Mit einem durchgehenden Beispiel zur Datenbank eines Unternehmens vermittelt dieses Buch wichtige Grundsätze für den Datenbankenentwurf und zur Datenbankerstellung.

Schwerpunkte von der Datenbankabfrage, -manipulation, -modellierung, bis hin zur -verbesserung, werden im *Oracle SQL Developer* anschaulich nah gebracht.

`CREATE TABLE`, `ALTER TABLE`, `SELECT`, `INSERT`, `CREATE VIEW`, Datentypen, Beziehungen, Schlüssel, Datenintegrität und vieles mehr werden innerhalb der **Praktikumsaufgaben 1 bis 5 von Prof. Dr. W. Gerken,** aus der Veranstaltung **Informationssysteme I,** ausführlich behandelt und erklärt.

Hochschule für Angewandte Wissenschaften Hamburg
Sommersemester 2015

€ 6,49 [D] € 6,69 [A]

0011607912

BoD™ BOOKS on DEMAND

Steven Illg Informationssysteme I

Informationssysteme I

Datenbankerstellung mit SQL

Steven Illg

Informationssysteme I

Datenbankerstellung mit SQL

Informationssysteme I

Datenbankerstellung mit SQL

Ausarbeitung der Praktikumsaufgaben 1 bis 5

Steven Illg
bei Prof. Dr. W. Gerken

Sommersemester 2015
Bachelor Wirtschaftsinformatik im 2. Semester

Hochschule für Angewandte Wissenschaften Hamburg
Department Technik und Informatik

BoD
Books on Demand GmbH

Covergestaltung Steven Illg
Layout und Gestaltung Steven Illg

Druck und Bindung:
Books on Demand GmbH, Norderstedt

BoD-Nummer: 00 1160791 2

http://www.bod.de/

Für meinen Bruder Jason

und meine treuen Uni-Kompagnons
Mert und Nemon

Cağtay, Pouria und Resul

Zum Inhalt des Buchs

Die Lösungen zu den Praktikumsaufgaben 1 bis 5 in diesem Buch, sind im Zeitraum von März bis Juni 2015, im zweiten Semester des Studiengangs *Bachelor Wirtschaftsinformatik,* an der *HAW Hamburg* entstanden. Die Konstruktion der Aufgabenstellungen, die Aufsicht und die Kontrolle der Lösungen leitete Professor Dr. W. Gerken in einem vierzehntägigen Intervall.

Die Lösungen dienen ausschließlich als exemplarisches Beispiel und sind keineswegs absolut und vollkommen. Es gibt viele verschiedene Wege und Methoden, wie man eine Datenbank entwerfen und eine SQL-Abfrage *(wobei SQL für „structured query language" steht)* formulieren kann. Ausschlaggebend sind hierbei das Ziel und die Adressaten der Datenbank. Abgesehen davon ist dieses Buch nicht als Lehrbuch zu verstehen und garantiert keine Vollständigkeit des gelehrten Lernstoffes der Veranstaltung *Informationssysteme I (fehlende Inhalte aus der Veranstaltung: Normalformen, funktionale Abhängigkeit, Praktikum 6 und 7).*

Wie das Buch zu lesen ist

Mit einem Rechteck eingerahmte Texte, bei denen die *obere-rechte* und die *untere-linke* Ecke fehlen, heben die Aufgabenstellungen von Prof. Dr. W. Gerken hervor.

Aufgabenstellung

Bei Rechtecken mit oberer und unterer Formkontur, handelt es sich um hervorgehobene *SQL-Statements* und *SQL-Abfragen.* Jeder SQL-Code, oder Datenbank betreffender Bezug, ist in `Courier New` geschrieben.

```
SQL-Statement
```

Bei Rechtecken, die *unten-rechts* umgeklappt sind, handelt es sich um allgemeine Hinweise zur Arbeit mit Datenbanken und zum Inhalt des gelehrten Lernstoffs.

Hinweis

Inhaltsverzeichnis

```
SELECT PRAKTIKUM FROM INFORMATIONSSYSTEME
    WHERE praktikumsaufgaben =
```

‘Praktikumsaufgabe 1‘

```
    AND autor = 'Steven Illg';
```

Termin: 24.März 2015
Abgabe: 13.April 2015

Aufgabe 1.1. Der erste Datenbankkontakt

Können Sie beliebige Tupel in die Beispieldatenbank eingeben?
Welche Einschränkungen gibt es für die Attributwerte?

- Der `Name` darf aus einer beliebigen Zeichenkette bestehen, weil das Attribut als Text – `VARCHAR` – definiert wurde.
- Auch Sonderzeichen sind für `Name` zulässig.
- `Name` darf maximal 20 Zeichen lang sein – `VARCHAR(20)`.
- Bei `Name` sind weder Bindestriche in Doppelnamen, noch leere Zeichen erlaubt *(Space, Tabulator oder Absatz)*.
- `Studiengang` darf nur zwei Buchstaben haben – `CHAR(2)`.
- `Studiengang` kann nur `WI`, `AI` und `TI` sein. `SA` wird nicht hinzugefügt – ermöglicht durch eine `CHECK`-Klausel.
- Mit `VARCHAR` implementierte Attribute werden nur hinzugefügt, wenn der Text im `INSERT`-Befehl `'...'` Apostrophe hat.
- `Datum` darf als Basisdatentyp `DATE` beliebig oft vorkommen.
- Das `Datum` muss ein gültiges Datum in der Reihenfolge `TT.MM.JJ` sein *(allerdings kann für das Jahr im `INSERT`-Befehl eine vierstellige Zahl aus dem 20. und 21. Jahrhundert gewählt werden, letztlich werden aber nur die Einer und Zehner in der Datenbank angezeigt)*.
- Keine irrationalen Geburtstage *(zum Beispiel irrational: 23.15.1812)*.
- Bei `Studiengang` und `Datum` existiert keine Pflichtangabe – mit anderen Worten: Hier sind `NULL`-Werte zulässig.
- `Matrikelnummer` ist der Primary Key dieser Relation – aus diesem Grund muss und darf ein Wert nur einmal vorkommen.
- `Matrikelnummer` kann nur eine Zahl zwischen `100` und `99.999` sein – also eine Zahl mit 3 bis 5 Stellen; dies wird realisiert mit `NUMBER(5)` und der `CHECK`-Klausel `>=100`.
- `Matrikelnummer` mit $\in \mathbb{N}$ zulässig – mit $\notin \mathbb{N}$ unzulässig.
- `Matrikelnummer` darf niemals mit einer `'0'` beginnen.

Können Sie eine Ausgabesortierung nach `Name` erreichen?

Es besteht die Möglichkeit, eine, mehrere, oder alle Spalten aus einer Relation auf- bzw. absteigend nach Wert oder Alphabet zu sortieren, spezifisch nach einer oder mehreren Bedingungen zu filtern und einzelne Spalten oder Datensätze ausgeben zu lassen. Hierfür werden im Folgenden drei Varianten aufgeführt.

Zum einen kann man das *Kontextmenü* verwenden, zum anderen kann man Spezialisierungen direkt an der *Relation* oder per *Eingabe* von SQL-Statements in die Kommandozeile vornehmen.

Kontextmenü: Mit einem rechten Mausklick wird das Kontextmenü mit dem Befehl *„Sortieren …“* geöffnet. Wenn man auf den Befehl klickt, wird ein separates Fenster geöffnet, in dem man eine Sortierung nach wenigen Kriterien vornehmen kann, die sich allerdings nur bedingt an die eigene Vorstellung anpassen lassen. Somit bietet diese Methode sehr wenig Spielraum für den Nutzer. Darüber hinaus sollte man keine großen Features erwarten.

An der Relation: Lässt man sich die Relation mit dem SQL-Befehl `SELECT * FROM gerken.Studierende` ausgeben, dann kann man mit der linken Maustaste auf einen der Attribute klicken. In solch einem Fall kann die Spalte `Name` mit einem Doppelklick alphabetisch absteigend sortiert werden. Nach erneuten Doppelklicks wechselt die auf- und absteigende Sortierungsvariante. Auf diese Weise lassen sich alternativ auch `Matrikelnummer` oder `Studiengang` ganz bequem auf- und absteigend sortieren. Mit einem rechten Mausklick kann man auch hier Spezifizierungen vornehmen.

Eingabe:

Die bewährte Methode der Wahl, zur Sortierung einer Relation, sollte stets der Weg über Eingabebefehle in die Kommandozeile sein.

Mit dem Befehl

```
1    SELECT Name FROM gerken.Studierende;
```

kann man sich nur die Spalte **Name** ausgeben lassen. Der ***** statt `Name` würde hingegen umgangssprachlich für *„alle Spalten“* stehen. Auch `Studiengang`, `Datum` oder `Matrikelnummer` wären statt `Name` möglich. Mit den zusätzlichen SQL-Klauseln **ORDER BY** oder **WHERE**, kann man die Relation *sortieren* oder spezifiziert *ausgeben* lassen:

ORDER BY

Der `ORDER BY` Befehl hat dieselbe Sortierung zur Folge, die auf Seite 12 bei *„An der Relation“* geschildert wurde. Mit

```
1    SELECT Name FROM gerken.Studierende
2      ORDER BY Name [ASC | DESC];
```

kann man eine alphabetisch **aufsteigende** ***(ASC** – für „ascending“; englisch für „aufsteigend“)* oder eine alphabetisch **absteigende** ***(DESC** – für „descending“; englisch für „absteigend“)* Sortierung für `Name` vornehmen. Das `ASC` kann man übrigens auch weglassen.

Möchte man mehrere Spalten gleichzeitig sortieren, ist auch das möglich. Allerdings muss man darauf achten, in wie weit die Attributwerte solch eine Sortierung zulassen, da alle Attributwerte fest in einem *Tupel* gehören. Entscheidet man sich jedoch dafür, dann müssen die gewünschten Attribute mit einem Komma getrennt und weiterhin am Ende des Befehls notiert werden:

```
1    SELECT * FROM gerken.Studierende
2      ORDER BY Name, Datum [ASC | DESC];
```

WHERE

Mit einer `WHERE`-Klausel kann man noch spezieller formulieren, was in der Ausgabe aufgeführt werden soll. So lassen sich alle Studenten aus der Spalte `Name` ausgeben, die eine bestimmte Bedingung erfüllen – z.B.: Alle Studenten, die den `Studiengang 'WI'` studieren. Der Befehl sieht formal so aus:

```
1    SELECT Name FROM gerken.Studierende
2      WHERE Studiengang = 'WI';
```

Also so viel wie: *„Gebe mir alle Namen aus Gerkens Studierendentabelle aus, bei denen der Studiengang WI entspricht“.*

Je mehr Bedingungen oder Operatoren eingegeben werden, desto stärker wird die Relation gefilterter. Man kann sich Studierende mit einem bestimmten Geburtsdatum ausgeben lassen, oder auch diejenigen, die bereits bei einem Drittversuch für eine Klausur sind. Kunden, die ein Abonnement angefordert haben, oder bereits eine bestimmte Altersgrenze erreicht haben *(alles auch andersherum möglich)*. Die Abfragegrenzen scheinen endlos zu sein.

Hinweis zu Attribute und Relationen

Attribute bezeichnen im Allgemeinen *Eigenschaften* von Objekten aus der erlebten Umwelt. Im vorherigen Beispiel sind `Name`, `Datum` und `Matrikelnummer` die Attribute der Relation `Studierende`. Die Daten die in den Attributen gespeichert werden bezeichnet man als *Attributwerte.*

In einem Datenbankmodell werden Relationen auch *Entitäten* genannt, die Individuen, reale Objekte, Ereignisse oder abstrakte Zusammenhänge sein können. Sie sind nichts anderes als die *Tabellen* der Datenbank und müssen mindestens ≥ 2 Attribute umfassen.

Mit den exemplarisch aufgeführten Attributen und der Relation `gerken.Studierende`, sind selbstverständlich beliebige Attribute und Tabellen aus den unterschiedlichsten Projekten gemeint.

Welche Wirkung haben die Befehle COMMIT und ROLLBACK?

COMMIT [WORK | FORCE]

Mit einem COMMIT wird eine *„Freischaltung"* zum Ausdruck gebracht, die eine Veränderung in einer Datenbank permanent macht. Dies gewinnt vor allem dann an Bedeutung, wenn mehrere Personen gleichzeitig Veränderungen an einer Datenbank vornehmen möchten. Darüber hinaus wird unter einem COMMIT auch der erfolgreiche Abschluss einer *Transaktion* verstanden.

Das Schlüsselwort WORK ist in der Anweisung Optional. FORCE zwingt das DBMS zum COMMIT.

ROLLBACK [WORK | FORCE]

Der Begriff ROLLBACK kommt aus dem englischen und bedeutet so viel wie *„zurückrollen", „zurückdrehen"* oder *„annullieren"*. Im Datenbankmanagementsystem *(DBMS) „Oracle SQL Developer"* wird darunter das *Abbrechen,* bzw. *Zurücksetzen* von Transaktionen verstanden *(bis hin zum Ausgangszustand)*. Sollte man eine COMMIT-Anweisung gemacht haben, ist ein ROLLBACK nicht mehr möglich. Der ROLLBACK wird als Invers vom COMMIT bezeichnet. WORK und FORCE Regelung wie bei COMMIT.

Transaktionen *(Zustandsübergänge)*

Eine Transaktion ist eine Sammlung von Verarbeitungsschritten mit unbestimmter Größe, die als Ganzes betrachtet wird. Aus diesem Grund müssen ***alle*** Verarbeitungsschritte abgeschlossen sein, oder es wird ***keine*** abgeschlossen. Ist ein Verarbeitungsschritt also nicht abgeschlossen, wird die komplette Transaktion abgebrochen.

Aufgabe 1.2. Der Beginn der ersten Datenbank

Überlegen Sie sich ein Datenbankschema für ein Handelsunternehmen, bei dem Kunden Bestellungen für die angebotenen Produkte aufgeben. Jede Bestellung besteht aus mehreren Bestellpositionen, wobei eine Bestellposition festlegt, welches Produkt in welcher Menge von einem Kunden bestellt wurde.

Stellen Sie die notwendigen Objekttypen und die sich daraus ergebenden Tabellen dar! Welche Spalten haben die Tabellen als Primärschlüssel?

Für das Datenbankschema werden vier Relationen notwendig sein, die im Folgenden nacheinander mit Erläuterungen aufgeführt werden.

Relation 1.: Kunden

Kundennummer	Name	Straße	PLZ	Ort
Kundennummer	Vor- und Nachname des Kunden	Straße des Kunden	Postleitzahl des Kunden	Wohnort des Kunden
NUMBER	VARCHAR	VARCHAR	VARCHAR	VARCHAR
Primärschlüssel				

Die Relation `Kunden` ist auf das wichtigste Minimum reduziert. Es wären je nach Anliegen auch Attribute wie `Geburtsdatum`, `Bundesland`, `Bankverbindung`, `E-Mail` und so weiter möglich.

Es wäre sinnvoller, wenn man `Name` in `Vorname` und `Nachname` aufteilen würde, jedoch wurde im Hinblick auf den weiteren Aufgabenverlauf auf diese Unterteilung verzichtet.

Der Primärschlüssel *(Primary Key)* ist das Attribut `Kundennummer`, die eine Zahl sein muss und sich niemals wiederholen darf. `Name`, `Straße`, `PLZ` und `Ort` bestehen aus Textdaten. Die *(Basis-)*Datentypen sind somit `NUMBER` und `VARCHAR`. Dazu aber mehr im 2. Praktikum.

Relation 2.: Bestellungen

Bestellnummer	Kundennummer	Bestelldatum	Status
Bestellnummer	Der Kunde	Datum der Bestellung	Bearbeitungsstatus der Bestellung
NUMBER	NUMBER	DATE	NUMBER
Primärschlüssel	Fremdschlüssel		

Da die Kundenrelation nun feststeht, werden im Folgenden zwei relevante Relationen für die aufgegebenen Bestellungen benötigt.

Die erste Relation fasst die `Bestellungen` *allgemein* zusammen.

Die `Bestellnummer` ist der Primärschlüssel dieser Relation, daher können auch hier die Attributwerte nur einmal vorkommen. Man hat die Möglichkeit, die `Bestellnummer` mit einer `SEQUENCE` automatisch bei jeder hinzugefügten Bestellung eine Zahl nach oben zu zählen *(ein Beispiel dazu ist in der Praktikumsaufgabe 5 zu sehen)*. Aufgrund der folgenden Relation `Bestellpositionen`, bedeutet dies aber nicht, dass jeder Kunde für jeden Artikel eine neue Bestellung aufgeben muss.

Der Name und die Anschrift des Kunden wurden bewusst aus der Relation gelassen. Diese Informationen sind irrelevant, da `Kundenummer` der *Fremdschlüssel* dieser Relation ist. Das Attribut bildet somit eine Referenz, die auf den Primärschlüssel in der Kundenrelation verweist. Auf diese Weise kann eine Kundennummer, je nach Anzahl der Bestellungen, mehrfach in der Spalte erscheinen. Die zusätzlichen Kundeninformationen können durch einen Tabellenwechsel oder mit einer SQL-Abfrage bequem eingesehen oder ausgegeben werden.

Das `Datum` dokumentiert, an welchem Tag eine Bestellung aufgegeben wurde. Da an einem Tag mehrere Bestellungen aufgegeben werden können, kann sich das `Bestelldatum` auch mehrfach wiederholen.

Der `Status` einer Bestellung, ist ein Attribut, das nicht zwangsläufig von besonderer Wichtigkeit ist. Dieses Attribut gibt mit einer einstelligen Zahl zwischen `1` und `5` an, wie weit ein Bestellvorgang bearbeitet wurde. So könnte die `1` für: *„Bestellung eingegangen“,* die `2` für: *„Bestellung wird bearbeitet“,* die `3` für: *„Bestellung wird im Paketzentrum bearbeitet“,* die `4` für: *„Paket wurde verschickt“* und die `5` für: *„Bestellung abgeschlossen“* stehen.

