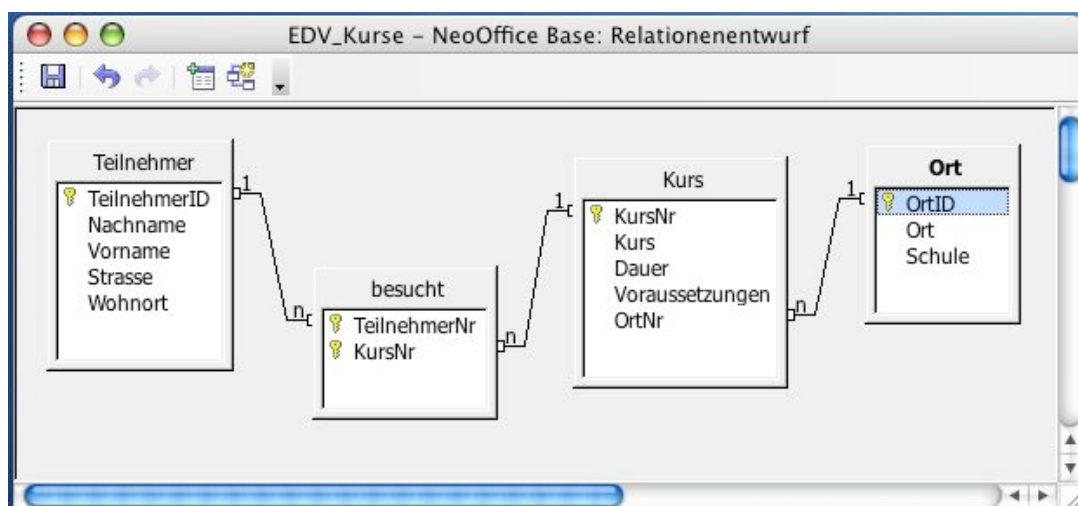


Abbildung 3.1: Die Datenbank *EDV\_Kurse*

Anhand dieser Datenbank wirst du die wichtigsten SQL-Befehle kennen lernen.

#### Bemerkung:

a) Diese Datenbank genügt nicht den Normalformen. Um die folgenden

SQL-Anweisungen besser zu verdeutlichen, wurden die Attributnamen zum Teil zweifach gewählt

- b) Da ich WORD auf „Typographische Anführungszeichen“ eingestellt habe, wurden in allen folgenden Lösungsbeispielen die ersten Anführungszeichen nach unten gesetzt. In SQL-Abfragen gehören diese Anführungszeichen nach oben. ACCESS übernimmt dies jedoch automatisch.

### 3.2 Projektion

Die Projektion ist eine einfache Operation, die nichts anderes macht, als bestimmte Spalten einer Tabelle auszuwählen. Von allen Objekten, die in einer Tabelle gespeichert sind, werden bestimmte Attribute ausgewählt. Diese Operation ist oft erforderlich, da in der Regel nicht immer alle Eigenschaften eines Objekts ausgewertet bzw. aufgelistet werden müssen.

TeilnehmerID	Nachname	Vorname	Strasse	Wohnort
0	Sorglos	Susi	Hauptstr. 142	70410 Sonnstetten
1	Hofmann	Helm	Am Bächle 3	66822 Heidelberg
2	Hauer	Hans	Im Winkel 16a	72329 Thalhausen
3	Pfeiffer	Claudia	Mozartweg 6	74121 Ludwigshafen
4	Wallung	Walter	Panoramapfad 33	69663 Grünstadt
5	Peters	Paul	Am Markt 1	53522 Köln
6	Unterländer	Elke	Max-Weber-Str. 12	81023 München
7	Nicks	Steffi	Holzweg 8	73124 Oberndorf
8	Hirsch	Harry	Baumgartenstr. 2	75175 Pforzheim

Abbildung 3.2: Projektion bedeutet Auswahl von Spalten einer Tabelle

#### 3.2.1 Auswahl aller Spalten einer Tabelle

Syntax: **SELECT \***  
**FROM <Tabelle>**

Beispiel: Gesucht werden die Informationen aller Teilnehmer.

Beispiel: Gesucht wird die gesamte Tabelle *Kurs*.

```
SELECT *
FROM Kurs
```

### 3.2.2 Auswahl einer Spalte einer Tabelle

Syntax:        **SELECT <Spalte>**  
                  **FROM <Tabelle>**

Beispiel:        Gesucht werden die Nachnamen aller Teilnehmer der Kurse.

```
SELECT Nachname
FROM Teilnehmer
```

### 3.2.3 Auswahl mehrerer Spalten einer Tabelle

Syntax:        **SELECT <Spalte1> , <Spalte2> , ...**  
                  **FROM <Tabelle>**

Beispiel:        Gesucht werden die Vor- und Nachnamen aller Teilnehmer der Kurse.

```
SELECT Vorname, Nachname
FROM Teilnehmer
```

Hinweis:        In SQL kann ein Abfrageergebnis mehrere identische Datensätze enthalten.

### 3.2.4 Auswahl ohne mehrfaches Auftreten desselben Tupels

Syntax:        **SELECT DISTINCT <Spalte>**  
                  **FROM <Tabelle>**

Beispiel:        Gesucht werden die verschiedenen Ortsnamen aller Schulen.  
Befinden sich zwei Schulen im gleichen Ort, so wird dieser Ort nur einmal aufgeführt.

```
SELECT DISTINCT Ort
FROM Ort
```

### 3.2.5 Formatierte Ausgabe und Berechnung in einer Projektion

Beispiel:        Es soll für jeden Kurs berechnet werden, wie viele Stunden dieser dauert.

```
SELECT „Die Fortbildung“, Kurs, „dauert“, Dauer*8,
„Stunden“ FROM Kurs
```

### 3.2.6 Umbenennen von Spalten

Syntax:        **SELECT <Spalte> AS <neuer Spaltenname>**

**FROM <Tabelle>**

Beispiel: Unter der Bezeichnung „Schulorte“ sind alle Schulen gesucht, die in der Tabelle *Ort* gespeichert sind.

```
SELECT DISTINCT Ort AS Schulorte
FROM Ort
```

**3.2.7 Sortierung**

Syntax: **SELECT <Spalte>**  
**FROM <Tabelle>**  
**ORDER BY <Spalte> {DESC/ASC}**

Beispiel: Gesucht werden für jeden Teilnehmer der Name und sein Wohnort. Dabei ist die Ergebnistabelle absteigend nach Nachnamen sortiert auszugeben.

```
SELECT Nachname, Wohnort
FROM Teilnehmer
ORDER BY Nachname DESC
```

**Übung 3.1:**

Erstelle in Kapitel 3.8.1 *Abfragen zum Thema Projektion und Selektion* zur Datenbank *Kaufhaus* die **SQL-Abfragen zur Projektion**.