Relation 3.: Bestellpositionen

Bestellnummer	Artikelnummer	Menge
Bestellnummer	Artikelnummer	Bestellmenge der Artikel
NUMBER	NUMBER	NUMBER
Fremdschlüssel	Fremdschlüssel	
\ Primärschlüssel /		

In der zweiten bestellrelevanten Relation `Bestellpositionen`, werden die Einzelheiten zu den einzelnen Bestellungen festgehalten. Es wird also explizit aufgeführt, welche Artikel in einer Bestellung enthalten sind und wie oft sie bestellt wurden. Da `Bestellnummer` ein Fremdschlüssel dieser Relation ist, referenziert diese Spalte auf den Primärschlüssel in der vorherigen Tabelle `Bestellungen`. Dank dieser Konstellation kann der Kunde so viele Artikel bestellen, wie er möchte, ganz ohne Übersichtlichkeitseinbußen und Redundanzen in der Datenbank. Um die Redundanz in dieser Relation komplett zu verhindern, muss man sich für einen *zusammengesetzten Primärschlüssel* entscheiden. Dieser besteht aus `Bestellnummer-Artikelnummer`. Wäre dieser nicht gegeben, könnte man exemplarisch für `Bestellnummer 1` und für `Artikelnummer 12`, die Mengen **2** und **1** eintragen. Dies würde zu zwei Tupel führen, die man auch in einem zusammenfassen kann: `Bestellnummer 1`, `Artikelnummer 12`, `Menge` **3**. Somit dürfen sich bei einem zusammengesetzten Primärschlüssel die Attributwerte von zwei verknüpften Attributen nicht wiederholen.

Zusammenfassend: Für einen Bestellvorgang sind zwei Relationen entscheidend – eine, die allgemeine Informationen zur Bestellung listet *(`Bestellnummer` und `Kundennummer`)* und eine zweite, die festhält, was und in welcher Menge bestellt wurde. Salopp gesagt hält die zweite Relation fest, was genau hinter einer `Bestellnummer` steckt.

Die Attribute der beiden Relationen umschreiben allgemein: **Wann** *(`Bestelldatum`)* wurde **was** *(`Artikelnummer`)* **von wem** *(`Kundenummer`)* und **in welcher Menge** *(`Menge`)* bestellt. Natürlich kann man die Attribute je nach Ziel wählen, benennen und anpassen.

Relation 4.: Artikel

Artikelnummer	Artikelname	Produktgruppe	Verkaufspreis
Artikelnummer	Name des Artikels	Produktgruppe des Artikels	Verkaufspreis des Artikels
NUMBER	VARCHAR	VARCHAR	NUMBER(%,2)
Primärschlüssel	UNIQUE		

Auch bei der vierten Relation `Artikel` fokussieren sich die Attribute auf das wichtigste Minimum. Es wären unter anderem auch `Zulieferer`, `Herstellungsfirma`, `Marke`, `Lagerbestand`, `Verkaufsrang`, `Einkaufspreis` und viele weitere Attribute möglich.

Man kann ohne weiteres erkennen, dass der Primärschlüssel in dieser Relation die `Artikelnummer` ist. Der `Artikelname` ist `UNIQUE`, um Redundanzen zu meiden *(Datenintegrität, siehe Seite 110)*. Anstelle dessen wäre auch ein zusammengesetzter Primärschlüssel denkbar.

Der Datentyp für `Artikelnummer` ist `NUMBER`, für `Artikelname` und `Produktgruppe` `VARCHAR` und für `Verkaufspreis` `NUMBER(%,2)`, wobei `%` hier eine beliebige Zahl mit $\in \mathbb{N}$ meint. Mit Hilfe von `NUMBER(%,2)` kann man für den `Verkaufspreis` statt einer herkömmlichen Dezimalzahl nur eine Zahl in ganzen `Euros` *(das `%` vor dem Komma)* und `Cent` auflisten *(die 2 für zwei Nachkommastellen)*.

Bestehen zwischen den vier entworfenen Relationen Beziehungen in Form von inhaltlich identischen Spalten?

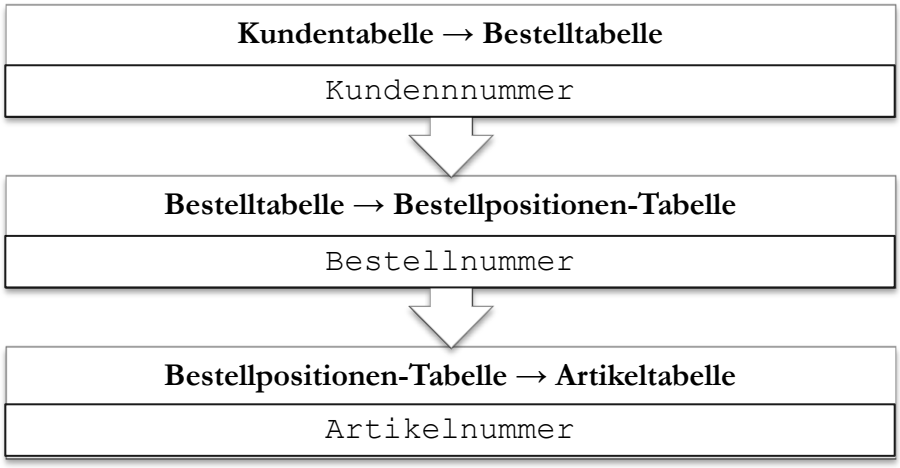
Die Kunden- und Artikeltabelle sind vollständig inkongruent. Das bedeutet, dass sich jedes Attribut nur in seiner für ihn vorbestimmten Relation befindet. Aus diesem Grund bestehen hier keine Beziehungen in Form von inhaltlich identischen Spalten. Sie sind also *disjunkt*.

Anders sieht es aus, wenn man die Bestell- und die Bestellpostionen-Tabellen zwischen die Kunden- und Artikeltabelle *schaltet*.

Die Kundentabelle steht in einer *1:n-Beziehung* mit der Bestelltabelle, da jeder Kunde mehrere Bestellungen aufgeben kann, aber jede Bestellung nur genau einem Kunden zugeordnet wird. Die Bestelltabelle steht wiederum in einer 1:n-Beziehung mit der Tabelle für die Bestellpositionen, da auch hier jede Bestellung mehrere Bestellpositionen umfasst, aber jede Bestellposition nur einer Bestellung und somit nur einem Kunden zugeordnet wird. Die letzte Beziehung zwischen der Bestellpostionen- und der Artikeltabelle ist ebenfalls eine 1:n-Beziehung, da jede Bestellposition mehrere Artikel beinhalten kann, aber jeder Artikel nur einer Bestellposition zugeordnet wird.

Aufgrund der vorherrschenden Beziehungen kommt in der Bestelltabelle eine Spalte aus der Kundentabelle und in der Bestellpositionen-Tabelle eine Spalten aus der Bestell- und eine aus der Artikeltabelle doppelt vor.

Eine Übersicht gewährt folgende Grafik:



Man kann deutlich einen *roten Faden* erkennen, der sich durch die Kunden-, Bestell-, Bestellpositionen- und die Artikeltabelle zieht und sich einen *Weg* von der Kunden- zur Artikeltabelle bahnt. Dieser Weg wird durch die Bestellpositionen-Tabelle ermöglicht, die die *n:m-Beziehung* zwischen der Bestell- und der Artikeltabelle realisiert.

Hinweis zu den Beziehungsformen

1:1-Beziehung: Eine eher seltenere Beziehungsform, die mit Hilfe eines zusammengesetzten Primärschlüssels oder mit einer Primär-Fremdschlüssel-Konstellation realisiert wird. Das Fremdschlüssel-Attribut muss im zweiten Fall allerdings `UNIQUE` sein, damit sich die Attributwerte nicht wiederholen können.

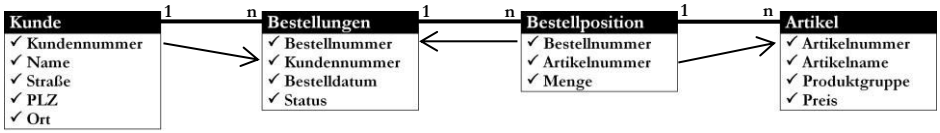
1:n-Beziehung: Die wahrscheinlich häufigste Beziehungsform, die mit Hilfe eines Primär- und eines Fremdschlüssels realisiert wird. Beispiele sind oben zu sehen.

n:m-Beziehung: Diese Beziehungsform lässt sich nur über das Einfügen einer *Zwischentabelle* realisieren. Die Relation `Bestellpositionen` ist ein Beispiel für solch eine Zwischentabelle. Beachte insbesondere den zusammengesetzten Primärschlüssel.

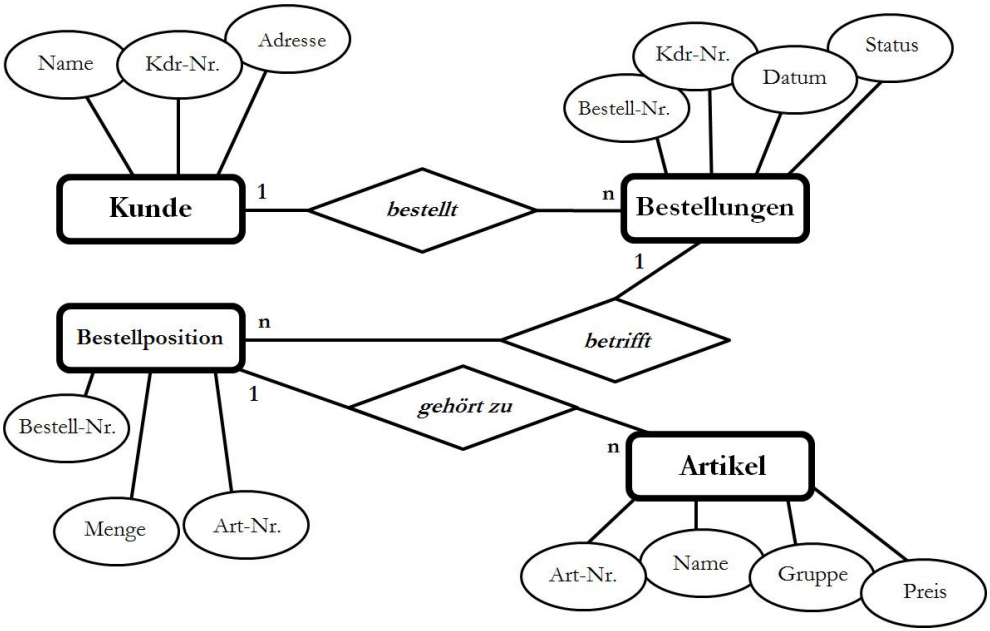
Überlegen Sie sich eine geeignete grafische Notation, um die Beziehungen der vier Relationen darzustellen!

Auf der nachfolgenden Seite sind zwei Varianten aufgeführt, die die Beziehungen grafisch darstellen. Das erste Beispiel ist das *Konzeptionelle Datenbankmodell,* das zweite hingegen das *relationale Datenbankmodell.* Das zweite ist das gebräuchlichere Modell in der Datenbankpraxis und wird für *ERM's* bzw. *ER-Modelle (Entity-Relationship-Modell)* verwendet.

Variante 1.: Konzeptionelles Datenbankmodell



Variante 2.: Relationales Datenbankmodell



Die Attribut- und Tabellenbezeichnungen sind aus Übersichtsgründen abgekürzt, zusammengefasst oder leicht verändert. Wenn man mit einem DBMS arbeitet, ist dies nicht ohne Weiteres möglich. Die Bezeichnungen müssen immer identisch bleiben, es sei denn, man weist einer Bezeichnung vorübergehend eine Neue zu. Dies geschieht über den Befehl `AS`, der jedoch optional ist. Wie man sich auch entscheidet, die neue Bezeichnung (Alias) muss immer direkt hinter der alten Bezeichnung notiert werden.

```
SELECT PRAKTIKUM FROM INFORMATIONSSYSTEME
    WHERE praktikumsaufgaben =
```

‘Praktikumsaufgabe 2‘

```
    AND autor = 'Steven Illg';
```

Termin: 13.April 2015
Abgabe: 27.April 2015

Aufgabe 2.1. Der CREATE TABLE Befehl

Erstellen Sie mit dem Befehl CREATE TABLE ... Ihre Tabellen aus der Praktikumsaufgabe 1. Achten Sie auf Primär- und Fremdschlüssel.

Im Folgenden werden die CREATE TABLE Befehle mit Erläuterungen für die vier Relationen Kunden, Bestellungen, Bestellpositionen und Artikel aus der Praktikumsaufgabe 1 nacheinander aufgeführt. Um die Inhalte zu einem späteren Zeitpunkt effizient und in kürzester Zeit nachvollziehen zu können, werden im gesamten Verlauf keine Abkürzungen und nur deutschsprachige Tabellen- bzw. Attributbezeichnungen verwendet.

Um die Textlänge so kurz wie möglich zu halten, wird bei den einzelnen Befehl-Erläuterungen auf doppelte Inhalte verzichtet. Allein für den ersten CREATE TABLE Befehl, für die Relation der Kunden, wird eine ausführliche Erläuterung erstellt, die man nahezu vollständig auf die übrigen Befehle übertragen kann. Bei analogen Zeilen wird entsprechend ein Hinweis auf den betroffenen CREATE TABLE Befehl gebracht oder zusammenfassend die wichtigsten Punkte genannt.

Relation 1: Kunden

```
1    CREATE TABLE Kunden (
2      Kundennummer NUMBER(7) NOT NULL
3        CHECK (Kundennummer >=1000),
4      Name VARCHAR2(55) NOT NULL,
5      Straße VARCHAR2(55) NOT NULL,
6      PLZ VARCHAR2(5) NOT NULL,
7      Ort VARCHAR2(45),
8      CONSTRAINT Kunden_PK
9        PRIMARY KEY (Kundennummer)
10   );
```

Jeder Befehl, der die Aufforderung eine Tabelle zu erstellen erteilt, beginnt wie hier in Zeile 1. mit `CREATE TABLE`. Darauf folgt der Name der Relation *(hier `Kunden`),* die erstellt werden soll. Umgangssprachlich bedeutet Zeile 1. so viel wie: *„Erstelle eine Tabelle, die den Namen ‚Kunden' trägt"*. Die in der Regel blau hervorgehobenen Worte symbolisieren die reservierten *Schlüsselworte* des *„Oracle SQL Developer"*.

Die geöffnete Klammer signalisiert den Anfang aller Attribute und Bedingungen, die für diese Relation implementiert werden sollen. Die Klammer kann in Zeile 1., aber auch in Zeile 2. stehen. Die geschlossene Klammer in Zeile 10. signalisiert wiederum das Ende aller Bedingungen. Somit werden alle Attribute und Bedingungen durch die Klammern in einem Block zusammengefasst. Sie geben an, welche Spalten in der Tabelle enthalten sein sollen und welche Bedingungen für die jeweiligen Attributwerte geltend gemacht werden. Die Eingabe in der Klammer ist also der *Bauplan* der Relation, der mit dem Befehl

```
1     CREATE TABLE <Tabellenname> (
2       <Attributname> [Bedingung],
n       ...
n+1   );
```

ausgeführt wird *(vgl. Klasse aus PR1)*. Das Semikolon am Ende des Befehls ist besonders zu beachten, welches an dieser Stelle notiert werden muss.

Ab Zeile 2. beginnen die *Deklarationen* der Attributnamen und die Implementierung der Bedingungen, die in der Praktikumsaufgabe 1 für die Kundentabelle festgelegt wurden.

Mit dem Anlegen der Spalte `Kundennummer`, wird gleichzeitig der Name des Attributs festgelegt. Die erste Position für `Kundennummer` passt hervorragend, da das Attribut später mit einem Primärschlüssel versehen wird. Direkt im Anschluss des Attributnamens folgen die Bedingungen, die den Eingaberahmen für die Attributwerte dieser Spalte limitieren. Eine Bedingung ist der *Datentyp,* der hier `NUMBER(7)` ist, was so viel bedeutet wie: *„In der Spalte ‚Kundennummer' darf nur eine Zahl mit maximal sieben Stellen eingetragen werden."* Statt `NUMBER` wäre auch `INT` *(für Integer)* möglich, dies hätte allerdings eine Zahlenlänge von 10 Stellen zur Folge. Die Speichergröße eines Integers beträgt 4 Byte und umfasst alle

Zahlen zwischen -2^{31} (–2.147.483.648) und $2^{31}-1$ (2.147.483.647). Nach `NUMBER(7)` folgt die zweite Einschränkung für die Attributwerte – der `NOT-NULL-CONSTRAINT`. Umgangssprachlich bedeutet dieser: *„Für ‚Kundennummer' muss immer ein Attributwert eingegeben werden."* Durch diese Regelung wird die Eingabe eines Attributwertes zwingend erforderlich, wie die Bezeichnung `CONSTRAINT` *(aus dem englischen für „Zwang")* ahnen lässt. Da das Attribut später als Primary Key festgelegt wird, kann man `NOT NULL` auch weglassen, da ein *Primary-Key-Attribut* die Bedingungen `NOT NULL` und `UNIQUE` automatisch erfüllt. Aus Verständnisgründen bleibt die Klausel vorerst dort stehen.

Hinter `NOT NULL` steht kein Komma, daher *„weiß"* das DBMS, dass Zeile 2. und 3. zusammengehören und beide Zeilen die Bedingungen für `Kundennummer` darstellen. In Zeile 3. wird somit die dritte und letzte Bedingung mit Hilfe einer `CHECK`-Klausel implementiert. Formal sieht eine allgemeine `CHECK`-Klausel folgendermaßen aus:

```
1    CHECK (<Spaltenname> [Bedingung])
```

Und mit

```
1    CHECK (<Spaltenname> IN ([Bedingung]))
```

kann ein differenzierterer Eingaberahmen formuliert werden. In beiden Fällen wird dem DBMS die Anweisung gegeben, zu überprüfen, ob die Eingabe eine bestimmte Bedingung erfüllt. Bei einer `CHECK`-Klausel muss die Bedingung immer einen *booleschen Wert* zurückgeben – also: Ja, die Eingabe erfüllt die Bedingung oder eben nicht. In diesem Fall soll die Kundennummer in jedem Fall ≥ 1000 sein. Da in Zeile 2. zusätzlich `NUMBER(7)` festgelegt wurde und dies der Primary Key ist, muss eine Zahl zwischen `1.000` und `9.999.999` als Kundennummer definiert werden. Übersetzt bedeutet die Bedingung in Zeile 3. also: *„Überprüfe, ob die Eingabe der ‚Kundennummer' größer oder gleich 1.000 ist."* Zeile 2. und 3. können auch in einer durchgehenden Zeile stehen, jedoch ist diese Art der Notation um deutliches angenehmer und übersichtlicher.

In Zeile 4. wird die Spalte für den Vor- und den Nachnamen des Kunden angelegt. Auch hier wird durch die Eingabe `Name` die Attributbezeichnung festgelegt *(vgl. Erläuterung zu Zeile 2.)*. Diese können in der Regel in einer beliebigen Sprache und mit einer beliebigen Abkürzung bezeichnet werden, jedoch müssen sie mit einem Buchstaben beginnen. Der Datentyp dieser Spalte ist `VARCHAR2(55)`, wobei die Zahl in der Klammer zusätzlich als Bedingung zu verstehen ist *(vgl. `NUMBER(7)` in Zeile 2.)*. `VARCHAR2(55)` hat zur Folge, dass der Name des Kunden aus einem *String* bestehen muss, der maximal 55 Zeichen lang sein darf und mit Apostrophen in die Kommandozeile eingegeben werden muss; in etwa so: `'[attributwert]'`. Statt `VARCHAR2` wäre im *„Oracle SQL Developer“* auch `VARCHAR` möglich. Anschließend folgt eine `NOT NULL` Bedingung, die bereits vorgestellt wurde. Der Name braucht einen Attributwert, da eine Kundenummer in Relation zu einer oder mehreren Personen steht. So kann eine Lieferung beispielsweise auch an eine Familie, eine Zahnarztpraxis oder an ein Unternehmen und Ähnlichem gehen. An den Beispielen ist zu erkennen, dass zwar der Vorname nicht in jedem Fall zwingend notwendig ist, jedoch ist ein Name an sich für die Verknüpfung an die Kundennummer von unschätzbarem Wert. Ohne diese Verknüpfung geht die Lieferung zurück zum Warenhaus.

In Zeile 5. wird die Spalte für die `Straße` angelegt, in der der Kunde lebt bzw. seine Unternehmung führt. An sich ist in dieser Zeile nichts Neues mehr zu finden. Auch hier wurden als einzige Bedingungen ein `NOT-NULL-CONSTRAINT` und ein `VARCHAR2` gewählt, der dieselbe Zeichenlänge von maximal `55` Stellen wie `Name` haben darf. Eine Lieferung muss letzten Endes irgendwo ankommen, daher ist hier eine `NOT NULL` Bedingung empfehlenswert. Eine Bestellung kann nicht an einen Kunden ohne Adresse ausgeliefert werden. Dieses Problem sollte man von vorneherein aus dem Weg gehen. Außerdem wäre es angebracht, den kompletten Straßennamen in der Tabelle aufzuführen. Der längste Straßenname in Deutschland beträgt 50 Zeichen *(ohne Straßennummer)*. Es handelt sich um die *„Bischöflich-Geistlicher-Rat-Josef-Zinnbauer-Straße“* in Dingolfing, Bayern *(gerne auch BGR-Josef-Zinnbauer-Straße abgekürzt)*.

Die Implementierung der Postleitzahl in Zeile 6. scheint im Hinblick auf den geeignetsten Datentyp für diesen Zweck am kniffligsten zu sein.

Auf Anhieb kommt einem als Datentyp vielleicht NUMBER(5) in den Sinn, der mit einer CHECK-Klausel kombiniert wird:

```
1    CHECK (PLZ >= 10.000)
```

Man erkennt schnell, dass bei dieser Bedingung Konflikte *vorprogrammiert* sind, da nur Postleitzahlen zwischen 10.000 und 99.999 möglich wären. Ob NUMBER oder INT, beides bereitet einem Sorgen. Wohin mit den Postleitzahlen von 01067 bis 01328 für Dresden? Was sollte mit 01454 für Radeberg geschehen? Einem Kunden erklären zu müssen, dass man ihm kein *„Radeberger Pils"* liefern könne, weil man die *„Radeberger Brauerei"* nicht in die Datenbank aufnehmen kann – weil eine Zahl nie mit einer 0 beginnt – klingt doch etwas obskur. Es muss also ein Datentyp her, der nichts mit einer Zahl zu tun hat. Darüber hinaus erübrigt sich auch die Wahl für die Bedingung

```
1    CHECK (PLZ IN ([PLZ1, PLZ2, PLZ-N, ...]))
```

die nur dann Sinn gemacht hätte, wenn man eine kleine und ausgewählte Gruppe von Kunden beliefern würde. In der Regel will man aber immense Kundenmengen beliefern. In Deutschland existieren derzeit 14.334 Postleitzahlen *(Quelle: http://deutschland-in-zahlen.de/)*. Es wäre also viel zu mühselige gewesen, jede einzelne von ihnen in einer CHECK-IN-Klausel unterzubringen. Und wie hätte es wohl ausgesehen, wenn man Kunden auf internationaler Ebene beliefern will?

Als einzige Möglichkeit bleibt ein String. Genaugenommen ein VARCHAR2(5), auch wenn dieser zur Folge hat, dass Buchstaben in der Eingabe möglich sind *(es sei denn, man schließt alle 26 Buchstaben mit einem DEFAULT CONSTRAINT aus)*. Somit ist die Wahrscheinlichkeit falsche Daten in die Datenbank aufzunehmen zwar höher, da man dieses Risiko aber nie komplett ausschließen kann, sollte man es gerne in Kauf nehmen. Immerhin ist die maximale Zeichenlänge auf fünf limitiert und das Fehlerrisiko ein wenig gesenkt. Für die PLZ gilt ebenfalls die NOT NULL Bedingung, da der Ortsname im Notfall allein mit der Postleitzahl

auffindbar ist. Umgekehrt ist die Eingabe eines Ortsnamen nicht zwingend erforderlich, weil man ohnehin die Postleitzahl hat.

In Zeile 7. wird die Spalte für den Wohnort des Kunden angelegt. Als Basisdatentyp und einzige Bedingung wurde ein `VARCHAR2(45)` gewählt, da sich das angenommene Kundenspektrum nur auf in Deutschland lebende Personen bezieht. Die maximale Zeichenlänge richtet sich an den längsten Ortsnamen in Deutschland. Dieser liegt in Baden-Württemberg und heißt mit 43 Zeichen *„Michelbach an der Bilz-Gschlachtenbretzigen“ (gerne auch „Michelbach an der Bilz“ abgekürzt).*

Zum Abschluss wird die `Kundennummer` in den Zeilen 8. und 9. als Primärschlüssel festgelegt. Die Implementierung sieht formal

```
1     CONSTRAINT <Constraintname>
2       PRIMARY KEY (<Spaltenname>)
```

oder alternativ auch

```
1     PRIMARY KEY (<Spaltenname>)
```

aus. Umgangssprachlich bedeutet die Anweisung: *„Lege für die Relation ‚Kunden‘ einen Primary Key (PK) an. Dieser Primary Key soll das Attribut ‚Kundennummer‘ sein.“* Zeile 8. und 9. können *(wie die `Check`-Klausel in Zeile 2. und 3.)* in einer durchgehenden Zeile stehen. Außerdem können sie auch als Zeile 4. und 5. im Befehl stehen – also direkt nach dem `Kundennummer` implementiert wurde. Diese Art der Notation ist jedoch am übersichtlichsten und lässt den Code wesentlich angenehmer lesen. Da die Anweisung in Zeile 7. die letzte ist, darf am Ende kein Komma mehr stehen. Zum Abschluss erscheint

```
table KUNDEN erstellt
```

in der Skriptausgabe.

In SQL sind Groß- und Kleinbuchstaben irrelevant (nicht Case sensitive). Auch die Leerzeichen, Absätze, Einrückungen, Tabulatoren und so weiter spielen für das jeweilige DBMS keine bedeutende Rolle.

Relation 2: Bestellungen

```
1    CREATE TABLE Bestellungen (
2      Bestellnummer NUMBER(6) NOT NULL
3        CHECK (Bestellnummer >=100000),
4      Kundennummer NUMBER(7) NOT NULL,
5      Bestelldatum DATE NOT NULL,
6      Status NUMBER(1) NOT NULL
7        CHECK (Status IN (1, 2, 3, 4, 5)),
8      CONSTRAINT Bestellungen_PK
9        PRIMARY KEY (Bestellnummer),
10     CONSTRAINT Bestellungen_FK
11       FOREIGN KEY (Kundennummer)
12       REFERENCES Kunden (Kundennummer)
13   );
```

Die Zeilen 1. bis 3. sind abgesehen von der zulässigen Stellengröße der `Bestellnummer` analog zur Kundentabelle. Für die Bestellnummer darf eine Zahl zwischen `100.000` und `999.999` eingegeben werden.

Die Implementierung der `Kundennummer` wird in Zeile 4. identisch wie bei der Kundentabelle in Zeile 2. vorgenommen. Da diese Spalte der Fremdschlüssel dieser Relation ist, wird auf die `CHECK`-Klausel verzichtet, wie sie in der Kundentabelle in Zeile 3. verwendet wurde.

In Zeile 5. wurde erstmals der Datentyp `DATE` gewählt. Dieser hat die Eigenschaft, dass man in der Spalte `Bestelldatum` nur Datumsangaben machen kann. Dabei ist die akzeptierte Zeitspanne im *„Oracle SQL Developer“* beachtlich hoch. Sie reicht vom 31.12.4712 vor Christus 00:00 Uhr, bis zum 31.12.9999 nach Christus 23:59:59 Uhr – also kaum ein Vergleich zum *„Microsoft SQL Server“* mit den Datentypen `SMALL-DATETIME` *(01.01.1900 bis 06.06.2079)* und `DATETIME` *(01.01.1753 bis 31.12.9999)*. Datumseingaben erfolgen ebenfalls mit Apostrophen.

Das letzte Attribut `Status` wird in Zeile 6. angelegt. In der ersten Praktikumsaufgabe wurde der Status mit einer Zahl zwischen 1 und 5 entworfen, die je nach Bearbeitungsstand der Bestellung durchnummeriert werden sollen. Eine Zahl wird mit genau einem Bearbeitungsschritt

assoziiert, daher wird als Attributwert genau eine Zahl zwischen 1 und 5 erwartet. Dementsprechend reicht ein `NUMBER(1)` völlig aus; `CHAR(1)` oder `INT` wären an dieser Stelle auch möglich. Damit aber keine beliebige Zahl zwischen 0 und 9 eingetragen werden kann, wird zur Fehlervermeidung eine `CHECK`-Klausel in Zeile 7. implementiert. Mit `CHEK` wird überprüft, ob der eingetragene Wert zwischen 1 und 5 liegt. Das Risiko falsche Daten einzugeben fällt also bedeutend gering aus. Äquivalent hätte man statt *„überprüfe, ob der Wert in einem bestimmten Bereich liegt"* auch sagen können: *„Füge alle Werte ein, bis auf ein paar bestimmte"*. Im Klartext: Statt einem `CHECK-IN` einen `DEFAULT CONSTRAINT`, der für `Status` folgendermaßen aussehen würde:

```
1     Status NUMBER(1) NOT NULL DEFAULT
2       (6, 7, 8, 9, 0)
```

Allerdings sollte man bei der Methode mit `DEFAULT` auf `CHAR` achten, da dieser Datentyp Buchstaben und Sonderzeichen beinhaltet.

Bei den restlichen Attributen sind keine `NULL`-Werte zulässig, da jede Spalte ihre bestimmte Aufgabe für einen reibungslosen Bestellvorgang hat. Die `Bestellnummer` *(Primary Key)* und die `Kundennummer` *(Foreign Key)* sind die *Relationswerkzeuge* zwischen `Bestellungen` und `Kunden`. Das `Bestelldatum` *(in Zeile. 5.)* zeigt, wie lange ein Bestellvorgang gedauert hat. Dieser Wert kann letztlich Informationen über die Motivation der eigenen Mitarbeiter, aber auch zu Verbesserungsmöglichkeiten für effizientere Bestellabläufe offenbaren. Der Status ist wichtig um zu sehen, wie weit eine Bestellung vorangeschritten ist. Ohne einen Status der als Zwischenstand fungiert, könnte eine Bestellung auf *„halber Strecke"* liegen bleiben und in Vergessenheit geraten.

In Zeile 8. und 9. wird die `Bestellnummer` analog wie oben in Zeile 8. und 9. die `Kundennummer` als Primary Key implementiert.

Von Zeile 10. bis 12. wird erstmals ein Foreign Key festgeschrieben. Den Code notiert man immer in der untergeordneten Relation. Der Fremdschlüssel bezieht sich auf `Kundennummer` in der Bestelltabelle und referenziert auf die `Kundennummer` in der Kundentabelle. Formal sieht der Befehl für einen Foreign Key folgendermaßen aus:

```
1     CONSTRAINT <Constraintname>
2       FOREIGN KEY (<Spaltenname>)
3       REFERENCES <überg.Tabelle> (<Spaltenname>)
```

oder alternativ auch

```
1     FOREIGN KEY <Spaltenname>
2       REFERENCES <übergeordnete Tabelle>
```

Umgangssprachlich bedeutet der Befehl: *„Lege für die Tabelle ‚Bestellungen' einen Foreign Key (FK) an. Dieser Foreign Key bezieht sich auf das Attribut ‚Kundennummer' und referenziert auf die ‚Kundennummer' in der Kundentabelle."* Mit `CONSTRAINT` wird eine Randbedingung definiert, die das DBMS prüft. Auch hier könnten die Zeilen 10. bis 12. in einer Zeile stehen.

Relation 3: Bestellpositionen

```
1  CREATE TABLE Bestellpositionen (
2    Bestellnummer NUMBER(6) NOT NULL,
3    Artikelnummer NUMBER(5) NOT NULL,
4    Menge NUMBER(4) NOT NULL,
5    CONSTRAINT Bestellpositionen_PK
6      PRIMARY KEY (Bestellnummer, Artikelnummer),
7    CONSTRAINT Bestellpositionen_FK
8      FOREIGN KEY (Bestellnummer)
9      REFERENCES Bestellungen (Bestellnummer),
10   CONSTRAINT Bestellpositionen2_FK
11     FOREIGN KEY (Artikelnummer)
12     REFERENCES Artikel (Artikelnummer)
13 );
```

Die Tabelle, die alle Bestellpositionen zusammenfasst, besteht aus insgesamt drei Spalten und einem Primary Key, der wiederum aus zwei Foreign Keys besteht. Bei allen Zeilen des `CREATE TABLE` Befehls für die

Bestellpositionen wurde bis auf einer kleinen Ausnahme wie bei den bereits geschilderten Befehlen vorgegangen.

Der Datentyp für die Bestellnummer in Zeile 2. ist nach wie vor ein NUMBER(6), wie bei Bestellungen ausführlich erklärt wurde. Auch hier ist die Eingabe eines Attributwerts durch NOT NULL zwingend erforderlich. Die Artikelnummer in Zeile 3. ist hingegen ein NUMBER(5), aber ebenfalls mit einem NOT-NULL-CONSTRAINT versehen. Näheres hierzu ist weiter unten beim CREATE TABLE Befehl für die Relation Artikel zu lesen.

In Zeile 4. wird die Spalte für die Bestellmenge der jeweils bestellten Artikel angelegt. Der Datentyp NUMBER(4) scheint am geeignetsten zu sein, da die Kunden einen Artikel entweder 1 Mal oder bis zu maximal 9.999 Mal bestellen können. Man kann annehmen, dass solch eine hohe Bestellmenge eines einzigen Artikels unrealistisch ist, aber es gibt tatsächlich Großabnehmer die unvorstellbare Mengen anfordern. Dieses Beispiel soll sich aber lediglich auf einen Online-Shop oder Ähnlichem beziehen. Die Eingabe für die Bestellmenge ist ebenfalls zwingend notwendig, da ihre Summe sowohl für den Bestand, als auch das betriebliche Rechnungswesen *(Ausgaben, Einnahmen, Jahresabschluss, Buchhaltung, Inventur, Inventar etc.)* äußerst bedeutsam sind. Abgesehen davon, kann man einem Kunden nicht zwei Artikel schicken, wenn er für drei bezahlt hat. Würden man ihm drei Artikel schicken, obwohl er zwei bezahlt hat, würde sich der Kunde zwar freuen, allerdings käme dieser Fehler auf Dauer dem Unternehmen teuer zu Buche und das nur, weil man die Bestellmenge nicht notiert hat.

Die Erläuterungen zu den beiden Befehlen für die Foreign Keys sind identisch mit den oben geschilderten Foreign Key für Kundennummer in der Bestelltabelle. Da an dieser Stelle jedoch zwei Foreign Keys stehen, muss einer der beiden nummeriert sein. So steht

```
1    CONSTRAINT Bestellpostionen_FK
```

für die Bestellnummer und

```
1    CONSTRAINT Bestellpositionen2_FK
```

für die Artikelnummer. Würde man nicht so vorgehen, dann würde der *„Oracle SQL Developer“* eine Fehlermeldung ausgeben, weil bei zweifacher Ausführung bereits ein CONSTRAINT Bestellpositionen_FK existieren würde *(ein vereinfachter Befehl steht auf Seite 123)*.