**3.3 Selektion**

Die Selektion stellt eine der wichtigsten Datenbankoperationen dar. Sie wird eingesetzt, um aus einer Tabelle die Menge der Datensätze herauszufiltern, die einer bestimmten Bedingung genügen.

TeilnehmerID	Nachname	Vorname	Strasse	Wohnort
0	Sorglos	Susi	Hauptstr. 142	40410 Sonnstetten
1	Hofmann	Helm	Am Bächle 3	66822 Heidelberg
2	Hauer	Hans	Im Winkl 16a	12329 Thalhausen
3	Pfeiffer	Claudia	Mozartweg 6	74121 Ludwigshafen
4	Wallung	Walter	Panoramapfad 33	09663 Grünstadt
5	Peters	Paul	Am Markt 1	53522 Köln
6	Unterlander	Elke	Max-Weber-Str. 12	81023 Munchen
7	Nicks	Steffi	Holzweg 8	73124 Oberndorf
8	Hirsch	Harry	Baumgartenstr. 2	75175 Pforzheim
<AutoFeld>				

Abbildung 3.3: Selektion bedeutet Auswahl von *Datensätzen* einer Tabelle

Syntax:           **SELECT** <Spalte>  
                       **FROM** <Tabelle>  
                       **WHERE** <Bedingung>

### 3.3.1 Selektion mit einfachem Vergleich

Beispiel:           Gesucht werden die Kurse, bei denen keine Voraussetzungen notwendig sind.

```
SELECT Kurs
FROM Kurs
WHERE Voraussetzungen = ,keine'
```

Beispiel:           Gesucht werden die Kurse, bei denen Voraussetzungen gefordert werden.

```
SELECT Kurs
FROM Kurs
WHERE NOT (Voraussetzungen = ,keine')
```

Beispiel:           Gesucht werden die Kurse, die mindestens 3 Tage dauern.

```
SELECT Kurs, Dauer
FROM Kurs
WHERE Dauer >= 3
```

### 3.3.2 Selektion mit mehreren Bedingungen

Beispiel: Gesucht werden alle Kurse, die Programmiererfahrung erfordern und nicht länger als 10 Tage dauern.

```
SELECT Kurs
FROM Kurs
WHERE (Voraussetzungen = ,Programmiererfahrung') AND
(Dauer <= 10)
```

Beispiel: Gesucht werden die Nachnamen und Wohnorte aller Teilnehmer, deren Vorname Harry oder Susi ist.

```
SELECT Nachname, Wohnort
FROM Teilnehmer
WHERE (Vorname = ,Harry') OR (Vorname = ,Susi')
```

### 3.3.3 Selektion mit dem Operator IN

Beispiel: Gesucht werden die Namen aller Schulen, die in Pforzheim oder Freiburg sind.

```
SELECT Schule
FROM Ort
WHERE Ort IN (,Pforzheim', ,Freiburg')
```

Beispiel: Gesucht werden die Namen aller Schulen, die nicht in Sindelfingen, Ulm und Freiburg sind.

```
SELECT Schule
FROM Ort
WHERE Ort NOT IN (,Sindelfingen', ,Ulm', ,Freiburg')
```

### 3.3.4 Selektion mit dem Operator LIKE

Der LIKE-Operator ermöglicht den Vergleich eines Strings mit einem Muster. Solche Muster werden aus beliebigen Zeichen eines Strings und den beiden Sonderzeichen „?“ und „\*“ gebildet. Hierbei steht „?“ für genau ein beliebiges Zeichen, während „\*“ für eine beliebig große (evtl. leere) Kette von beliebigen Zeichen steht.

Beispiel: Gesucht werden die Vornamen aller Teilnehmer, deren Postleitzahl mit „3“ endet.

```
SELECT Vorname
FROM Teilnehmer
WHERE Wohnort LIKE ,????3*
```

Beispiel: Gesucht werden die Vornamen aller Teilnehmer, deren Nachname mit „H“ und deren Wohnort nicht mit „H“ beginnt.

```
SELECT Vorname
FROM Teilnehmer
WHERE (Nachname LIKE ,H*') AND (Wohnort NOT LIKE ,?????H*')
```

**Bemerkung:**

Die sechs Fragezeichen entsprechen der fünfstelligen PLZ und dem anschließendem Leerzeichen.

### 3.3.5 Selektion und NULL-Werte

NULL wird interpretiert als ein Platzhalter für die Aussage „Information/Attribut ist nicht vorhanden oder nicht bekannt oder nicht anwendbar“.

Beispiel: Gesucht werden die Kurse, die keine Dauer angegeben haben (oder deren Dauer nicht bekannt ist).

```
SELECT Kurs
FROM Kurs
WHERE Dauer IS NULL
```

Übung 3.2:

Erstelle in Kapitel 3.8.1 *Abfragen zum Thema Projektion und Selektion* zur Datenbank *Kaufhaus* die **SQL-Abfragen zur Selektion**.

Übung 3.3:

Erstelle in Kapitel 3.8.1 *Abfragen zum Thema Projektion und Selektion* zur Datenbank *Kaufhaus* die **Umgangssprachliche Formulierung**.

Übung 3.4:

Erstelle in Kapitel 3.8.1 *Abfragen zum Thema Projektion und Selektion* zur Datenbank *Kaufhaus* die **Wiederholungsaufgaben**.

### 3.4 Kreuzprodukt

Das (kartesische) Kreuzprodukt ist eine etwas künstliche Operation, die nur in Zusammenhang mit der Operation JOIN eine praktische Bedeutung hat. Das Kreuzprodukt zweier Tabellen ergibt eine neue Tabelle, die als Zeilen alle möglichen Kombinationen der Datensätze beider Relationen enthält.

```
Syntax:      SELECT *  
            FROM <Tabelle1> , <Tabelle2>
```

Beispiel: Gesucht ist die Schule, in welcher der Kurs „Grundlagen  
Datenbanken“ stattfindet.

```
SELECT * FROM Kurs, Ort
```

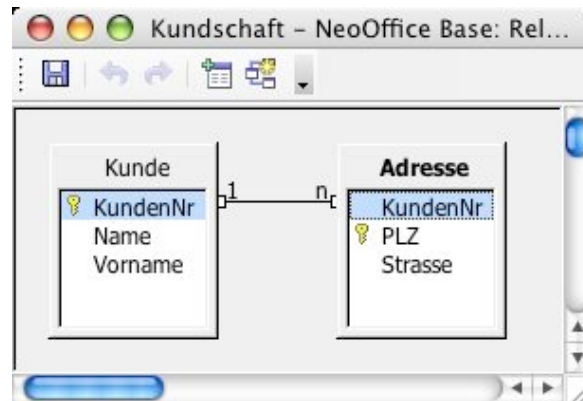
Die Abbildung 3.1 zeigt, dass zur Beantwortung dieser Frage die Informationen aus zwei Tabellen miteinander verknüpft werden müssen, nämlich aus Tabelle *Kurs* und *Ort*. Deswegen versuche folgende Abfrage:

```
SELECT * FROM Kurs, Ort
```

Du hast nun ein Kreuzprodukt der beiden Tabellen *Kurs* und *Ort* erstellt. Das Kreuzprodukt dieser beiden Tabellen liefert eine neue Tabelle, in der jeder Datensatz aus *Kurs* mit jedem Datensatz aus *Ort* kombiniert wird. Du erkennst dies daran, dass diese neue Tabelle nun  $72 = 12 * 6$  Datensätze enthält.