Würden die beiden Foreign Keys auf ein und dieselbe Tabelle referenzieren, dann hätte man die Befehle in einen zusammenfassen können. Dies hätte in solch einem Fall so ausgesehen:

```
1    CONSTRAINT <Constraintname>
2      FOREIGN KEY (<Spaltenname1>,
3                   ..., <SpaltennameN>)
4      REFERENCES <übergeordnete Tabelle>
5                   (<Spaltenname1>,
6                   ..., <SpaltennameN>)
```

So etwas in der Art ist in Zeile 5. und 6. für den zusammengesetzten Primary Key der Fall, womit die n:m-Beziehung realisiert wird:

```
1    CONSTRAINT Bestellpositionen_PK
2     PRIMARY KEY(Bestellnummer, Artikelnummer)
```

Relation 4: Artikel

```
1    CREATE TABLE Artikel (
2      Artikelnummer NUMBER(5) NOT NULL
3        CHECK (Artikelnummer >=10000),
4      Artikelname VARCHAR2(50) NOT NULL,
5      Produktgruppe VARCHAR2(30) NOT NULL,
6      Verkaufspreis NUMBER(6,2) NOT NULL,
7      CONSTRAINT Artikel_PK
8        PRIMARY KEY (Artikelnummer)
9    );
```

Der `CREATE TABLE` Befehl für die Relation `Artikel` beherbergt, bis auf die Neuerung in Zeile 6. keine unbekannten Inhalte. Die `Artikelnummer` *(Zeile 2.)* wurde als `NUMBER(5)` mit einem `NOT-NULL-CONSTRAINT` und mit einer `CHECK`-Klausel *(Zeile 3.)* implementiert. So sind Artikelnummern zwischen `10.000` und `99.999` erforderlich.

`Artikelname` und `Produktgruppe` haben jeweils den Datentyp `VARCHAR2`, wobei die Zeichenlänge der Namen auf maximal `50` und die der Produktgruppe auf maximal `30` limitiert ist. Darüber hinaus ist die Eingabe eines Attributwertes in beiden Fällen erforderlich.

In Zeile 7. und 8. wird `Artikelnummer` als PK festgeschrieben.

Nun zu Zeile 6.: Für den Verkaufspreis wurde der Datentyp `NUMBER` mit einer speziellen Bedingung gewählt, die mit zwei Zahlen, einem Komma und einer Klammer notiert wird `(6,2)`. Diese Bedingung hat zur Folge, dass man mittels Apostrophe eine beliebige Dezimalzahl zwischen `1,00` und `999.999,99` als Verkaufspreis eingeben kann. Es handelt sich also um eine Dezimalzahl, bei der die 6 vor dem Komma die Stellen der ganzen Euros entspricht und die 2 die Stellen der Cents.

Hinweis zu den Schlüsseln

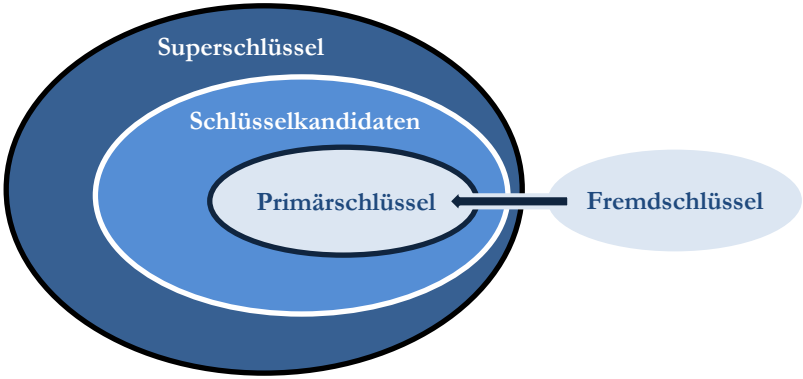
Primärschlüssel: Der *Primary Key (PK)* ist ein Attribut *(Spalte),* welches aus den Schlüsselkandidaten ausgewählt wurde, um die Tupel *(Zeilen)* einer Relation *(Tabelle)* eindeutig zu identifizieren.

Ein Primary Key ist zwar keine Pflicht, aber man sollte darauf achten, in jeder Relation mindestens ein Attribut als Primary Key ausgewählt zu haben.

Fremdschlüssel: Der *Foreign Key (FK)* ist ein Attribut *(oder eine Attributkombination)* einer Relation, welches auf einen Primärschlüssel einer anderen oder der gleichen Relation verweist. Ein Foreign Key kann auch gleichzeitig ein Primary Key sein.

Schlüsselkandidat: Der *Candidate Key* ist eine *„minimale"* Menge von Attributen, die die Tupel einer Relation eindeutig identifizieren. Allgemein formuliert: Für die Relation $R(A)$ mit A als die Menge aller Attribute und $a \subseteq A$ gilt $a \rightarrow A$.

Der *Superschlüssel* ist die Obermenge aller Schlüsselkandidaten und diese sind wiederum die Obermenge vom Primärschlüssel, der aus dieser Menge auserwählt wird. Die explizite Einordnung der Schlüsselformen soll folgende Grafik verdeutlichen:



Hinweise zu den Datentypen

INT	Ganze Zahlen zwischen -2^{31} bis $2^{31}-1$ Weniger Speicherplatz als NUMBER Werte benötigen für jedes verwendete Zeichen Speicherplatz
NUMBER(p, s)	Gesamtzahl zwischen $p = 1$ bis 38 Stellen oder Dezimalzahl $s = -84$ bis127 Vor- bzw. Nachkommastellen Werte benötigen für jedes definierte Zeichen Speicherplatz
CHAR	Zeichenkette fester Länge zwischen 1 bis 2.000 Werte benötigen für jedes definierte Zeichen Speicherplatz
VARCHAR **VARCHAR2**	Zeichenkette mit variabler Länge zwischen 1 bis 4.000 Weniger Speicherplatz als CHAR Werte benötigen für jedes verwendete Zeichen Speicherplatz
DATE	Datums- und Uhrzeitangaben sekundengenau Datum zwischen 01.01.4712 v. Chr. bis 31.12.9999 n. Chr.

Aufgabe 2.2. Der `INSERT INTO` Befehl

Speichern Sie mit dem `INSERT INTO` Befehl Tupel in den Tabellen ab. Lassen Sie sich die Tupel mit `SELECT * FROM` ausgeben.

Der `INSERT INTO` Befehl sieht allgemein folgendermaßen aus:

```
1    INSERT INTO <Tabellenname> VALUES (...);
```

Der Befehl bedeutet umgangssprachlich: *„Füge in die Tabelle x folgende Werte ein."* Textwerte mit `VARCHAR` und Datumsangaben müssen mit Apostrophe in die Klammer eingegeben werden, andernfalls wird kein einziger Wert hinzugefügt. Zahlen erkennt das DBMS automatisch. Die Attributwerte müssen hinsichtlich ihrer Anzahl und ihrer Reihenfolge mit den Attributen der Relation übereinstimmen. Sie werden alle in der Klammer eingegeben und mit Hilfe von Kommas getrennt. Für zulässige `NULL`-Werte muss explizit `NULL` als Attributwert geschrieben werden.

Im Folgenden werden alle `INSERT INTO` Befehle nacheinander aufgelistet und anschließend die Tabellen aus dem DBMS gezeigt. Die Auflistung schließt die Tabellen aus der Aufgabe 2.4. mit ein.

Relation 1: Kunden

```
INSERT INTO KUNDEN VALUES (11111, 'Rick Grimes',
'Walking-Road 23', '01934', 'Dead Ville');
INSERT INTO KUNDEN VALUES (11112, 'Carl Grimes',
'Walking-Road 23', '01934', 'Dead Ville');
INSERT INTO KUNDEN VALUES (11333, 'Hershel Greene',
'Farm im Grünen 101', '22115', 'Farmers Hood');
INSERT INTO KUNDEN VALUES (141424, 'Daryl Dixon',
'Merler-Landstraße 20', '22111', 'Philadelphia');
INSERT INTO KUNDEN VALUES (141426, 'Carol Paletier',
'Klosterweg 13', '20224', 'Atlanta');
INSERT INTO KUNDEN VALUES (152341, 'Abraham Ford',
'Karottenallee 4', '30821', 'Washington D.C.');
```

Namen angelehnt an „The Walking Dead"

```
INSERT INTO KUNDEN VALUES (2233, 'Walter White',
'Erlenmeyer-Straße 12a', '42351', 'Albuquerque');
INSERT INTO KUNDEN VALUES (213454, 'Skyler White',
'Erlenmeyer-Straße 12a', '42351', 'Albuquerque');
INSERT INTO KUNDEN VALUES (223344, 'Hank Shrader',
'An der Polizeiwache 80', '39874', NULL);
INSERT INTO KUNDEN VALUES (21456, 'Jesse Pinkman',
'Bei-der-Schulbibliothek 12', '39874', NULL);
INSERT INTO KUNDEN VALUES (2948517, 'Gustavo Fring',
'Pollo Hermanos 15', '38911', 'New Mexico');
INSERT INTO KUNDEN VALUES (2837472, 'Saul Goodman',
'Bei-den-Anwälten 34', '42351', 'Albuquerque');
```

Namen angelehnt an „Breaking Bad“

```
INSERT INTO KUNDEN VALUES (324567, 'Sheldon Cooper',
'Big-Banger-Kreisel 42b', '77345', 'Pasadena');
INSERT INTO KUNDEN VALUES (3256788, 'Leonard Hofstadt-
er', 'Big-Banger-Kreisel 42b', '77345', 'Pasadena');
INSERT INTO KUNDEN VALUES (385454, 'Rajesh Koothrap-
pali', 'Universitätsallee 69', '78090', NULL);
INSERT INTO KUNDEN VALUES (3987634, 'Howard Wolowitz',
'Neil-Armstrong-Platz 8', '76548', 'Altadena');
INSERT INTO KUNDEN VALUES (37643, 'Amy Farrah Fowler',
'Charles-Darwin-Weg 112', '77355', 'Pasadena');
INSERT INTO KUNDEN VALUES (309876, 'Bernadette Rosten-
kowski', 'Lamarck-Straße 2', '78090', NULL);
```

Namen angelehnt an „The Big Bang Theory“

Alle 18 Adressen sind frei erfunden!

```
SELECT * FROM KUNDEN
```

	KUNDENNUMMER	NAME	STRAßE	PLZ	ORT
1	11111	Rick Grimes	Walking-Road 23	01934	Dead Ville
2	11112	Carl Grimes	Walking-Road 23	01934	Dead Ville
3	11333	Hershel Greene	Farm im Grünen 101	22115	Farmers Hood
4	141424	Daryl Dixon	Merler-Landstraße 20	22111	Philadelphia
5	141426	Carol Paletier	Klosterweg 13	20224	Atlanta
6	152341	Abraham Ford	Karottenallee 4	30821	Washington D.C.
7	2233	Walter White	Erlenmeyer-Straße 12a	42351	Albuquerque
8	213454	Skyler White	Erlenmeyer-Straße 12a	42351	Albuquerque
9	223344	Hank Shrader	An der Polizeiwache 80	39874	(null)
10	21456	Jesse Pinkman	Bei-der-Schulbibliothek 12	39874	(null)
11	2948517	Gustavo Fring	Pollo Hermanos 15	38911	New Mexico
12	2837472	Saul Goodman	Bei-den-Anwälten 34	42351	Albuquerque
13	324567	Sheldon Cooper	Big-Banger-Kreisel 42b	77345	Pasadena
14	3256788	Leonard Hofstadter	Big-Banger-Kreisel 42b	77345	Pasadena
15	385454	Rajesh Koothrappali	Universitätsallee 69	78090	(null)
16	3987634	Howard Wolowitz	Neil-Armstrong-Platz 8	76548	Altadena
17	37643	Amy Farrah Fowler	Charles-Darwin-Weg 112	77355	Pasadena
18	309876	Bernadette Rostenkowski	Lamarck-Straße 2	78090	(null)

Relation 2: Bestellungen

```
INSERT INTO BESTELLUNGEN VALUES
(121226, 152341, '25.04.2015', 3);
INSERT INTO BESTELLUNGEN VALUES
(121227, 3987634, '27.04.2015', 1);
INSERT INTO BESTELLUNGEN VALUES
(121228, 223344, '26.04.2015', 2);
INSERT INTO BESTELLUNGEN VALUES
(121229, 2948517, '26.04.2015', 2);
INSERT INTO BESTELLUNGEN VALUES
(121230, 2948517, '25.04.2015', 3);
INSERT INTO BESTELLUNGEN VALUES
(121231, 2948517, '26.04.2015', 2);
INSERT INTO BESTELLUNGEN VALUES
(121232, 324567, '26.04.2015', 2);
INSERT INTO BESTELLUNGEN VALUES
(121233, 324567, '25.04.2015', 3);
INSERT INTO BESTELLUNGEN VALUES
(121234, 11111, '26.04.2015', 2);
```

```
SELECT * FROM BESTELLUNGEN
```

	BESTELLNUMMER	KUNDENNUMMER	BESTELLDATUM	STATUS
1	121234	11111	26.04.15	2
2	121226	152341	25.04.10	3
3	121227	3987634	25.04.10	1
4	121228	223344	25.04.10	2
5	121229	2948517	26.04.15	2
6	121230	2948517	25.04.15	3
7	121231	2948517	26.04.15	2
8	121232	324567	26.04.15	2
9	121233	324567	25.04.15	3

Relation 3: Bestellpositionen

```
INSERT INTO BESTELLPOSITIONEN VALUES (121226, 12456, 2);
INSERT INTO BESTELLPOSITIONEN VALUES (121226, 12459, 1);
INSERT INTO BESTELLPOSITIONEN VALUES (121226, 12451, 3);
INSERT INTO BESTELLPOSITIONEN VALUES (121227, 12451, 4);
INSERT INTO BESTELLPOSITIONEN VALUES (121228, 12456, 2);
INSERT INTO BESTELLPOSITIONEN VALUES (121229, 12459, 1);
INSERT INTO BESTELLPOSITIONEN VALUES (121230, 12459, 3);
INSERT INTO BESTELLPOSITIONEN VALUES (121230, 12456, 3);
INSERT INTO BESTELLPOSITIONEN VALUES (121230, 12459, 1);
INSERT INTO BESTELLPOSITIONEN VALUES (121230, 12451, 1);
INSERT INTO BESTELLPOSITIONEN VALUES (121230, 12457, 1);
INSERT INTO BESTELLPOSITIONEN VALUES (121230, 12459, 2);
INSERT INTO BESTELLPOSITIONEN VALUES (121231, 13675, 2);
INSERT INTO BESTELLPOSITIONEN VALUES (121231, 13645, 4);
INSERT INTO BESTELLPOSITIONEN VALUES (121231, 13645, 1);
INSERT INTO BESTELLPOSITIONEN VALUES (121233, 12451, 2);
INSERT INTO BESTELLPOSITIONEN VALUES (121234, 13645, 3);
INSERT INTO BESTELLPOSITIONEN VALUES (121234, 12456, 1);
```

```
SELECT * FROM BESTELLPOSITIONEN
```

	BESTELLNUMMER	ARTIKELNUMMER	MENGE
1	121226	12456	2
2	121226	12459	1
3	121226	12451	3
4	121227	12451	4
5	121228	12456	2
6	121229	12459	1
7	121230	12459	3
8	121230	12456	3
9	121230	13645	1
10	121230	12451	1
11	121230	12457	1
12	121230	13675	2
13	121231	13675	2
14	121231	12456	4
15	121231	13645	1
16	121233	12451	2
17	121234	13645	3
18	121234	12456	1

Relation 4: Artikel

```
INSERT INTO ARTIKEL VALUES (14460, 'James Audrey Tik-
Tok', 'Herrenuhren', '149,00');
INSERT INTO ARTIKEL VALUES (12440, 'James Audrey Busi-
ness Clock', 'Herrenuhren', '349,00');
INSERT INTO ARTIKEL VALUES (12450, 'James Audrey Man
Fitness', 'Herrenuhren', '149,00');
INSERT INTO ARTIKEL VALUES (13670, 'James Audrey Lady
Fitness', 'Frauenuhren', '149,00');
INSERT INTO ARTIKEL VALUES (14456, 'James Audrey Mrs
Audreys Chrono One', 'Frauenuhren', '179,00');
INSERT INTO ARTIKEL VALUES (12456,
'James Audrey Chrono One', 'Herrenuhren', '249,00');
INSERT INTO ARTIKEL VALUES (12459,
'James Audrey Big Data', 'Herrenuhren', '149,00');
INSERT INTO ARTIKEL VALUES (12457,
'James Audreys Window', 'Sonnenbrillen', '99,99');
INSERT INTO ARTIKEL VALUES (13675, 'James Audrey Mrs
Audreys Window', 'Sonnenbrillen', '89,99');
INSERT INTO ARTIKEL VALUES (13645,
'James Audrey Big Data', 'Frauenuhren', '149,00');
INSERT INTO ARTIKEL VALUES (12451, 'James Audrey Cameron
Bends', 'Herrenarmbänder', '49,99');
```

```
SELECT * FROM ARTIKEL
```

	ARTIKELNUMMER	ARTIKELNAME	PRODUKTGRUPPE	VERKAUFSPREIS
1	14460	James Audrey TikTok	Herrenuhren	149
2	12440	James Audrey Business Clock	Herrenuhren	349
3	12450	James Audrey Man Fitness	Herrenuhren	149
4	13670	James Audrey Lady Fitness	Frauenuhren	149
5	14456	James Audrey Mrs Audreys Chrono One	Frauenuhren	179
6	12456	James Audrey Chrono One	Herrenuhren	249
7	12459	James Audrey Big Data	Herrenuhren	149
8	12457	James Audreys Window	Sonnenbrillen	99,99
9	13675	James Audrey Mrs Audreys Window	Sonnenbrillen	89,99
10	13645	James Audrey Big Data Mrs	Frauenuhren	149
11	12451	James Audrey Cameron Bends	Herrenarmbänder	49,99

Relation 5: Konditionen

```
INSERT INTO KONDITIONEN VALUES
(3165689, 12440, '139,60');
INSERT INTO KONDITIONEN VALUES
(3165689, 12450, '59,60');
INSERT INTO KONDITIONEN VALUES
(3128889, 12451, '22,00');
INSERT INTO KONDITIONEN VALUES
(3123789, 12451, '19,99');
INSERT INTO KONDITIONEN VALUES
(3165689, 12451, '25,45');
INSERT INTO KONDITIONEN VALUES
(2456456, 12456, '99,60');
INSERT INTO KONDITIONEN VALUES
(1789123, 12457, '39,99');
INSERT INTO KONDITIONEN VALUES
(2456456, 12459, '59,60');
INSERT INTO KONDITIONEN VALUES
(2456456, 13645, '59,60');
INSERT INTO KONDITIONEN VALUES
(3123789, 13670, '59,60');
INSERT INTO KONDITIONEN VALUES
(1789123, 13675, '35,99');
INSERT INTO KONDITIONEN VALUES
(2456456, 14456, '71,60');
INSERT INTO KONDITIONEN VALUES
(2456456, 14460, '59,60');
```

Relation 6: Lieferanten

```
INSERT INTO LIEFERANTEN VALUES (1789123, 'Günes-Gözlügü
AG', 'Sonnenallee 112', '77345', 'Pasadena');
INSERT INTO LIEFERANTEN VALUES (2456456, 'Clock-Work
GmbH', 'Additionsstraße 16', '42351', 'Albuquerque');
INSERT INTO LIEFERANTEN VALUES (3123789, 'Schmuck-und-
Accessoires GbR', 'Ticari Weg 24a', '01934', NULL);
INSERT INTO LIEFERANTEN VALUES (3165689, 'Schmuck-und-
Herrenartikel', 'Ticari Weg 24b', '01934', NULL);
INSERT INTO LIEFERANTEN VALUES (3128889, 'Your Brends
Herrenarmbänder','Schlangenhügel 10','20224','Atlanta');
```

SELECT * FROM KONDITIONEN

	LIEFERANTENNUMMER	ARTIKELNUMMER	EINKAUFSPREIS
1	3165689	12440	139,6
2	3165689	12450	59,6
3	3128889	12451	22
4	3123789	12451	19,99
5	3165689	12451	25,45
6	2456456	12456	99,6
7	1789123	12457	39,99
8	2456456	12459	59,6
9	2456456	13645	59,6
10	3123789	13670	59,6
11	1789123	13675	35,99
12	2456456	14456	71,6
13	2456456	14460	59,6

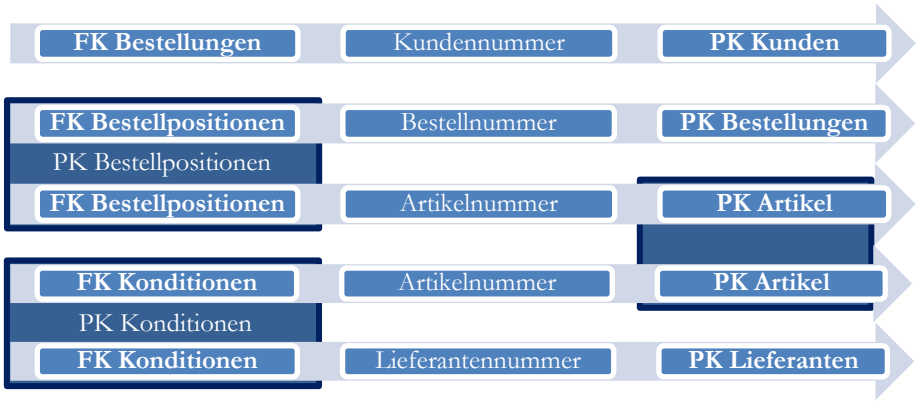
SELECT * FROM LIEFERANTEN

	LIEFERANTENNUMMER	NAME	STRAßE	PLZ	ORT
1	1789123	Günes-Gözlügü AG	Sonnenallee 112	77345	Pasadena
2	2456456	Clock-Work GmbH	Additionsstraße 16	42351	Albuquerque
3	3123789	Schmuck-und-Accessoires GbR	Ticari Weg 24a	01934	(null)
4	3165689	Schmuck-und-Herrenartikel	Ticari Weg 24b	01934	(null)
5	3128889	Your Brends Herrenarmbänder	Schlangenhügel 10	20224	Atlanta

Beim Hinzufügen der Datensätze wurde insbesondere darauf geachtet, dass eine ausreichende Anzahl an Tupel für die späteren *Queries* aus den Praktikumsaufgaben 3 und 4 vorhanden ist. Darüber hinaus wurde darauf geachtet, dass sich die Datensätze an folgende Vorgaben halten.

Für jeden Foreign Key soll es in der „Mastertabelle" mindestens einen Schlüssel ohne Eintrag in der abhängigen Tabelle geben.

Es sind insgesamt sechs verschiedene Tabellen, mit sechs unterschiedlichen Primary Keys *(PK)* und fünf Foreign Keys *(FK)*.



Für den ersten Testfall wurde in jeder Foreign Key Spalte mindestens ein Primary Key Attributwert nicht eingetragen, obwohl dieser in der abhängigen Tabelle existiert. Es befinden sich achtzehn Kunden in der Kundentabelle, von denen zwölf keine Bestellung aufgegeben haben. Somit kommen die Kundennummern *2233, 11112, 11333, 21456, 37643, 141424, 141426, 213454, 309876, 385454, 2837472, 3256788* zwar als Primary Key in der Kundentabelle vor, aber nicht als Foreign Key in der Bestelltabelle. Dasselbe gilt für die Bestellnummer *121232* und die Artikelnummern *14460, 12440, 12450, 13670* und *14456,* die nicht in der Bestellpositionen-Tabelle aufgeführt werden. In der Konditionstabelle sind jedoch alle Lieferanten- und Artikelnummern aus den abhängigen Tabellen zu finden.

Würde man einen Attributwert *(z.B. Kundennummer)* als Foreign Key in der Bestelltabelle eintragen, der nicht als Primary Key in der Kundentabelle vorkommt, würde dies zu einer Fehlermeldung in der Ausgabe führen. Der Grund ist simpel: Die Aufnahme von falschen Daten in der Datenbank soll verhindert bzw. minimiert werden *(Datenintegrität).*

Jeder Attributwert aus der Menge der PK Attributwerte bildet auf mindestens einen Attributwert aus der Menge der FK Attributwerte ab. Jedem FK Attributwert kann aber nur genau einem PK Attributwert zugeordnet werden. Dieser Zusammenhang erinnert an eine *surjektive Abbildung,* sofern alle Attributwerte au der PK Menge betroffen sind.

Es muss mindestens einen Kunden mit mehreren Bestellungen und eine Bestellung mit mehreren Bestellpositionen geben.

Insgesamt haben zwei fiktive Kunden – Gustavo Fring *(2948517)* und Sheldon Cooper *(324567)* – mehrere Bestellungen aufgegeben. Dementsprechend werden die drei Bestellnummern *121229, 121230* und *121231* Gustavo Fring und die beiden Bestellnummern *121232* und *121233* Sheldon Cooper zugeordnet.

Vier Bestellungen haben wiederum mehrere Bestellpositionen. Die Bestellnummern *121226* und *121231* umfassen **drei**, *121234* **zwei** und die *121230* sogar **sechs** einzelne Bestellpositionen.

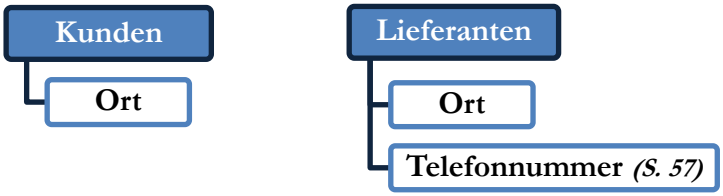
Mindestens ein Artikel soll von mehreren Kunden bestellt werden.

Die Kunden haben insgesamt sechs von elf zur Auswahl stehenden Artikel aus dem fiktiven Sortiment bestellt. Die Artikel *12457* und *13675* wurden jeweils einmal in Auftrag gegeben. Die Artikel *13645* und *12459* wurden jeweils dreimal von zwei unterschiedlichen Kunden mit unterschiedlichen Bestellnummern bestellt. Der Artikel *12451* wurde viermal von vier Kunden mit vier verschiedenen Bestellnummern gekauft und als absoluter *Verkaufsrenner* erwies sich der Artikel *12456*, der fünfmal in den Versand ging, jedoch wollten nur vier verschiedene Kunden die 249,00€ teure Herrenuhr *„James Audrey – Chrono One"* haben.

Für jedes Attribut, bei dem NULL-Werte zulässig sind, muss es auch einen Eintrag mit NULL geben.

Da die Attributwerte bezüglich der Zuordnungsfähigkeit innerhalb einer Relation besonders wichtig sind, kommen zulässige NULL-Werte ausschließlich bei Attributen vor, in denen die Möglichkeit von zulässigen

NULL-Werten sinnvoll ist. Dies war bei insgesamt zwei von sechs Relationen und drei Attributen der Fall:



Die Wohnorte der Kunden *223344*, *21456*, *385454* und *309876* wurden mit NULL-Werten gespeichert. Auch die Lieferanten *3123789* und *3165689* haben im INSERT INTO Befehl statt eines Ortsnamen den Wert NULL bekommen. Das Attribut Telefonnummer wird weiter unten in der Aufgabe 3.1. auf Seite 57 erläutert.

Für Attribute mit der CHECK-Klausel *muss es genügend Einträge geben. Ein Wert sollte nicht vorkommen, einer mehrfach.*

Von einer erwähnenswerten CHECK-Klausel – genauer gesagt von einer CHECK (<...> IN ([...])) Klausel – wurde bei allen CREATE TABLE Befehlen nur einmal Gebrauch gemacht. Dieser ist in Zeile 7. bei Bestellungen zu finden und limitiert die Eingabe für den Bestellstatus, der nur zwischen der 1 und 5 liegen soll.

Der Bestellstatus *1* kommt nur einmal vor, der Status *3* hingegen dreimal. Am meisten ist der Bestellstatus *2* zu finden – nämlich fünfmal. Sowohl Status *4*, als auch *5*, kommen in der Tabelle nicht vor.

Wenn man versucht die Werte *6*, *7*, *8*, *9* und *0* einzutragen, wird das komplette Tupel den Wüschen entsprechend nicht hinzugefügt und der INSERT INTO Befehl mit einer Fehlermeldung quittiert.

Aufgabe 2.3. Glossar

Erstellen Sie ein Glossar, das für jede Ihrer Datenbanktabellen und für jede Ihrer Attribute eine kurze Beschreibung des Inhalts enthält.

Im Folgenden wird das Glossar aufgeführt *(einschließlich der Aufgabe 2.4.)*.

RELATION	ATTRIBUT	BESCHREIBUNG	KEY
Kunden		**Daten der Kunden**	
	Kundennummer	Kundennummer	PK
	Name	Voller Name des Kunden	
	Straße	Straße des Kunden	
	PLZ	Postleitzahl des Kunden	
	Ort	Wohnort des Kunden	
Bestellungen		**Daten der Bestellungen**	
	Bestellnummer	Bestellnummer	PK
	Kundennummer	Kunde mit einer Bestellung	FK
	Bestelldatum	Datum des Bestellauftrags	
	Status	Bearbeitungsstatus	
Bestellpositionen		**Daten über eine Bestellung**	
	Bestellnummer Artikelnummer	Bestellposition einer Bestellung Artikel einer Bestellung	P-FK
	Menge	Bestellmenge einer Bestellung	
Artikel		**Daten über die Artikel**	
	Artikelnummer	Artikelnummer	PK
	Artikelname	Bezeichnung des Artikels	
	Produktgruppe	Produktgruppe des Artikels	
	Verkaufspreis	Verkaufspreis des Artikels	
Konditionen		**Daten über die Konditionen**	
	LieferantenNr Artikelnummer	Lieferantennummer Artikel des Lieferanten	P-FK
	Einkaufspreis	Bezugspreis des Artikels	
Lieferanten		**Daten der Lieferanten**	
	LieferantenNr	Lieferantennummer	PK
	Name	Name des Lieferanten	
	Straße	Straße des Lieferanten	
	PLZ	Postleitzahl des Lieferanten	
	Ort	Wohnort des Lieferanten	

Aufgabe 2.4. Noch mehr `CREATE TABLE`

Die Artikel sollen jetzt von Lieferanten bezogen werden können. Dazu soll eine Tabelle `Konditionen` eingerichtet werden, aus der ersichtlich ist, welcher Artikel, von welchem Lieferanten, zu welchem Preis bezogen werden kann.

Erstellen Sie mit dem `CREATE TABLE` Befehl die beiden Relationen `Konditionen` und `Lieferanten`.

Relation 5: Konditionen

```
1  CREATE TABLE Konditionen (
2    Lieferantennummer NUMBER(7) NOT NULL,
3    Artikelnummer NUMBER(5) NOT NULL,
4    Einkaufspreis NUMBER(6,2) NOT NULL,
5    CONSTRAINT Konditionen_PK
6      PRIMARY KEY (Lieferantennummer,Artikelnummer),
7    CONSTRAINT Konditionen_FK
8      FOREIGN KEY (Lieferantennummer)
9      REFERENCES Lieferanten (Lieferantennummer),
10   CONSTRAINT Konditionen2_FK
11     FOREIGN KEY (Artikelnummer)
12     REFERENCES Artikel (Artikelnummer)
13 );
```

Die Relation `Konditionen` fasst in drei Spalten zusammen, **welcher** Lieferant *(`Lieferantennummer`)*, **welchen** Artikel *(`Artikelnummer`)* zu **welchem** Preis *(`Einkaufspreis`)* liefert.

Ähnlich wie bei `Bestellpositionen`, sind auch hier drei Attribute und zwei Foreign Keys zu finden. Dementsprechend kann man die beiden `CREAT TABLE` Befehle analog betrachten: Die `Liefernummer`, die `Artikelnummer` und der `Einkaufspreis` sind alle vom Datentyp `NUMBER`, mit dem einzigen unterschied, dass die maximale

Zeichenlängen variiert. Der Einkaufspreis verhält sich mit der speziellen Bedingung ähnlich wie der Verkaufspreis. Der Hintergrund und der Eingaberahmen für die Attributwerte sind identisch mit der Implementierung für den Verkaufspreis *(vgl. CREATE TABLE Artikel, Zeile 6., Seite 35)*. Für alle drei Spalten sind keine NULL-Werte zulässig. In Zeile 5. und 6. wird analog zur Relation Bestellpositionen der zusammengesetzte Primary Key und von Zeile 7.-12. für die Attribute Lieferantennummer und Artikelnummer die Foreign Keys erstellt. In Zeile 10. muss der CONSTRAINT ebenfalls nummeriert werden *(vgl. CREATE TABLE Bestellpositionen, Zeile 10, Seite 33)*, da sonst eine Fehlermeldung aufgrund der redundanten Bezeichnung ausgegeben würde.

Relation 6: Lieferanten

```
1    CREATE TABLE Lieferanten (
2      Lieferantennummer NUMBER(7) NOT NULL,
3      Name VARCHAR2(30) NOT NULL,
4      Straße VARCHAR2(55) NOT NULL,
5      PLZ VARCHAR2(5) NOT NULL,
6      Ort VARCHAR2(45),
7      CONSTRAINT Lieferanten_PK
8        PRIMARY KEY (Lieferantennummer)
9    );
```

Die Erläuterungen sind mittlerweile an einem Punkt angelangt, an dem auf irgendeiner Weise auf jede einzelne Zeile in einem anderen CREAT TABLE Befehl bereits eingegangen wurde. Deshalb bezieht sich der folgende Abschnitt nur noch auf die Lieferantenspezifischen Punkte.

Das Sortiment bietet elf Artikel aus vier großen Produktgruppen, die von fünf Lieferanten bezogen werden. Dennoch hat man sich gegen eine CHECK-Klausel für Lieferantennummer entschieden, weil man niemals ausschließen kann, dass ein Lieferant kurzzeitig oder komplett wechseln muss. Aus dieser Sicht wäre ein CHECK (Lieferanten-

nummer IN ([…])) oder ein CHECK (Ort IN ([…])) kontraproduktiv. Da sich das Verhältnis zu einem Lieferanten schnell ändern kann, wurde nicht einmal eine CHECK (Lieferantennummer >=1000000) implementiert. Man lässt also nach unten hin alles offen und geht möglichen Problemen von Vorneherein aus dem Weg, auch wenn dies ein erhöhtes Fehlerrisiko im INSERT-Befehl nach sich zieht. Man kann dadurch aber erkennen, dass der Datenbankentwurf auch vorausschauendes Denken erfordert.

Die Lieferantennummer ist der Primary Key *(Zeile 7. und 8.)* und die Befehle zur Erzeugung der Attribute Name *(Zeile 3.),* Straße *(Zeile 4.),* PLZ *(Zeile 5.)* und Ort *(Zeile 6.)* sind analog zu den gleichnamigen Attributen aus der Kundentabelle.