Somit enthält das Kreuzprodukt viele sinnlose Datensätze. Erst durch eine anschließende Selektion und Projektion bekommst du sinnvolle Aussagen.

Bevor du am obigen Beispiel weiter arbeitest, wird das Kreuzprodukt an Hand einer kleinen Datenbank *Kundschaft* bestehend aus den beiden kleinen Relationen *Kunde* und *Adresse* noch einmal verdeutlicht:

Abbildung 3.4: Die Datenbank *Kundschaft*

Kunde			Adresse		
KundenNr	Name	Vorname	KundenNr	PLZ	Strasse
1	Meier	Hans	1	81673	Hauptstr.
2	Gasser	Evi	2	80000	Eichenweg
3	Schmidt	Markus			

Abbildung 3.5: Die Tabellen *Kunde* und *Adresse***Bemerkung:**

Beachte, dass in diesem Beispiel in beiden Relationen der gleiche Attributname *KundenNr* erscheint. Vergleiche hierzu auch den Hinweis in Kapitel 3.5.

Beispiel: Gesucht ist die Straße, in der die Kundin Gasser wohnt.

```
SELECT * FROM Kunde, Adresse
```

Du bildest also zuerst das Kreuzprodukt dieser beiden Tabellen. Damit wird eine neue Tabelle mit vielen sinnlosen Datensätzen erzeugt. Es wird jeder Datensatz aus *Kunde* mit jedem Datensatz aus *Adresse* kombiniert:

Kunde x Adresse					
KundenNr	Name	Vorname	KundenNr	PLZ	Strasse
1	Meier	Hans	1	81673	Hauptstr.
1	Meier	Hans	2	80000	Eichenweg
2	Gasser	Evi	1	81673	Hauptstr.
2	Gasser	Evi	2	80000	Eichenweg
3	Schmidt	Markus	1	81673	Hauptstr.
3	Schmidt	Markus	2	80000	Eichenweg

Abbildung 3.6: Kreuzprodukt aus den Tabellen *Kunde* und *Adresse*

In dieser Tabelle bildet der 4. Datensatz die Lösung. Um an diesen Datensatz zu kommen, verwendest du eine Selektion mit der Bedingung, dass die *KundenNr* aus *Kunde* und die *KundenNr* aus *Adressen* den gleichen Wert haben.

Ein solches Kreuzprodukt mit anschließender Selektion und Projektion bezeichnet man als JOIN.

### 3.5 Verbund (Join)

Das obige Beispiel aus der Datenbank *Kundschaft* wird nun weiter besprochen.

Beispiel: Gesucht ist die Straße, in der die Kundin Gasser wohnt.

```
SELECT *
FROM Kunde, Adresse
WHERE Kunde.KundenNr = Adresse.KundenNr
```

Diese Abfrage liefert nun folgende Tabelle:

Kunde [x]		Adresse			
KundenNr	Name	Vorname	KundenNr	PLZ	Strasse
1	Meier	Hans	1	81673	Hauptstr
2	Gasser	Evi	2	80000	Eichenweg

**Abbildung 3.7:** Kunden mit der korrekzten Adresse

Somit hast du nun aus dem Kreuzprodukt alle Kunden herausgefiltert, deren Primärschlüssel aus *Kunde* mit dem Fremdschlüssel aus *Adresse* übereinstimmen. Diese Tabelle enthält somit nur noch Kunden, deren Straßen bekannt sind.

Die Kundin Gasser erhält man mit Hilfe einer zweiten Selektionsbedingung.

Beispiel: Gesucht ist die Straße, in der die Kundin Gasser wohnt.

```
SELECT *
FROM Kunde, Adresse
WHERE (Kunde.KundenNr = Adresse.KundenNr) AND
(Name = ‚Gasser‘)
```

Kunde [x]		Adresse			
KundenNr	Name	Vorname	KundenNr	PLZ	Strasse
2	Gasser	Evi	2	80000	Eichenweg

**Abbildung 3.8:** Datensatz enthält die gesuchte Kundin

Laut Aufgabenstellung interessiert uns nur das Attribut *Name* aus *Kunde* und *Strasse* aus *Adresse*. Also führst du zuletzt noch eine Projektion durch, die diese Attribute herausfiltert.

Beispiel: Gesucht ist die Straße, in der die Kundin Gasser wohnt.

```
SELECT Name, Strasse
FROM Kunde, Adresse
WHERE (Kunde.KundenNr = Adresse.KundenNr) AND
(Name = „Gasser“)
```

Kunde [x]	Adresse
Name	Strasse
Gasser	Eichenweg

Abbildung 3.9: Lösung der Abfrage

Syntax: **SELECT** <Spalte1> , <Spalte2> , ...  
**FROM** <Tabelle1> , <Tabelle2> , ...  
**WHERE** <JOIN-Bedingung>

Hinweis: Wenn die Tabellen, die miteinander zu verbinden sind, Spalten mit gleichem Spaltennamen aufweisen, dann muss jeweils spezifiziert werden, welche Spalte welcher Tabelle gemeint ist.

Tabellenverknüpfungen sind ein so wesentlicher Bestandteil der Datenbankabfrage, dass mit der JOIN-Klausel eine weitere Möglichkeit eingeführt wurde.