Hinweis zur Erzeugung von Relationen

Anstelle der CREATE TABLE Befehle, kann man auch oben-links im *Oracle SQL Developer* mit einem rechten Mausklick auf *„Tabellen"* klicken und im Kontextmenü *„Neue Tabelle…"* auswählen. Dort wird ein separates Fenster mit einer Tabelle geöffnet, in der man die gewünschten Attributbezeichnungen und ihre Bedingungen eintragen kann. Von links nach rechts wird in den einzelnen Spalten folgendes festlegen:

Ein Attribut als Primary Key auswählen *(die Zeile wird daraufhin mit einem Schlüssel und einem Sternchen markiert),* einen beliebigen Attributnamen festlegen, seinen Datentyp über einen Reiter auswählen und anschließend eine Zahl für die maximale Zeichenlänge eintragen. In der fünften Spalte kann man bequem mit einem Häkchen festlegen, ob NULL-Werte zulässig sein sollen oder nicht. Mit einem + bzw. ×, die sich rechts in der Ecke befinden, kann man Attribute beliebig hinzufügen oder entfernen.

Aufgabe 2.5. Datenbankanalyse

Es gibt jetzt also eine Tabelle Kunden und eine Tabelle Lieferanten. Gibt es evtl. eine andere Möglichkeit? Beurteilen Sie diese Lösungen.

An sich gibt es an dieser Lösung nicht allzu viel zu bemängeln. Zwei Tabellen die zwischen `Kunden` und `Lieferanten` unterscheiden, kann als hervorragende Herangehensweisen verstanden werden und verschafft eine Übersicht im höchsten Maß. Dadurch ist gegeben, dass man in kürzester Zeit alle notwendigen Daten über die Kunden, aber auch die Lieferanten, mit Hilfe von Queries abrufen oder einsehen kann. Auf der einen Seite stehen die Einnahmen durch Kunden und auf der anderen die Ausgaben durch Lieferanten gegenüber.

Dennoch beinhalten beide Relationen ähnliche Attribute – nämlich `Name`, `Straße`, `PLZ` und `Ort`. Deshalb kann eine Zusammenführung mit `Geschäftspartnernummer` als Primary Key sinnvoll sein. Die vorangeführten Vorteile müssen deshalb aber nicht verloren gehen. Man erstellt drei neue Relationen: `Geschäftspartner`, `Kundendaten` und `Lieferantendaten` und zwei *VIEW's*: `Kunden` und `Lieferanten`. Die Views beinhalten dann die kongruenten Attribute, die vorher die Relationen `Kunden` und `Lieferanten` beinhalteten. Alles Notwendige zu den Views wird in der Praktikumsaufgabe 5 gezeigt.

Eine weitere Verbesserungsmöglichkeit wäre die Zusammenführung der beiden Relationen `Artikel` und `Konditionen`. Anstatt die beiden Tabellen getrennt zu führen, kann man alle drei Attribute aus der Konditionstabelle in die Artikeltabelle mit einspeisen; bzw. zwei Attribute, da in diesem Fall das Attribut `Artikelnummer` aus der Konditionstabelle wegfallen würde. Außerdem hätte dies den Vorteil, dass man alle artikelbezogenen Daten noch effizienter einsehen kann. Auch das Fehlerrisiko würde geringer ausfallen, da die Eingabe Artikel für Artikel erfolgen würde und nicht über zwei Tabellen hinweg. Für die weitere Bearbeitung der Praktikumsaufgaben wird diese Idee jedoch verworfen.

```
SELECT PRAKTIKUM FROM INFORMATIONSSYSTEME
    WHERE praktikumsaufgaben =
```

‘Praktikumsaufgabe 3‘

```
    AND autor = 'Steven Illg';
```

Termin: 27.April 2015
Abgabe: 11.Mai 2015

Aufgabe 3.1. Der `ALTER TABLE` Befehl

Fügen Sie bei der Relation `Lieferanten` ein weiteres Attribut `Telefonnummer` hinzu und passen Sie Ihre Testdaten an.

Um eine bessere Übersichtlichkeit für die SQL-Befehle zu gewährleisten, werden `TABELLENNAMEN` zukünftig nur noch in Großbuchstaben und `spaltennamen` durchgehend in Kleinbuchstaben geschrieben.

Jeder Befehl, der eine Relation verändert, beginnt mit `ALTER TABLE`. Nach dieser Anweisung folgt der Name der Relation, die geädert werden soll. Im nächsten Schritt wird angegeben, was das DBMS tun soll – mit `ADD` fügt es z.B. ein neues Attribut mit definierten Bedingungen hinzu.

```
1    ALTER TABLE <TABELLENNAME>
2      ADD <spaltenname> [Bedingung/en];
```

Folgender `ALTER TABLE` Befehl *(englisch für „Tabelle ändern“),* fügt das Attribut `telefonnummer` zur `LIEFERANTEN` Relation hinzu:

```
1    ALTER TABLE LIEFERANTEN
2      ADD telefonnummer VARCHAR2(18);
```

Eine `UPDATE`-Anweisung besteht aus drei Teilen: Der Anweisung selbst mit der betreffenden Relation, das betroffene Attribut mit dem neu zusetzende Attributwert *(angesprochen mit `SET`)* und einem Bezugsattribut mit einem Bezugsattributwert *(mit `WHERE`)*. Der Attributwert, auf dem sich der neu zusetzende Wert bezieht, kann aus derselben oder einer anderen Spalte stammen. Formal wird dies folgendermaßen notiert:

```
1  UPDATE <TABELLENNAME>
2    SET <spaltenname> = '[neuer Wert]'
3    WHERE <spaltenname> = '[vorhandener Wert]'
```

Mit einer UPDATE-Anweisung kann man Daten in eine neu angelegte Spalte einzeln eintragen *(siehe unten für LIEFERANTEN)*, oder mit einer allgemeinen Bedingung *(siehe noch weiter unten bei ARTIKEL)* vom DBMS füllen lassen. Außerdem eignet sich eine UPDATE-Anweisung auch dazu, bereits vorhandene Attributwerte zu aktualisieren.

```
1    UPDATE LIEFERANTEN
2      SET telefonnummer = '017612345612'
3      WHERE lieferantennummer = 1789123;
4    UPDATE LIEFERANTEN
5      SET telefonnummer = '017961212345'
6      WHERE lieferantennummer = 3123789;
7    UPDATE LIEFERANTEN
8      SET telefonnummer = '017661255882'
9      WHERE lieferantennummer = 3128889;
```

Die ursprüngliche Lieferantentabelle:

	LIEFERANTENNUMMER	NAME	STRAßE	PLZ	ORT
1	1789123	Günes-Gözlügü AG	Sonnenallee 112	77345	Pasadena
2	2456456	Clock-Work GmbH	Additionsstraße 16	42351	Albuquerque
3	3123789	Schmuck-und-Accessoires GbR	Ticari Weg 24a	01934	(null)
4	3165689	Schmuck-und-Herrenartikel	Ticari Weg 24b	01934	(null)
5	3128889	Your Brends Herrenarmbänder	Schlangenhügel 10	20224	Atlanta

Die Lieferantentabelle nach ALTER TABLE und UPDATE:

	LIEFERANTENNUMMER	NAME	STRAßE	PLZ	ORT	TELEFONNUMMER
1	1789123	Günes-Gözlügü AG	Sonnenallee 112	77345	Pasadena	017612345612
2	2456456	Clock-Work GmbH	Additionsstraße 16	42351	Albuquerque	(null)
3	3123789	Schmuck-und-Accessoires GbR	Ticari Weg 24a	01934	(null)	017961212345
4	3165689	Schmuck-und-Herrenartikel	Ticari Weg 24b	01934	(null)	(null)
5	3128889	Your Brends Herrenarmbänder	Schlangenhügel 10	20224	Atlanta	017661255882

Spätestens jetzt kann man anhand der Tabelle erkennen, dass NULL-Werte für telefonnummer zulässig sind.

Fügen Sie der Relation Artikel ein weiteres Attribut ABCKlassifikation, mit den möglichen Werten 'A', 'B' und 'C' hinzu und passen Sie Ihre Testdaten an.

Mit dem nächsten ALTER TABLE Befehl wird vorgestellt, wie das Attribut ABCKlassifikation zur Artikeltabelle hinzufügt wird. Außerdem wurde eine CHECK-Klausel formuliert, die die Eingabemöglichkeit nur auf die Attributwerte 'A', 'B' und 'C' zulässt.

```
1  ALTER TABLE ARTIKEL
2    ADD abcklassifikation CHAR(1)
3    CHECK (abcklassifikation IN ('A', 'B', 'C'));
```

Mit den vier unten stehenden UPDATE-Anweisungen, wird das Attribut ABCKlassifikation automatisch vom DBMS mit den drei zulässigen Attributwerten befüllt und klassifiziert. Dies wird durch die SET-WHERE-Klausel ermöglicht. Alle Artikel, die zur Produktgruppe Herrenuhren gehören, erhalten die Klassifikation 'A', die Frauenuhren 'B' und alle Sonnenbrillen und Accessoires 'C'.

```
1    UPDATE ARTIKEL
2      SET abcklassifikation = 'A'
3      WHERE produktgruppe = 'Herrenuhren';
4    UPDATE ARTIKEL
5      SET abcklassifikation = 'B'
6      WHERE produktgruppe = 'Frauenuhren';
7    UPDATE ARTIKEL
8      SET abcklassifikation = 'C'
9      WHERE produktgruppe = 'Herrenarmbänder';
```

```
10   UPDATE ARTIKEL
11     SET abcklassifikation = 'C'
12     WHERE produktgruppe = 'Sonnenbrillen';
```

Die ursprüngliche Artikeltabelle:

	ARTIKELNUMMER	ARTIKELNAME	PRODUKTGRUPPE	VERKAUFSPREIS
1	14460	James Audrey TikTok	Herrenuhren	149
2	12440	James Audrey Business Clock	Herrenuhren	349
3	12450	James Audrey Man Fitness	Herrenuhren	149
4	13670	James Audrey Lady Fitness	Frauenuhren	149
5	14456	James Audrey Mrs Audreys Chrono One	Frauenuhren	179
6	12456	James Audrey Chrono One	Herrenuhren	249
7	12459	James Audrey Big Data	Herrenuhren	149
8	12457	James Audreys Window	Sonnenbrillen	99,99
9	13675	James Audrey Mrs Audreys Window	Sonnenbrillen	89,99
10	13645	James Audrey Big Data Mrs	Frauenuhren	149
11	12451	James Audrey Cameron Bends	Herrenarmbänder	49,99

Die Artikeltabelle nach `ALTER TABLE` und `UPDATE`:

	ARTIKELNUMMER	ARTIKELNAME	PRODUKTGRUPPE	VERKAUFSPREIS	ABCKLASSIFIKATION
1	14460	James Audrey TikTok	Herrenuhren	149	A
2	12440	James Audrey Business Clock	Herrenuhren	349	A
3	12450	James Audrey Man Fitness	Herrenuhren	149	A
4	13670	James Audrey Lady Fitness	Frauenuhren	149	B
5	14456	James Audrey Mrs Audreys Chrono One	Frauenuhren	179	B
6	12456	James Audrey Chrono One	Herrenuhren	249	A
7	12459	James Audrey Big Data	Herrenuhren	149	A
8	12457	James Audreys Window	Sonnenbrillen	99,99	C
9	13675	James Audrey Mrs Audreys Window	Sonnenbrillen	89,99	C
10	13645	James Audrey Big Data Mrs	Frauenuhren	149	B
11	12451	James Audrey Cameron Bends	Herrenarmbänder	49,99	C

Möchte man ein vorhandenes Attribut nachträglich ändern *(z.B. die Attributbezeichnung oder den Datentyp),* dann braucht man folgenden Befehl:

```
1     ALTER TABLE <TABELLENNAME>
2       [ALTER COLUMN | MODIFY]
3       <spaltenname> [neue Bedingung];
```

Mit DROP kann man eine komplette Spalte oder Tabelle löschen:

```
1     ALTER TABLE <TABELLENNAME>
2       DROP [COLUMN | TABLE] [<Attribut> | <Relation>];
```

Und Attributwerte löscht man mit einer DELETE-Anweisung:

```
1     DELETE FROM <TABELLENNAME>
2       WHERE <spaltenname> [Bedingung];
```

Hinweis zu ALTER TABLE und UPDATE

Für ALTER TABLE und UPDATE ist zu beachten, dass pro Anweisung nur eine Veränderung vorgenommen werden kann. Aus diesem Grund sind die UPDATE-Anweisungen der Lieferantentabelle auf drei und die der Artikeltabelle auf vier einzelne Befehle aufgeteilt und nicht in einem durchgehenden Befehl vorzufinden.

Beachte, dass man bei einer UPDATE-Anweisung für SET und WHERE meistens nicht denselben Spaltenname wählen sollte, da das DBMS unter Umständen keinen Bezug zwischen den einzelnen Tupel eines Attributs herstellen und neue Werte setzen kann. Der Befehl wäre syntaktisch zwar korrekt, jedoch würde dies zu einer `0 Zeilen aktualisiert.` Meldung in der Skriptausgabe führen. Diese Entscheidung macht lediglich nur dann Sinn, wenn man in einer Relation einen oder sich wiederholenden Wert hat und diesen komplett austauschen möchte *(beispielsweise den Namen „Müller" in „Bauer" setzen).*

Aufgabe 3.2. Formulierung von SQL-Queries

Am Anfang ist der Inhalt aller verwendeten Tabellen mit dem Befehl `SELECT * FROM …` auszugeben.

Die abfragerelevanten Tabellen sind innerhalb der Queries stets bei `FROM <TABELLENNAME>` einzusehen. Zur Lösung dieser Aufgabe werden fast alle Tabellen benötigt, die bis zur Praktikumsaufgabe 2 erstellt wurden. Nur die Relation `Lieferanten` wird nicht gebraucht.

Die Tabellen werden an dieser Stelle nicht nochmals gezeigt. Stattdessen verweisen folgende Seitenzahlen auf den Ort in diesem Buch:

Kunden Seite **40**

Bestellungen Seite **41**

Bestellpositionen Seite **42**

Artikel Seite **60**

Konditionen Seite **45**

Für jede Query ist die Aufgabenstellung, den SQL-Code, eine kurze Beschreibung und das Ausgabeergebnis abzugeben.

Auf den Folgenden acht Seiten werden die SQL-Abfragen *(Queries)* von *a.* bis *h.* nacheinander gezeigt und erklärt. Es ist zu beachten, dass die Abfragen in diesem Buch nur eine Möglichkeit von mehreren darstellen und keineswegs auf Vollständigkeit beruhen. Sie zeigen beispielhaft einen Weg, der von einfachen bis hin zu komplexen Abfragemustern geht.

a. Wie viele Artikel gibt es?

```
1    SELECT COUNT (artikelnummer) AS artikelanzahl
2      FROM ARTIKEL;
```

Der Befehl `SELECT COUNT ...` steht für: *„Zähle alle Zeilen aus der Spalte Artikelnummer und gebe die Summe als Artikelanzahl aus“,* wobei das `COUNT` für *„zähle“* steht. Anstelle von `COUNT (artikelnummer)` wäre auch `COUNT (*)` möglich. Sowohl `AS`, was einer Spalte einen Alias verleiht, als auch `AS artikelanzahl` sind nicht zwingend notwendig, jedoch benennt diese Anweisung die Ausgabe um und gibt die Spalte mit der gewünschten Bezeichnung `Artikelanzahl` aus.

	ARTIKELANZAHL
1	11

b. Gebe die Artikelnamen mit der Klassifikation „A“ aufsteigend sortiert aus.

```
1    SELECT * FROM ARTIKEL
2      WHERE abcklassifikation = 'A'
3      ORDER BY artikelname ASC;
```

Das `SELECT * FROM ...` steht für: *„Gebe alle Spalten aus der Artikeltabelle aus“,* wobei das Sternchen für *„alle Spalten“* steht. Durch die `WHERE`-Klausel wird die Zeilenausgabe auf alle Zeilen beschränkt, die in der Spalte `ABCKlassifikation` zum Attributwert `'A'` gehören. Das `ORDER BY ... ASC` *(siehe Seite 13)* sortiert abschließend die Zeilen der Spalte `Artikelname` alphabetisch aufsteigend nach Name.

	ARTIKELNUMMER	ARTIKELNAME	PRODUKTGRUPPE	VERKAUFSPREIS	ABCKLASSIFIKATION
1	12459	James Audrey Big Data	Herrenuhren	149	A
2	12440	James Audrey Business Clock	Herrenuhren	349	A
3	12456	James Audrey Chrono One	Herrenuhren	249	A
4	12450	James Audrey Man Fitness	Herrenuhren	149	A
5	14460	James Audrey TikTok	Herrenuhren	149	A

c. Geben Sie die Nummer des Lieferanten aus, der den Artikel mit z.B. der Artikelnummer 100 am günstigsten anbietet.

```
1    SELECT lieferantennummer,
2    MIN(einkaufspreis) AS günstigste
3         FROM KONDITIONEN
4         WHERE artikelnummer = 12451
5         GROUP BY lieferantennummer
6         ORDER BY günstigste ASC;
```

Mit `SELECT lieferantennummer … FROM KONDITIONEN` wird das Attribut `lieferantennummer` aus der Konditionstabelle als erste Spalte im Abfrageergebnis ausgegeben. Für die zweite Spalte filtert das DBMS mit der `MIN`-Anweisung den kleinsten Attributwert aus der Spalte `Einkaufspreis` heraus und gibt ihr mit `AS` den Alias *„günstigste“*. Die erste Zeile lautet also umgangssprachlich: *„Gebe die Lieferantennummern und den kleinsten Attributwert aus der Spalte Einkaufspreis als ‚günstigste‘ aus“*.

Mit Hilfe der `WHERE`-Klausel fokussiert sich die Ausgabe auf alle Lieferanten, die den Artikel mit der Artikelnummer `12451` liefern. Mit einer `GROUP BY` Anweisung, die für: *„Gruppiere alle doppelten Zeilen nach Lieferantennummer“* steht, wird die Ausgabe nach Lieferanten gruppiert. Der tatsächlich günstigste Einkaufspreis ist zum Schluss durch das `ORDER BY günstigste ASC` rechts oben im ersten Tupel zu finden. Für einen Preisvergleich stehen die Zeilen darunter zur Verfügung.

	LIEFERANTENNUMMER	GÜNSTIGSTE
1	3123789	19,99
2	3128889	22
3	3165689	25,45

d. Die Namen derjenigen Kunden ausgeben, in deren Wohnort es eine Burg gibt oder gab (wie z.B. in Hamburg).

```
1    SELECT name, ort FROM KUNDEN
2      WHERE ort LIKE '%que%';
```

Das `SELECT ... FROM KUNDEN` gibt auch hier an, dass die Attribute `name` und `ort` aus der Kundentabelle ausgegeben werden sollen. Direkt im Anschluss wird die Ausgabe mit `WHERE` auf die Attributwerte beschränkt, bei denen ein `'que'` im Ortsnamen enthalten ist. Folgerichtig steht `LIKE` für: *„Die Zeilenausgabe auf Worte beschränken, bei denen im Wort ein ‚que' enthalten ist"*. Anstelle von `'que'` hätte man auch `'Burg'` schreiben können, da aber keiner der Kunden in einem Ort lebt, bei dem das Wort *„Burg"* enthalten ist, wurde als Alternative das *„que"* gewählt. In diesem Datenbankbeispiel erwartet man den Ortsname *„Albuquerque"*, New Mexico. Das `%` steht in diesem Statement für: *„Mit beliebige Zeichen vor und nach dem ‚que'"*.

Bei einer `LIKE`-Abfrage ist zu beachten, dass das DBMS zwischen Groß- und Kleinschreibung unterscheidet. Die Abfrage mit `'%albu%'` würde im Gegensatz zu `'%Albu%'` zu keinem gefundenen Ort führen.

	NAME	ORT
1	Walter White	Albuquerque
2	Skyler White	Albuquerque
3	Saul Goodman	Albuquerque

e. Wie hoch ist der Durchschnittspreis bei den Konditionen zu einem bestimmten Artikel?

```
1    SELECT AVG (einkaufspreis) AS Durchschnitt
2      FROM KONDITIONEN
3      WHERE artikelnummer = 12451;
```

Das AVG steht für: *„Berechne den Durchschnitt“ (und AVG wiederum für das englische Wort „average“, was „Durchschnitt“ bedeutet).* Die Eingabe in der Klammer nach dem AVG gibt das Attribut an, von dem der Durchschnitt berechnet werden soll *(hier einkaufspreis, vgl. COUNT in Abfrage a. auf Seite 63),* der mit der WHERE-Klausel auf den Artikel 12451 beschränkt wird. Die Daten bezieht das DBMS nur aus der Konditionstabelle. Mit der AS-Anweisung wird der errechnete Durchschnittspreis des gewünschten Artikels als Durchschnitt ausgegeben. Es ist zu beachten, dass man zur Berechnung des Durchschnittspreises verschiedene Einkaufspreise in der Konditionstabelle zu einem Artikel braucht.

	DURCHSCHNITT
1	22,48

f. Lassen Sie sich die Anzahl der Bestellpositionen je Bestellung (also je Bestellnummer) ausgeben.

```
1    SELECT bestellnummer, COUNT (*) AS anzahl
2      FROM BESTELLPOSITIONEN
3        GROUP BY bestellnummer
4        ORDER BY bestellnummer ASC;
```

Mit SELECT bestellnummer werden alle Bestellnummern aus der Bestellpositionen-Tabelle ausgegeben und mit COUNT (*) werden alle

Tupel in dieser Relation gezählt. Mit `GROUP BY bestellnummer` wird die Ausgabe nach Bestellnummer gruppiert. Dadurch wird die jeweils bestellte Menge aller Artikel je Bestellnummer zusammenaddiert.

Ferner bezieht sich das `SELECT` auch auf die Artikelsumme, die mit `AS` den Alias *„Anzahl"* bekommt. Die Daten für die Abfrage bezieht das DBMS aus der Tabelle `Bestellpositionen`. Letztlich wird aufgezeigt, wie viele Artikelnummern zu einer Bestellnummer gehören oder genauer gesagt: Wie viele Bestellpositionen eine Bestellung enthält.

Das `ORDER BY` am Ende ist nicht notwendig, allerdings schenkt eine Sortierung nach `bestellnummer` eine kleine Übersichtlichkeit.

	BESTELLNUMMER	ANZAHL
1	121226	3
2	121227	1
3	121228	1
4	121229	1
5	121230	6
6	121231	3
7	121233	1
8	121234	2

g. Lassen Sie sich den Gesamtwert der Bestellpositionen je Kunde (Name und Kundennummer) ausgeben.

```
1     SELECT K.kundennummer, name,
2     SUM(menge * verkaufspreis) AS gesamtwert
3     FROM KUNDEN K, BESTELLUNGEN B,
4     BESTELLPOSITIONEN BPOS, ARTIKEL A
5       WHERE K.kundennummer = B.kundennummer
6       AND B.bestellnummer = BPOS.bestellnummer
7       AND BPOS.artikelnummer = A.artikelnummer
8         GROUP BY K.kundennummer, name
9         ORDER BY K.kundennummer ASC;
```

Im Vergleich zu den vorherigen Abfragen werden *g.* und *h.* etwas komplexer, beherbergen aber eine klare, nachvollziehbare Struktur.

Mit dem `SELECT` Befehl werden insgesamt drei Spalten ausgegeben: Die `kundennummer` und der `name` aus der Kundentabelle und der `Gesamtwert` der Bestellpositionen je Kunde.

Zur Vereinfachung und für eine verbesserte Lesbarkeit der Abfrage, werden die Bezeichnungen der Relationen mit einem vorübergehenden Alias versehen. Die Relation Kunden wird mit **`K`**, Bestellungen mit **`B`**, Bestellpositionen mit **`BPOS`** und Artikel mit **`A`** abgekürzt.

Mit `K.kundennummer` in Zeile 1. wird deutlich definiert, dass die Kundennummer aus der Kundentabelle und nicht aus der Bestelltabelle bezogen werden soll. Dies ist vor allem dann bedeutsam, wenn das DBMS auf zwei Relationen zugreifen soll, in der inhaltlich identische Attribute vorzufinden sind. Im Gegensatz dazu kommt das Attribut `name` nur in der Kundentabelle vor und braucht deshalb keine explizite Tabellendefinition. Die letzte Spalte ist das Produkt aus der Bestellmenge und dem Verkaufspreis. Die Bedingung wird mit `SUM` formuliert, dass für *„berechne die Summe von …"* steht. Durch die `AS`-Anweisung wird das Abfrageergebnis als *„Gesamtwert"* ausgegeben.

Um diese Abfrage ausführen zu können, muss das DBMS auf alle vier Relationen aus der Praktikumsaufgabe 1 zugreifen. Um dies möglich zu machen, werden die Relationen mit Hilfe von `WHERE`, `AND` und einem Gleichheitsoperator miteinander *verknüpft.* Zeile 3. bis 7. bedeutet umgangssprachlich: *„Beziehe dich auf die Daten der Kunden-, Bestell-, Bestellpositions- und Artikeltabelle, bei denen die Kundennummer in der Kunden- und Bestelltabelle identisch ist, die Bestellnummer bei der Bestell- und Bestellpositionstabelle und die Artikelnummer bei der Bestellpositions- und Artikeltabelle."* Jetzt wird deutlich, dass das `WHERE` für: *„Bei denen … ist"* steht oder für *„wo … ist"* und `AND` für *„und"* oder *„und auch auf …"*. Die Verknüpfung von Relationen wird nur durch die Überprüfung auf Gleichheit möglich, die aufgrund der Primary und Foreign Keys gegeben ist.

Zum Abschluss wird die Ausgabe nach `K.kundennummer` und `name` gruppiert. Dadurch werden gleiche Kundennummern zusammengefasst und die Produkte aus der Multiplikation addiert. Außerdem wird die Ausgabe aufsteigend nach `K.kundennummer` sortiert.

	KUNDENNUMMER	NAME	GESAMTWERT
1	11111	Rick Grimes	696
2	152341	Abraham Ford	796,97
3	223344	Hank Shrader	498
4	324567	Sheldon Cooper	99,98
5	2948517	Gustavo Fring	3146,94
6	3987634	Howard Wolowitz	199,96

h. Geben Sie die Bezeichnungen der Artikel aus, die z.B. der Kunde mit dem Namen „Asterix" bestellt hat.

```
1    SELECT artikelname, name
2    FROM KUNDEN K, BESTELLUNGEN B,
3    BESTELLPOSITIONEN BPOS, ARTIKEL A
4      WHERE K.kundennummer = B.kundennummer
5      AND B.bestellnummer = BPOS.bestellnummer
6      AND BPOS.artikelnummer = A.artikelnummer
7      AND name LIKE '%Gustavo%'
8        GROUP BY artikelname, name
9        ORDER BY artikelname ASC;
```

Die Zeilen 2.-6. der Abfrage *h.* wurden analog zu den Zeilen 3.-7. der Abfrage *g.* formuliert. In Zeile 1. wird wie gewohnt mit einem `SELECT`-Befehl vorgegeben, welche Spalten ausgegeben werden sollen. Weil das Attribut `artikelname` nur in der Artikeltabelle und `name` nur in der Kundentabelle vorkommt, ist keine explizite Tabellendefinition notwendig *(vgl. Abfrage g. Zeile 1.)*. Auch hier muss das DBMS auf alle vier Relationen aus der Praktikumsaufgabe 1 zugreifen. Daher werden die Relationen mit Hilfe von `WHERE`, `AND` und einem Gleichheitsoperator miteinander *verknüpft (vgl. Abfrage g., Zeile 5.-7.)*. Anders als bei Abfrage *g.*, wurde hier in Zeile 7. ein weiteres `AND` und ein `LIKE` verwendet, die zusätzlich überprüfen, ob in der Spalte `name` ein Teilattributwert anstelle von

„Asterix“ der Name *„Gustavo“* enthalten ist. Dadurch gibt das DBMS nur Artikelnamen aus, die der Kunde mit dem Namen `Gustavo` bestellt hat. Zum Schluss wird die Ausgabe nach `artikelname` und `name` gruppiert und aufsteigend nach `artikelname` sortiert.

	ARTIKELNAME	NAME
1	James Audrey Big Data	Gustavo Fring
2	James Audrey Big Data Mrs	Gustavo Fring
3	James Audrey Cameron Bends	Gustavo Fring
4	James Audrey Chrono One	Gustavo Fring
5	James Audrey Mrs Audreys Window	Gustavo Fring
6	James Audreys Window	Gustavo Fring

Hinweis zu `artikelname` aus Abfrage *h.*

Zunächst wurde der Artikel *„James Audrey Big Data“* nur einmal im Abfrageergebnis aufgeführt, obwohl Herr Fring sowohl die Frauenuhr, als auch die Herrenuhr bestellt hat. Da der `artikelname` für beide Produkte zunächst identisch war, wurden diese Attributwerte aufgrund der `GROUP BY` Anweisung zusammengefasst. Das führte aber dazu, dass von den sechs unterschiedlichen Artikeln die bestellt wurden, lediglich fünf im Abfrageergebnis erschienen sind, obwohl es sich um zwei unterschiedliche Artikel mit zwei unterschiedlichen Artikelnummern handelt. Dies stellte ein Problem dar und zeigt, wie einfach es zu einem verfälschten Abfrageergebnis kommen kann.

Beachte: Ein Attributwert sollte nicht doppelt vorkommen, wenn er von anderen identifizierbar sein soll *(z.B. unterschiedliche Artikel mit individuellen Bezeichnungen und `Name` als Schlüsselkandidat).*

Man kann solch ein Problem auf verschiedene Art und Weise lösen, indem man zu Beginn im `CREATE TABLE` Befehl einen `UNIQUE CONSTRAINT`, oder einen zusammengesetzten Primary Key formuliert, oder die Relation nachträglich mit `ALTER TABLE MODIFY` ändert *(siehe Seite 60),* oder die Datensätze mit einer `UPDATE` Anweisung anpasst *(siehe Seite 58).*

```
SELECT PRAKTIKUM FROM INFORMATIONSSYSTEME
    WHERE praktikumsaufgaben =
```

‘Praktikumsaufgabe 4‘

```
    AND autor = 'Steven Illg';
```

Termin: 11.Mai 2015
Abgabe: 26.Mai 2015

Aufgabe 4.1. Noch mehr SQL-Queries

Am Anfang ist der Inhalt aller verwendeten Tabellen mit dem Befehl `SELECT * FROM …` auszugeben.

Die abfragerelevanten Tabellen sind in der jeweiligen Query nach `FROM` oder `JOIN ON <TABELLENNAME>` einzusehen. Zur Lösung dieser Aufgabe werden alle bisher erstellten Tabellen benötigt.

An dieser Stelle verweisen die Seitenzahlen wieder auf die fünf unveränderten Tabellen in diesem Buch. Die angepasste Artikeltabelle wird im Anschluss der Übersicht auf der nächsten Seite gezeigt.

Kunden Seite **40**

- Für Abfrage *b.*, *d.*, *e.* und *h.*

Bestellungen Seite **41** und Seite **94**

- Für Abfrage *b.*, *d.*, *e.*, *h.* und *j.*

Bestellpositionen Seite **42** und Seite **93**

- Für Abfrage *b.*, *h.* und *j.*

Artikel Seite **74**

- Für Abfrage *a.*, *b.*, *c.*, *f.*, *g.*, *h.*, *i.* und *j.*

Konditionen Seite **45**

- Für Abfrage *a.*, *b.*, *f.* und *g.*

Lieferanten Seite **58**

- Für Abfrage *b.* und *g.*

Auf den Folgenden 21 Seiten werden die SQL-Abfragen von *a.* bis *j.* gezeigt und erklärt. Bei den Queries gelten dieselben Bestimmungen wie bei der Praktikumsaufgabe 3. Die angepasste Bestell- und Bestellpositionen-Tabelle für die Abfrage *h.* sind mitsamt den SQL-Befehlen zur Anpassung der Tabellen am Ende der Aufgabe aufgeführt.

Neu angepasste Artikeltabelle:

	ARTIKELNUMMER	ARTIKELNAME	PRODUKTGRUPPE	VERKAUFSPREIS	ABCKL	LAGERBESTAND
1	14460	James Audrey TikTok	Herrenuhren	149	A	15
2	12440	James Audrey Business Clock	Herrenuhren	349	A	25
3	12450	James Audrey Man Fitness	Herrenuhren	149	A	35
4	13670	James Audrey Lady Fitness	Frauenuhren	149	B	45
5	14456	James Audrey Mrs Audreys Chrono One	Frauenuhren	179	B	10
6	12456	James Audrey Chrono One	Herrenuhren	249	A	20
7	12459	James Audrey Big Data	Herrenuhren	149	A	30
8	12457	James Audreys Window	Sonnenbrillen	99,99	C	40
9	13675	James Audrey Mrs Audreys Window	Sonnenbrillen	89,99	C	50
10	13645	James Audrey Big Data Mrs	Frauenuhren	149	B	60
11	12451	James Audrey Cameron Bends	Herrenarmbänder	49,99	C	70

Für jede Query ist die Aufgabenstellung, der SQL-Code, eine kurze Beschreibung und das Ausgabeergebnis abzugeben.

Bei den folgenden zehn Abfragen sind die Hintergründe größtenteils aus dem vorherigen Praktikum bekannt. Aus diesem Grund erfolgen die Erläuterungen nicht mehr detailliert für jede Zeile, sondern allgemein und so kurz wie möglich. Es soll näher gebracht werden, was hinter dem jeweiligen SQL-Code steckt und wie dieser zu interpretieren ist.

a. Fügen Sie der Artikeltabelle das Attribut „Lagerbestand“ hinzu. Lassen Sie sich den Gesamtwert Ihrer Artikel ausgeben. Verwenden Sie dazu als Preis den kleinsten Einkaufspreis je Artikel aus der Konditionstabelle.

```
1   SELECT artikelname, lagerbestand,
2   KON.einkaufspreis,
3   SUM (lagerbestand * einkaufspreis) AS gesamtwert
4   FROM ARTIKEL A, KONDITIONEN KON
5     WHERE KON.einkaufspreis IN (
6     SELECT MIN(einkaufspreis) FROM KONDITIONEN KON
7       WHERE A.artikelnummer = KON.artikelnummer)
8         GROUP BY artikelname, lagerbestand,
9         KON.einkaufspreis
10        ORDER BY gesamtwert ASC;
```

Da der `ALTER TABLE` Befehl Bestandteil der Praktikumsaufgabe 3 war, wird auf den SQL-Code verzichtet, der das Attribut `Lagerbestand` zur Artikeltabelle hinzufügt und am Ende der Aufgabe notiert.

Um den Gesamtwert aller Artikel im Lager zu erhalten, muss der Lagerbestand mit dem Einkaufspreis multipliziert werden. Eben das steht in den ersten drei Zeilen des Befehls: *„Gebe den Artikelnamen, den dazugehörigen Lagerbestand und das Multiplikationsprodukt aus Lagerbestand und Einkaufspreis, die in der Artikel- und Konditionstabelle stehen, als Gesamtwert aus."*

Im zweiten Teil wird ein Faktor mit einer Unterabfrage *(Subquery)*, die mit einem zweiten `SELECT` im Befehl gekennzeichnet ist, auf den kleinsten Einkaufspreis je Artikel beschränkt. Die Einleitung geschieht hier in Zeile 5. über eine `WHERE-IN`-Anweisung, auf die ein weiteres `SELECT` in Klammern folgt. Die Unterabfrage in der Klammer sieht wiederum wie eine herkömmliche `SELECT` Abfrage aus. In der Unterabfrage gibt man dem DBMS die Anweisung, zur Berechnung des Gesamtwerts nur den kleinsten Einkaufspreis aus der Konditionstabelle zu nehmen – in SQL also: `SELECT MIN(einkaufspreis)`.