Beispiel: Gesucht ist die Straße, in der die Kundin Gasser wohnt.

```
SELECT Name, Strasse
FROM Kunde INNER JOIN Adresse
ON Kunde.KundenNr = Adresse.KundenNr
WHERE Name = ‚Gasser‘
```

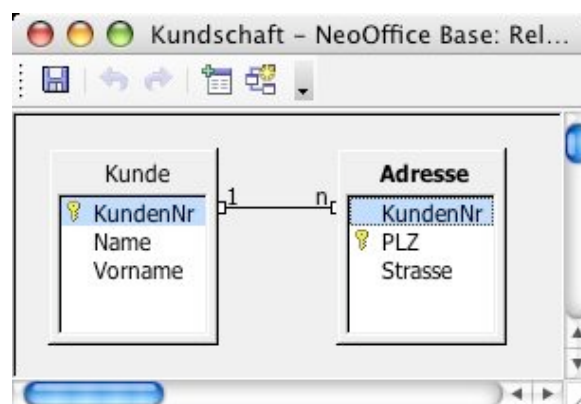


Abbildung 3.10: Die Datenbank *Kundschaft*

Diese Art der Join-Abfragen solltest du dir genauer anschauen. In der Datenbank *Kundschaft* erfolgt die obige Abfrage über zwei Tabellen. Hierbei wird die Tabelle

*Kunde* die übergeordnete oder Elterntabelle und  
*Adresse* die untergeordnete oder Kindtabelle

Genannt. Bei Abfragen zu einem solchen Sachverhalt verwendet man einen der folgenden Join-Typen:

- INNER JOIN** Liefert nur Datensätze, bei denen die beiden verknüpften Attribute identische Werte.
- LEFT JOIN** Liefert zusätzlich auch die Datensätze der Elterntabelle, die mit keinem Datensatz der Kindtabelle verknüpft sind. Im Beispiel also: alle Kunden mit Namen, auch die Kunden, für die keine Adresse eingetragen ist.  
„Left“ deswegen, weil die Elterntabelle in der Join-Syntax immer als erste, also links, genannt wird.
- RIGHT JOIN** Liefert zusätzlich auch die Datensätze der Kindtabelle, die mit keinem Datensatz der Elterntabelle verknüpft sind. Da in der Regel die Tabellen mit referentieller Integrität verknüpft werden, kommt dieser JOIN-Typ kaum zum Einsatz.

In diesem Fall ist nur der Typ INNER JOIN interessant:

```
Syntax:      SELECT <Spalte1> , <Spalte2> , ...  
             FROM <Elterntabelle> INNER JOIN <Kindtabelle>  
             ON <JOIN-Bedingung>  
             WHERE <Abfrage-Bedingung>
```

Beispiel: Gesucht ist die Straße, in der die Kundin Gasser wohnt.

```
SELECT Name, Strasse  
FROM Kunde INNER JOIN Adresse  
ON Kunde.KundenNr = Adresse.KundenNr  
WHERE Name = ‚Gasser‘
```

Die Bedingungen, die von der Datenbankstruktur vorgegeben sind, werden in der ON-Klausel genannt, während Bedingungen, die der Abfrager stellt, weiterhin in die WHERE-Klausel kommen.

JOIN ist eine fundamentale Operation mit praktischer Bedeutung, wenn es um die Verknüpfung von mehreren Tabellen geht. Es handelt sich hierbei um eine

Kombination aus verschiedenen anderen Operationen wie Kreuzprodukt, Selektion und Projektion.

Die weiteren Beispiele behandeln wieder die Datenbank *EDV\_Kurse*.

**Bemerkung:**

Ich werde den Verbund von Tabellen in den zukünftigen Beispielen sowohl über das Kreuzprodukt als auch mit **INNER JOIN** besprechen.

### 3.5.1 Einfacher Equijoin mit zwei Tabellen

Beispiel: Gesucht werden die Namen aller Schulen und die dort stattfindenden Kurse, sortiert nach Schule und Kurs.

```
SELECT Schule, Kurs
FROM Ort, Kurs
WHERE Ort.OrtID = Kurs.OrtNr ORDER BY Schule ASC,
Kurs ASC
```

Beispiel: Gesucht werden alle Schulorte und die zugehörigen Kurse

```
SELECT Ort, Kurs
FROM Ort INNER JOIN Kurs
ON Ort.OrtID = Kurs.OrtNr
```

**Bemerkung:**

Zur Verkürzung des Abfragetexts können für die Tabellen in der FROM-Komponente auch Alias-Namen vergeben werden. Diese Alias-Namen können bereits in der SELECT-Komponente verwendet werden, auch wenn sie erst in der FROM-Komponente definiert werden.

Beispiel: Gesucht werden die Namen aller Schulen und die dort stattfindenden Kurse, sortiert nach Schule und Kurs.

```
SELECT Schule, Kurs
FROM Ort O, Kurs K
WHERE O.OrtID = K.OrtNr ORDER BY Schule ASC, Kurs
ASC
```

Beispiel: Gesucht ist der Veranstaltungsort des Kurses ‚Projektmanagement‘.

```
SELECT Kurs, Ort
```

```
FROM Ort INNER JOIN Kurs
ON Ort.OrtID = Kurs.OrtNr
WHERE Kurs.Kurs = 'Projektmanagement'
```

### Übung 3.5:

Stelle die obigen Abfragen jeweils in der alternativen Variante Kreuzprodukt bzw. INNER JOIN.

### 3.5.2 Einfacher Equijoin über $n > 2$ Tabellen

Beispiel: Gesucht sind die Kurse, die Frau Hofmann belegt, sortiert nach Kurs.

```
SELECT Nachname, Kurs
FROM Teilnehmer P, besucht B, Kurs K
WHERE (P.Nachname = 'Hofmann') AND (PTeilnehmerID =
BTeilnehmerNr) AND (BKursNr = KKursNr)
ORDER BY Kurs ASC
```

**Bemerkung:** Bei mir funktioniert nicht die Bezeichnung Teilnehmer T. Warum??

Beispiel: Gesucht sind die Kurse, die Frau Hofmann belegt, sortiert nach Kurs.

```
SELECT Nachname, Kurs
FROM Teilnehmer INNER JOIN (
  Kurs INNER JOIN besucht
  ON (Kurs.KursNr = besucht.KursNr)
ON (Teilnehmer.TeilnehmerID = besucht.TeilnehmerNr)
WHERE Nachname = 'Hofmann'
```

Beispiel: Gesucht sind die Kurse, die Frau Hofmann in Pforzheim belegt, sortiert nach Kurs.

```
SELECT Nachname, Kurs
FROM Teilnehmer P, besucht B, Kurs K, Ort O
WHERE (P.Nachname = 'Hofmann') AND (O.Ort =
'Pforzheim') AND (PTeilnehmerID = BTeilnehmerNr) AND
(BKursNr = KKursNr) AND (K.OrtNr = O.OrtID)
ORDER BY Kurs ASC
```

### Übung 3.6:

Stelle die obige Abfrage jeweils in der alternativen Variante Kreuzprodukt bzw. INNER JOIN.

### 3.5.3 Equijoin mit Umbenennen der Spalte

Es ist oft sinnvoll Spalten umzubenennen, um Missverständnisse auszuschließen.

Beispiel: Für jeden Teilnehmer wird die Schulungsdauer in Stunden gesucht.

```
SELECT Nachname, Vorname, Kurs, Dauer *8 AS Stunden
FROM Teilnehmer P, besucht B, Kurs K
WHERE (P.TeilnehmerID = B.TeilnehmerNr) AND (B.KursNr
= K.KursNr)
ORDER BY Nachname
```

Übung 3.7:

Löse in Kapitel 3.8.2 *Abfragen zum Thema Kreuzprodukt und Join* zur Datenbank *Kaufhaus* die **Wiederholung Kreuzprodukt**.

Übung 3.8:

Erstelle in Kapitel 3.8.2 *Abfragen zum Thema Kreuzprodukt und Join* zur Datenbank *Kaufhaus* die **SQL-Abfragen zum Kreuzprodukt**.

Übung 3.9:

Erstelle in Kapitel 3.8.2 *Abfragen zum Thema Kreuzprodukt und Join* zur Datenbank *Kaufhaus* die **SQL-Abfragen zum INNER JOIN**.

### 3.5.4 Zusatzkapitel: Vereinigung mit UNION (optional)

**Bemerkung:**

Diese Abfrage funktioniert in Access, jedoch nicht in NeoOffice.

Warum??

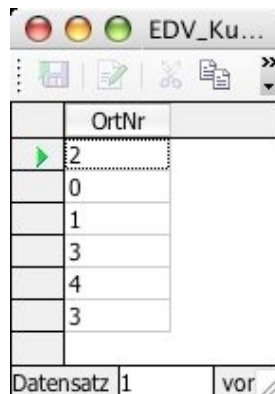
Die Datensätze von Tabellen, die identische Spalten enthalten, können durch UNION zusammengefasst werden. Der Operator UNION vereinigt die Ergebnisse zweier SELECT-Anweisungen. Sein Rückgabewert ist eine Tabelle mit allen Zeilen, die mindestens eine der beiden Anweisungen erfüllen.

**Bemerkung:**

Das folgende Beispiel erscheint als „konstruiert“. Mir ist jedoch kein besseres eingefallen.

Beispiel: Gesucht sind die Ortsnummern aller Orte, an denen Kurse ohne Voraussetzungen stattfinden.

```
SELECT OrtNr
FROM Kurs K
WHERE K.Voraussetzungen = ,keine'
```

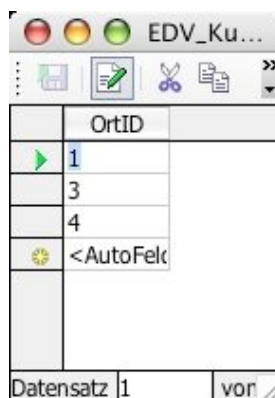


OrtNr
2
0
1
3
4
3

Abbildung 3.11: Tabelle der Ortsnummern mit Kursen ohne Voraussetzung

Beispiel: Gesucht sind die Ortsnummern der Orte, deren Namen Pforzheim oder Freiburg sind.

```
SELECT OrtID
FROM Ort O
WHERE (O.Ort = ,Pforzheim') OR (O.Ort = ,Freiburg')
```



OrtID
1
3
4
<AutoFeld>

Abbildung 3.12: Tabelle der Ortsnummern der Orte mit Namen Pforzheim und Freiburg

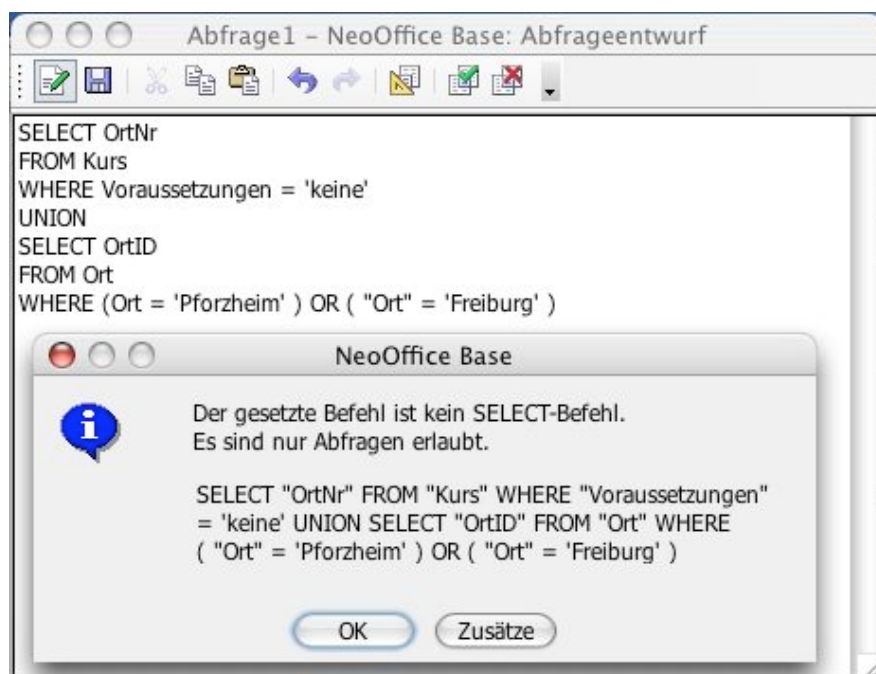
Mit der Operation UNION lassen sich diese beiden Tabelle zusammenfassen, ohne dass doppelte Datensätze auftreten.

Beispiel: Gesucht sind die Ortsnummern aller Orte, an denen Kurse ohne Voraussetzungen stattfinden oder deren Namen Pforzheim oder Freiburg sind.

```
SELECT OrtNr
FROM Kurs K
WHERE K.Voraussetzungen = ,keine'

UNION

SELECT OrtID
FROM Ort O
WHERE (O.Ort = ,Pforzheim') OR (O.Ort = ,Freiburg')
```



**Abbildung 3.13:** Tabelle der Ortsnummern mit Kursen ohne Voraussetzung oder mit Namen Pforzheim und Freiburg  
(Abfrage funktionierte nicht)

Übung 3.10:

Löse in Kapitel 7.8.3 *Abfragen zum Thema UNION* zur Datenbank *Kaufhaus* die **Verknüpfung von Tabellen**.

### 3.6 Geschachtelte SQL-Abfragen (Unterabfragen, Subqueries)

SQL erlaubt verschachtelte Abfragen.

```
Syntax:      SELECT <Spalte>
              FROM <Tabelle1>
              WHERE <Schlüsselattribut> IN
                (SELECT <Schlüsselattribut>
                 FROM <Tabelle2>
                 WHERE <Bedingung> )
```

Beispiel: Gesucht wird der Name der Schule, in der der Kurs „Vertiefung Java“ stattfindet.

```
SELECT Schule
FROM Ort
WHERE OrtID IN
  (SELECT OrtNr
   FROM Kurs
   WHERE Kurs LIKE 'Vertiefung Java');
```

Die innere Abfrage liefert die Menge aller Datensätze mit den *OrtNr* derjenigen Orte, in denen der Kurs „*Vertiefung Java*“ stattfindet. Diese innere Abfrage ist in eine äußere Abfrage eingebettet, die die *OrtID* jedes Tupels aus *Ort* mit dieser Menge vergleicht. Falls die *OrtID* in dieser Menge vorkommt, ist die entsprechende *Schule* Teil des Ergebnisses.

#### Bemerkung:

Die Schachtelung von Abfragen stellt eine Alternative zu JOIN dar. Die obige Aufgabenstellung lässt sich mit Hilfe von JOIN folgendermaßen lösen.

Beispiel: Gesucht wird der Name der Schule, in der der Kurs „Vertiefung Java“ stattfindet (mit Hilfe von JOIN)

```
SELECT Kurs, Schule
FROM Kurs, Ort
WHERE (OrtID = OrtNr) AND (Kurs = 'Vertiefung Java')
```

### Übung 3.11

Diskutiere beide Lösungswege! Welche Vor- bzw. Nachteile haben sie?

#### JOIN:

Vorteil: Ausgabe von Attributen aus mehreren Tabellen leicht möglich.

Nachteil: Bildung des Kreuzprodukts ist aufwändig.

### Geschachtelte SQL-Abfrage:

- Vorteil:** Es werden nur die Datensätze herausgesucht, die wirklich gebraucht werden.
- Nachteil:** Als Ergebnis können immer nur Teile **einer** Tabelle ausgegeben werden.

**Beispiel:** Gesucht werden alle Kurse, die in Pforzheim stattfinden.

```
SELECT Kurs
FROM Kurs
WHERE OrtNr IN
  (SELECT OrtID
   FROM Ort
   WHERE Ort LIKE ‚Pforzheim’);
```

**Beispiel:** Gesucht wird jede Schule, an der ein Kurs stattfindet, der Programmiererfahrung verlangt.

```
SELECT Schule
FROM Ort
WHERE OrtID IN
  (SELECT OrtNr
   FROM Kurs
   WHERE Voraussetzungen = ‚Programmiererfahrung’);
```

### Übung 3.12

Formuliere die obige Abfrage in SQL mit Hilfe von Kreuzprodukt bzw. INNER JOIN.

**Beispiel:** Gesucht werden alle Kurse, die nicht in Sindelfingen stattfinden.

```
SELECT Kurs
FROM Kurs
WHERE OrtNr NOT IN
  (SELECT OrtNr
   FROM Ort
   WHERE Ort = ‚Sindelfingen’);
```

**Bemerkung:**  
Identisch mit obiger Abfrage ist die folgende Abfrage:

**Beispiel:** Gesucht werden alle Kurse, die nicht in Sindelfingen stattfinden.

```
SELECT Kurs
FROM Kurs
WHERE OrtNr IN
  (SELECT OrtNr
   FROM Ort
   WHERE Ort <> ‚Sindelfingen’);
```

Abfragen können auch mehrfach geschachtelt sein.

Beispiel: Gesucht werden alle Teilnehmer des Kurses „Grundlagen Datenbanken“

```
SELECT Nachname, Vorname
FROM Teilnehmer
WHERE TeilnehmerID IN
  (SELECT TeilnehmerNr
   FROM besucht
   WHERE KursNr IN
     (SELECT KursNR
      FROM Kurs
      WHERE Kurs = 'Grundlagen Datenbanken'))
```

Übung 3.13:

Löse in Kapitel 3.8.4 *Abfragen zum Thema Geschachtelte Abfragen* zur Datenbank *Kaufhaus* die **Wiederholung**.

Übung 3.14:

Löse in Kapitel 3.8.4 *Abfragen zum Thema Geschachtelte Abfragen* zur Datenbank *Kaufhaus* die **Aufgabe 2**.

Übung 3.15:

Löse in Kapitel 3.8.4 *Abfragen zum Thema Geschachtelte Abfragen* zur Datenbank *Kaufhaus* die **Aufgabe 3**.

Übung 3.16:

Löse in Kapitel 3.8.5 *Aufwandsbetrachtungen* zur Datenbank *Kaufhaus* die **Aufgabe 1 und Aufgabe 2**.

## 3.7 Aggregatfunktionen und Gruppenbildung

### 3.7.1 Zählfunktion

Syntax: **SELECT COUNT ([DISTINCT] <Spaltenliste | \*>  
FROM <Tabelle>**

Beispiel: Gesucht wird die Anzahl der Fortbildungsteilnehmer.

```
SELECT COUNT (*)  
FROM Teilnehmer
```

Beispiel: Wie viele verschiedene Schulorte existieren?

```
SELECT COUNT (DISTINCT ORT) AS Schulorte  
FROM Ort
```

### 3.7.2 Arithmetische Funktionen

Syntax: **SELECT AVG ({numerische Spalte | Arithmetischer  
Ausdruck mit numerischen Spalten})  
FROM <Tabelle>**

Beispiel: Gesucht wird die Durchschnittsdauer aller Fortbildungsveranstaltungen.

```
SELECT AVG (Dauer) AS Durchschnittsdauer  
FROM Kurs
```

Syntax: **SELECT SUM ({numerische Spalte | Arithmetischer  
Ausdruck mit numerischen Spalten})  
FROM <Tabelle>**

Beispiel: Gesucht ist die Gesamtdauer aller Fortbildungen, die in Freiburg stattfinden.

```
SELECT SUM (Dauer) AS Gesamtdauer_in_Freiburg  
FROM Kurs K, Ort O  
WHERE (K.OrtNr = O.OrtID) AND (O.Ort = ‚Freiburg‘)
```

### 3.7.3 Min- / Max - Funktionen

Syntax: **SELECT MAX ({numerische Spalte | Arithmetischer Ausdruck mit numerischen Spalten})  
FROM <Tabelle>**

Syntax: **SELECT MIN ({numerische Spalte | Arithmetischer Ausdruck mit numerischen Spalten})  
FROM <Tabelle>**

Beispiel: Gesucht wird die kürzeste Kursdauer.

```
SELECT MIN (Dauer) AS Minimale_Kursdauer  
FROM Kurs
```

#### Übung 3.17:

Löse in Kapitel 3.8.6 *Abfragen zum Thema Aggregatfunktionen* zur Datenbank *Kaufhaus* die **Aufgabe 1** bis **Aufgabe 12**.

### 3.7.4 Verschiedenes

Beispiel: Gesucht wird die durchschnittliche Kursdauer.

```
SELECT AVG(Dauer) AS MittlereDauer  
FROM Kurs
```

Beispiel: Gesucht werden alle Kurse mit Dauer, die länger dauern als der Durchschnitt.

(Dazu brauchst du eine **Unterabfrage**, welche die durchschnittliche Dauer ermittelt. Diese wird in Klammern gesetzt und zwar an der Stelle, wo das Ergebnis dieser Unterabfrage gebraucht wird.)

```
SELECT Kurs, Dauer  
FROM Kurs  
WHERE Dauer > (SELECT AVG(Dauer) FROM Kurs)
```

### 3.7.5 Zusatzkapitel: Gruppenbildung in SQL-Abfragen (optional)

In den vorangegangenen Beispielen wurden die Aggregatfunktionen immer auf eine ganze Tabelle angewandt. Daher bestand das Abfrageergebnis immer nur aus einem Datensatz. In SQL ist es aber auch möglich, eine Tabelle in Gruppen „aufzuteilen“, d.h. die Datensätze einer Tabelle in Gruppen einzuteilen, und dann die Aggregatfunktionen jeweils auf die Gruppen anzuwenden.

Syntax: **SELECT <Spalte>, <Aggregatfunktion ...>**  
**FROM <Tabelle>**  
**GROUP BY <Spalte>**

Beispiel: Es sind zwei Gruppen auszugeben: Alle verschiedenen Voraussetzungen und deren Gesamtkursdauer nach Gesamtkursdauer absteigend sortiert.

```
SELECT Voraussetzungen, SUM (Dauer) AS Gesamtdauer
FROM Kurs
GROUP BY Voraussetzungen
ORDER BY SUM (Dauer) DESC
```

Beispiel: Für jeden Kursort soll die Anzahl der Veranstaltungen ermittelt werden.

```
SELECT Ort, COUNT (Ort) AS Veranstaltungen
FROM Ort
GROUP BY Ort
ORDER BY SUM (Ort) DESC, Ort ASC
```

### 3.7.6 Zusatzkapitel: Auswahl von Gruppen (optional)

Syntax: **SELECT <Spalte>, <Aggregatfunktion ...>**  
**FROM <Tabelle>**  
**GROUP BY <Spalte>**  
**HAVING <Bedingung>**

Beispiel: Es sind alle Orte zusammen mit der Gesamtdauer auszugeben, wenn die Gesamtdauer mindestens 10 Tage lang ist.

```
FROM Kurs, Ort
WHERE Kurs.OrtNr = Ort.OrtID
GROUP BY Ort
HAVING SUM (Dauer) >= 10
```

### 3.8 Aufgaben zu SQL

Verwende in den folgenden Aufgaben die Datenbank *Kaufhaus!*

#### Bemerkung:

Diese Aufgaben zu Kapitel 3.8 wurden entnommen dem Skript von Frau A. Bierschneider-Jakobs; Michaeli Gymnasium München

#### 3.8.1 Abfragen zum Thema Projektion und Selektion

1. **SQL – Anfragen zur Projektion:** (speichere als *qryPro1a*, usw.)
  - a) Gib eine Preisliste (Artikelnummer und Verkaufspreis) heraus.
  - b) Welche verschiedenen Artikel führt das Kaufhaus?
  - c) Wie heißen die Abteilungsleiter?
  - d) Welche Informationen sind über die Abteilungen vorhanden?
  - e) Denke Dir selbst mindestens drei neue Fragen aus und stelle sie (schriftlich) Deinem Nachbarn.
  
2. **SQL – Anfragen zur Selektion:** (speichere als *qrySel2a*, usw.)
  - a) Gesucht sind alle Informationen über Herrenhose und Sommerkleid!
  - b) Welche Artikelnummer hat der Zwieback?
  - c) Welche Waren (Artikelnummer und Verkaufspreis) werden für mehr als 25 € verkauft?
  - d) Welche Artikel (Angabe der Bezeichnung) bietet das Kaufhaus an?
  - e) Gesucht sind die Artikelnummern aller Artikel mit Ausnahme der Artikelnummer 2046.
  - f) Gib die Artikelnummern und die Verkaufspreise aller Herrenhosen aus, die für höchstens 20 € verkauft werden! Der Spaltenname für die Verkaufspreise soll in der Ergebnistabelle “Sonderangebot” heißen.
  - g) Gib Artikelnummer und Verkaufspreis aller Waren aus, deren Einkaufspreis zwischen 1,50 € und 5 € liegt.
  - h) Gesucht sind Artikelnummer und Vorrat aller Artikel aus der Textil-Abteilung.

- i) Gesucht sind alle Informationen über die Abteilungen, die im zweiten Stock platziert sind oder von Frau Stiehl geleitet werden.

### 3. Umgangssprachliche Formulierung:

Formuliere folgende SQL-Anweisung umgangssprachlich:

- a) `SELECT DISTINCT Abteilungsleiter FROM Abteilung WHERE NOT(Abteilungsname = Kosmetik);`
- b) `SELECT ArtNr FROM Bestand WHERE Abteilungsname = 'Lebensmittel' AND Vorrat <= 100;`

### 4. Wiederholungsaufgaben: (speichere als *qryWie4a*, usw.)

- a) Erstelle eine Liste, die alle verschiedenen Artikel des Kaufhauses und ihren Verkaufspreis enthält!
- b) Welche Artikel kosten zwischen 5 und 10 Euro?
- c) Gib eine Liste aller Artikel des Kaufhauses aus, bei denen der Gewinn mehr als 1 Euro beträgt (bzw. zwischen 1 und 5 Euro liegt).
- d) Was kostet eine Herrenhose?
- e) Was kostet ein Kamm oder eine Seife?
- f) Welche Artikel werden für mehr als 5 Euro und weniger als 20 Euro eingekauft?
- g) Welche Informationen über die Abteilungen sind verfügbar?
- h) Welche Artikel (Angabe der Artikelnummer) führt die Lebensmittelabteilung?
- i) Welche Personen arbeiten in dem Kaufhaus?
- j) Welche Informationen sind über Monika Stiehl gespeichert?
- k) Welche Artikel werden mit Verlust verkauft?

## 3.8.2 Abfragen zum Thema Kreuzprodukt und Join

### 1. Wiederholung Kreuzprodukt

Gib alle Informationen der Kaufhausdatenbank in einer Tabelle aus! Führe dazu folgende Schritte aus und sieh dir nach jedem Schritt die Ergebnistabelle an!

- a) Bilde das Kreuzprodukt Artikel x Bestand x Abteilung.
- b) Selektiere nun alle Datensätze, die die gleiche Artikelnummer haben.
- c) Selektiere dann alle Datensätze, die den gleichen Abteilungsnamen haben.
- d) Überlege dir, wie viele Datensätze du bei jedem Schritt erhalten hast und überprüfe das Ergebnis.

## 2. SQL – Abfragen zum Kreuzprodukt

Formuliere nachfolgende Anfragen durch Verwendung eines Kreuzproduktes. Mache Dir dabei auch die Wirkung des Kreuzproduktes, der anschließenden Selektion und Projektion klar!

(Hilfe: Beginne mit `SELECT * FROM Tabelle1, Tabelle2, ...`

Sieh dir das Ergebnis der Abfrage an und baue dann die Abfrage weiter aus.) (speichere als *qryKre2a*, usw.)

- a) Erstelle eine Artikelliste mit den Informationen Artikelnummer, Artikelname und Bestand.
- b) Gib eine Liste aus, aus der hervorgeht, in welcher Abteilung (Name) ein Artikel (Nummer und Name) verkauft wird.
- c) Welche Artikel (Nummer und Bezeichnung) werden in der Lebensmittelabteilung verkauft?
- d) Wie heißt die Abteilung, in der Herrenhosen verkauft werden?
- e) Gib alle Informationen über die Artikel aus, die von Monika Stiehl verkauft werden.
- f) Wie viele Packungen Zwieback sind noch vorrätig?
- g) In welchem Stockwerk wird Räucherlachs verkauft?
- h) Wie viele Artikel führt die Kosmetik-Abteilung?
- i) Was ist der teuerste Artikel des Kaufhauses und was kostet er?
- j) Welchen Verkaufswert haben alle vorhandenen Herrenhosen?
- k) Welchen Gesamtwarenwert (Einkaufspreise) umfasst das Kaufhaus?

### 3. SQL – Abfragen zu INNER JOIN

Formuliere die obigen Anfragen aus 2. durch Verwendung von INNER JOIN. (speichere als *qryInJo3a*, usw.)

#### 3.8.3 Abfragen zum Thema UNION

##### 1. Verknüpfung von Tabellen:

Eine SQL-Anfrage liefert als Ergebnis eine Tabelle zurück, die wieder als Eingabe einer neuen Anfrage verwendet werden kann. Insbesondere können zwei Tabellen über die Mengenoperatoren

- a) UNION (Vereinigung zweier Tabellen)
- b) INTERSECT (Schnittmenge zweier Tabellen)
- c) EXCEPT (Tabelle 1 ohne die Datensätze von Tabelle 2)

verknüpft werden.

Probiere alle drei Befehle aus! Denke Dir für jeden Befehl schriftlich eine Aufgabe aus und stelle sie Deinem Nachbarn.

#### 3.8.4 Abfragen zum Thema Geschachtelte Abfragen

##### 1. Wiederholung: (speichere als *qryGesch1a*, usw.)

Führe nachfolgende Abfragen aus. Verwende gegebenenfalls einen Join.

- a) Welcher Abteilungsleiter ist für Textilien zuständig?
- b) In welchem Stockwerk werden T-Shirts verkauft?
- c) Wie viele Artikel mit der Nummer 1401 sind noch vorrätig?
- d) Gesucht sind Artikelnummer und Vorrat aller Artikel aus der Textil-Abteilung.
- e) Gesucht sind alle Informationen über die Abteilungen, die im zweiten Stock platziert sind oder von Frau Stiehl geleitet werden.

##### 2. Formuliere nachfolgende Anfragen durch Verwendung einer geschachtelten SQL Abfrage: (speichere als *qryGesch2a*, usw.)

- a) In welcher Abteilung wird Zwieback verkauft?

- b) Gib die Bezeichnungen und die Artikelnummern aller Artikel aus, die nicht mehr als der Artikel mit der Artikelnummer 1401 kosten!
  - c) Gesucht sind Bezeichnung und Verkaufspreis aller Artikel, die in der Lebensmittelabteilung verkauft werden!
  - d) Welche Produkte (Angabe der Artikelnummer) werden im Erdgeschoss verkauft?
  - e) Welche Produkte (Angabe der Artikelnummer und Bezeichnung) werden im Erdgeschoss verkauft?
3. Welche Aufgaben von Aufgabe 2 können auch mit Hilfe eines Joins gelöst werden? (speichere als *qryGesch3a*, usw.)

### 3.8.5 Aufwandsbetrachtungen: JOIN oder Geschachtelte Abfrage

1. In welchem Stockwerk wird Räucherlachs verkauft? (speichere als *qryAufw1a*, usw.)
  - a) Formuliere diese Abfrage mit Hilfe eines JOIN.
  - b) Wie viele Datensätze werden dabei vom Rechner in der WHERE Anweisung schrittweise überprüft?
  - c) Wie wirkt sich diese Art der Abfrage bei großen Tabellen (echtes Kaufhaus) aus?
2. Formuliere Aufgabe 1 mit einer geschachtelten SQL Anweisung. Wie viele Datensätze werden nun ausgewählt und geprüft? (speichere als *qryAufw2*, usw.)

### 3.8.1 Abfragen zum Thema Aggregatfunktionen

Formuliere nachfolgende Abfragen an die Kaufhausdatenbank in SQL!  
(speichere als *qryAggl*, usw.)

1. Gib den Preis des billigsten Artikels aus.
2. Gib den billigsten Artikel mit der Bezeichnung „Billigster\_Artikel“ und

- dem Verkaufspreis aus. Verwende dazu die Abfrage aus 1 als Unterabfrage.
3. Wie viele Artikel (mit verschiedenen Artikelnummern) bietet das Kaufhaus an?
  4. Wie viele einzelne Artikel liegen im gesamten Kaufhaus auf Vorrat?
  5. Wie viele Abteilungen werden von Monika Stiehl geleitet?
  6. Was kostet die billigste Herrenhose?
  7. Was ist der durchschnittliche Einkaufspreis aller Artikel?
  8. Welche Artikel mit dem Anfangsbuchstaben 'S' gibt es?
  9. Welche Artikelnummern haben an der 2. Stelle eine Null?
  10. Heißt die Abteilungsleiterin „Stiehl“ oder „Steihl“?
  11. Gib alle Artikel, die das Kaufhaus führt, ohne Duplikate aus, sortiert nach Bezeichnung.
  12. Gib alle Informationen über die Artikel aus sortiert nach dem Einkaufspreis.
  13. Gesucht ist der Preis des teuersten Artikels.
  14. Wie viele Abteilungen hat das Kaufhaus?
  15. Gib den durchschnittlichen Vorrat an Artikeln aus.
  16. Was kostet in dem Kaufhaus ein Hemd durchschnittlich?
  17. Was kostet die billigste Herrenhose?
  18. Wie viele Artikel mit unterschiedlicher Bezeichnung führt das Kaufhaus?
  19. Wie viele Artikel führt die Kosmetik-Abteilung?
  20. Was ist der teuerste Artikel des Kaufhauses und was kostet er?
  21. Welchen Verkaufswert haben alle vorhandenen Herrenhosen?
  22. Welchen Gesamtwarenwert (Einkaufspreise) umfasst das Kaufhaus?