Die Konditionstabelle listet insgesamt dreizehn Artikel, die fast alle nur einmal vorkommen und dementsprechend nur einen Einkaufspreis haben, der automatisch als der Kleinste gewertet wird.

Interessant wird es aber bei den *„James Audreys Cameron Bends",* mit der Artikelnummer `12451`. Diese werden mit drei unterschiedlichen Einkaufspreisen zu je `19,99€`, `22,00€` und `25,45€` bezogen. Der gewünschte Faktor zur Berechnung des Gesamtwerts wäre also der kleinsten Einkaufspreis von `19,99€` pro Stück. In der zweiten Zeile der unten stehenden Ausgabetabelle kann man erkennen, dass die Unterabfrage den Wünschen entsprechend ausgeführt wird.

In Zeile 6. wird die Artikel- und die Konditionstabelle wie gewohnt mit Hilfe der Primary und Foreign Keys, über das Attribut `Artikelnummer`, miteinander verknüpft. Zum Schluss wird die Ausgabe gruppiert und aufsteigend nach `Gesamtwert` sortiert.

	ARTIKELNAME	LAGERBESTAND	EINKAUFSPREIS	GESAMTWERT
1	James Audrey Mrs Audreys Chrono One	10	71,6	716
2	James Audrey Cameron Bends	70	19,99	1399,3
3	James Audreys Window	40	39,99	1599,6
4	James Audrey Mrs Audreys Window	50	35,99	1799,5
5	James Audrey Chrono One	20	99,6	1992
6	James Audrey Business Clock	25	139,6	3490
7	James Audrey TikTok	15	59,6	4470
8	James Audrey Big Data	30	59,6	8940
9	James Audrey Man Fitness	35	59,6	10430
10	James Audrey Lady Fitness	45	59,6	13410
11	James Audrey Big Data Mrs	60	59,6	17880

b. Geben Sie die Namen der Kunden aus, die Artikel bestellt haben, die der Lieferant mit z.B. dem Namen „Asterix“ in seinem Lieferprogramm hat.

```
1  SELECT K.name AS kundenname, L.name AS lieferantenname
2      FROM KUNDEN K, BESTELLUNGEN B,
3      BESTELLPOSITIONEN BPOS, ARTIKEL A,
4      LIEFERANTEN L, KONDITIONEN KON
5        WHERE K.kundennummer = B.kundennummer
6        AND B.bestellnummer = BPOS.bestellnummer
7        AND BPOS.artikelnummer = A.artikelnummer
8        AND A.artikelnummer = KON.artikelnummer
9        AND KON.lieferantennummer = L.lieferantennummer
10       AND L.name = 'Clock-Work GmbH'
11         GROUP BY K.name, L.name
12         ORDER BY kundenname ASC;
```

Bei der Abfrage *b.* muss das DBMS auf alle sechs Tabellen zugreifen und die unterschiedlichen Attributwerte der einzelnen Attribute logisch miteinander verknüpfen, um am Ende die richtigen Namen ausgeben zu können. Von der Kundentabelle, von der die Namen der Kunden kommen, geht es zur Bestelltabelle und von dort aus zu den Bestellpositionen. Hier kann das DBMS entnehmen, welche Artikel zu welcher Bestellung gehören und wiederum von welchem Kunden bestellt wurden. Von

den Bestellpositionen aus geht es über die `Artikelnummer` zuerst zur Artikeltabelle, dann zu den Konditionen und von dort aus zu den Lieferanten. Nun kann das DBMS bestimmen, welche Artikel von welchem Lieferanten geliefert werden und in den Bestellpositionen der Bestellungen der Kunden enthalten sind. Es sind also alle Informationen aus den sechs Tabellen notwendig, um herauszufinden, welcher Lieferant die Artikel liefert, die ein Kunde bestellt hat.

Im Grunde genommen wird im Abfragebefehl ein ähnlicher Weg gegangen. Zunächst wird die Anweisung gegeben, die Kundennamen und die Lieferantennamen auszugeben. Es ist sinnvoller auch den Lieferantennamen auszugeben, da es zu einem späteren Zeitpunkt für einen selbst oder für andere vom Vorteil sein kann, wenn man beispielsweise vergessen hat oder gar nicht wissen kann, welcher Lieferantennamen in der Abfrage eingetragen wurde.

Von Zeile 4. bis 7. werden die sechs Tabellen miteinander verknüpft und bilden somit den oben aufgeführten *„roten Faden"* für die Abfrage.

In Zeile 8. steht die zusätzliche Bedingung, dass in der Ausgabe nur die Zeilen ausgegeben werden sollen, bei denen der Lieferantenname *„Clock-Work GmbH"* entspricht. Dadurch wird die Ausgabe auf einen Lieferanten beschränkt. Am Ende werden die Zeilen gruppiert und aufsteigend nach Kundenname sortiert.

	KUNDENNAME	LIEFERANTENNAME
1	Abraham Ford	Clock-Work GmbH
2	Gustavo Fring	Clock-Work GmbH
3	Hank Shrader	Clock-Work GmbH
4	Rick Grimes	Clock-Work GmbH

c. Welche Artikel haben den gleichen Verkaufspreis wie zum Beispiel der Artikel 100?

```
1    SELECT artikelname, verkaufspreis
2    FROM ARTIKEL
3      WHERE verkaufspreis = (
4        SELECT verkaufspreis FROM ARTIKEL
5        WHERE artikelnummer = 12459);
```

Anders als bei der Abfrage *b.*, werden hier nur die Informationen aus der Artikeltabelle benötigt. Als erstes wird die Anweisung gegeben, dass die Spalten `Artikelname` und `Verkaufspreis` aus der Artikeltabelle ausgegeben werden sollen. Direkt im Anschluss wird die Ausgabe wiederum mit einem Gleichheitsoperator und einer Unterabfrage beschränkt. Diese filtern das Ausgabeergebnis auf alle Artikelnamen, die den gleichen Verkaufspreis haben wie der Artikel mit der Artikelnummer `12459`. Das „=“ bedeutet also: *„Nur die Artikelnamen aus der Artikeltabelle ausgeben, bei denen der Verkaufspreis gleich hoch ist, wie bei dem Artikel mit der Artikelnummer 12459.“* Aufgrund dieser Bedingung vergleicht das DBMS alle Zeilen aus `Verkaufspreis` mit dem Preis von Artikel `12459`.

Das DBMS findet tatsächlich fünf von elf Artikel aus dem fiktiven Sortiment, die den gleichen Verkaufspreis haben:

	ARTIKELNAME	VERKAUFSPREIS
1	James Audrey TikTok	149
2	James Audrey Man Fitness	149
3	James Audrey Lady Fitness	149
4	James Audrey Big Data	149
5	James Audrey Big Data Mrs	149

d. Geben Sie die Namen der Kunden aus, für die es Bestellungen gibt. Formulieren Sie diese Query auf vier verschiedene Arten: Mit `WHERE`, `JOIN`, einer Subquery und `EXISTS`.

Query mit `WHERE`

```
1    SELECT name FROM KUNDEN K, BESTELLUNGEN B
2      WHERE K.kundennummer = B.kundennummer
3        GROUP BY name
4        ORDER BY name ASC;
```

Bei einer Abfrage mit `WHERE` wird zunächst die Kundentabelle mit der Bestelltabelle verknüpft. Durch das `WHERE` vergleicht das DBMS das Attribut `Kundennummer` zwischen den beiden Tabellen und gibt anschließend alle Kundennamen aus, von denen die Kundennummer in der Bestelltabelle vorkommt. Es gilt: Wenn eine Kundennummer in der Bestelltabelle ist, dann hat der dazugehörige Kunde etwas bestellt.

Query mit `JOIN`

```
1    SELECT name FROM KUNDEN K
2      INNER JOIN BESTELLUNGEN B
3        ON K.kundennummer = B.kundennummer
4          GROUP BY name
5          ORDER BY name ASC;
```

Bei der zweiten Methode wurde ein `INNER JOIN` gewählt, bei dem sich die Herangehensweise völlig von der mit der `WHERE`-Klausel unterscheidet. Hier werden die Tabellen zwar auch durch das `INNER JOIN ON` miteinander verknüpft, aber nun werden die Spalten nicht mehr auf Gleichheit überprüft, um anschließend die Namen der identischen Attributwerte von `Kundennummer` auszugeben. Der `INNER JOIN` bildet ein *kartesisches Produkt* von allen Attributen die in beiden Tabellen vorhanden sind *(auch für die, die sie nicht gemeinsam haben)*. Durch das bilden

eines kartesischen Produkts, entsteht eine enorm hohe Zeilenausgabe, da jede Zeile der einen Relation mit jeder Zeile der anderen Relation verknüpft wird. Deshalb ist eine `GROUP BY` Anweisung in Kombination mit einem `JOIN`, stets zu empfehlen, um die überflüssigen Zeilen auszufiltern. Auf diese Weise werden alle Kundennamen ausgegeben, von denen die Kundennummer in der Bestelltabelle ist. Folgerichtig sind dies jene Kunden, die eine Bestellung aufgegeben haben.

Subquery mit `IN`

```
1    SELECT name FROM KUNDEN
2      WHERE kundennummer IN (
3        SELECT kundennummer FROM BESTELLUNGEN)
4          ORDER BY name ASC;
```

Bei der dritten Methode wird eine `IN`-Unterabfrage angewendet, bei der das DBMS überprüft, welche Kundennummern aus der Kundentabelle bei den Kundennummern in der Bestelltabelle vorkommen. Umgangssprachlich bedeutet der Befehl also: *„Überprüfe, welche Kundennummern aus der Kundentabelle in der Spalte Kundennummer bei der Bestelltabelle vorkommen und gebe die dazugehörigen Kundennamen aus."* Die Herangehensweise lässt an die der `WHERE`-Klausel erinnern, allerdings hat man hier den Vorteil, dass die Tabellen vorher nicht miteinander verknüpfen werden müssen, weil die Unterabfrage separat auf die Bestelltabelle zugreift.

Subquery mit `EXISTS`

```
1    SELECT name FROM KUNDEN K
2      WHERE EXISTS (
3        SELECT kundennummer FROM BESTELLUNGEN B
4          WHERE K.kundennummer = B.kundennummer)
5            ORDER BY name ASC;
```

Bei der letzten Methode wird eine `EXISTS`-Unterabfrage angewendet, bei der die Spalte `Kundennummer` aus der Kunden- und Bestelltabelle miteinander verknüpft und anschließend verglichen wird, um heraus zu

finden, ob gleiche Attributwerte in der Spalte `Kundennummer` vorhanden sind. Ist dies der Fall, wird der Name des dazugehörigen Kunden ausgegeben. Umgangssprachlich kann man diesen Befehl auch folgendermaßen formulieren: *„Nimm die Spalte Kundennummer aus der Kunden- und Bestelltabelle und überprüfe, welche Kundennummern aus der Kundentabelle in der Bestelltabelle existieren und gebe ihre dazugehörigen Kundennamen aus."* Auch hier impliziert, dass ein Kunde eine Bestellung aufgegeben haben muss, wenn seine Kundennummer in der Bestelltabelle auftaucht.

Das Ausgabeergebnis sieht für alle vier Abfragen identisch aus, daher wird das Bild der Tabelle nur einmal eigebunden:

	NAME
1	Abraham Ford
2	Gustavo Fring
3	Hank Shrader
4	Howard Wolowitz
5	Rick Grimes
6	Sheldon Cooper

Hinweis zu `WHERE`, `JOIN`, `IN` und `EXISTS`

`WHERE`, `JOIN`, `IN` und `EXISTS` wirken auf den ersten Blick relativ gleich bzw. ähnlich, dennoch gibt es bedeutende Unterschiede zwischen diesen Methoden, die an dieser Stelle kurz angerissen werden:

Die Abfragezeit kann sich bei wesentlich größeren Tabellen stark voneinander unterscheiden. Außerdem können `NULL`-Werte bei einer `IN`-Unterabfrage für eine völlig leere Ausgabe sorgen, wohingegen `NULL`-Werte bei einer `EXISTS`-Unterabfrage keinen Einfluss auf das Ausgabeergebnis nehmen. Man sollte also seine Tabellen und die Abfragemethoden kennen, um das bestmögliche aus ihnen herauszuholen.

e. Geben Sie die Namen der Kunden aus, die nichts bestellt haben, für die es also keine Bestellungen gibt.

```
1    SELECT kundennummer, name FROM KUNDEN
2      WHERE kundennummer NOT IN (
3        SELECT kundennummer FROM BESTELLUNGEN)
4          ORDER BY kundennummer ASC;
```

Bis auf wenige Unterschiede ist die Abfrage *e.* mit der `IN`-Unterabfrage aus der Aufgabe *d.* identisch. Endscheidend für diese Abfrage ist das Schlüsselwort `NOT IN`, mit der das DBMS überprüft, welche Kundennummern aus der Kundentabelle nicht bei den Kundennummern in der Bestelltabelle vorkommen. Umgangssprachlich kann man auch sagen: *„Überprüfe, welche Kundennummern aus der Kundentabelle nicht in der Spalte Kundennummer in der Bestelltabelle vorkommen und gebe die dazugehörigen Kundennamen aus."* Mit der `IN`-Unterabfrage aus *d.* wurde hingegen geprüft, welche Kundennummern aus der Kundentabelle in der Bestelltabelle sind. Nun wird überprüft, welche Kundennummern in der Kundentabelle sind, aber nicht in der Bestelltabelle. Wenn eine Kundennummer nicht in der Bestelltabelle ist, dann hat der Kunde auch nichts bestellt.

	KUNDENNUMMER	NAME
1	2233	Walter White
2	11112	Carl Grimes
3	11333	Hershel Greene
4	37643	Amy Farrah Fowler
5	141426	Carol Paletier
6	213454	Skyler White
7	309876	Bernadette Rostenkowski
8	385454	Rajesh Koothrappali
9	3256788	Leonard Hofstadter

f. Geben Sie die Bezeichnung der A-Artikel aus (Artikel mit der Klassifikation „A“), die zum Beispiel bei Lieferant 0815 nicht im Lieferprogramm vorkommen.

```
1     SELECT artikelname AS Lieferant3165689NichtA
2     FROM ARTIKEL A
3       WHERE A.abcklassifikation = 'A'
4       AND NOT EXISTS (
5         SELECT lieferantennummer FROM KONDITIONEN KON
6           WHERE KON.artikelnummer = A.artikelnummer
7           AND KON.lieferantennummer = 3165689);
```

Bei der Abfrage *f.* wird eine NOT-EXISTS-Unterabfrage angewendet, mit der die Anweisung gegeben wird alle Artikelnamen, bei denen in der ABCKlassifikation A steht, als Lieferant3165689NichtA auszugeben. Durch den Alias der Spalte kann man zu einem späteren Zeitpunkt nachvollziehen, um welchen Lieferanten sich die Abfrage handelt *(nämlich Lieferant3165689)* und was gefragt wurde *(NichtA, bzw. „liefert nicht die Artikel mit der Klassifikation A“)*. Das impliziert allerdings nicht, dass der Lieferant 3165689, *„Schmuck-und-Herrenartikel“,* keine A-Artikel in seinem Lieferprogramm führt. Die Bedingung von Zeile 3. bis 7. ermöglicht es, dass alle Artikelnamen mit der Klassifikation A ausgegeben werden, die der Lieferant 3165689 nicht liefert. Mit der NOT-EXISTS-Unterabfrage werden alle Lieferantennummern aus der Konditionstabelle mit den Artikelnummern und ihren ABCKlassifikationen aus der Artikeltabelle gegenüber gestellt. Anschließend werden diejenigen A-Artikelnamen ausgegeben, von denen die Artikelnummern nicht beim Lieferanten 3165689 in der Konditionstabelle stehen.

Der Lieferant mit der Lieferantennummer 3165689 führt in seinem Lieferprogramm die Menge $L_1 = \{12450A, 12440A, 12451C\}$, aber nicht die Menge $L_2 = \{14460A, 12456A, 12459A\}$, wobei die Zahlen für die Artikelnummern und die Buchstaben für Klassifikation stehen. Das erwartete Ausgabeergebnis besteht somit aus Artikelnamen, dessen Artikelnummern in der Menge L_2 enthalten sind. Das Ausgabeergebnis auf der nächsten Seite zeigt, dass die Überlegungen richtig sind:

	LIEFERANT3165689NICHTA
1	James Audrey Big Data
2	James Audrey Chrono One
3	James Audrey TikTok

g. Geben Sie für jeden Lieferanten den Namen und die Anzahl der A-, B- und C-Artikel aus ihrem Lieferprogramm aus.

```
1    SELECT name, abcklassifikation, COUNT (*) AS anzahl
2    FROM LIEFERANTEN L, ARTIKEL A, KONDITIONEN KON
3      WHERE L.lieferantennummer = KON.lieferantennummer
4      AND KON.artikelnummer = A.artikelnummer
5        GROUP BY name, abcklassifikation
6        ORDER BY abcklassifikation ASC;
```

Diese Abfrage lässt sich bequem mit einer `WHERE`-Klausel lösen, bei der insgesamt drei Spalten ausgegeben werden sollen. Die erste Spalte soll `Lieferantenname` sein, die zweite die `ABCKlassifikation` und die dritte ergibt sich aus der Summe der jeweiligen ABCKlassifikationen, die der jeweilige Lieferant in seinem Lieferprogramm führt. Außerdem soll die Summe der ABCKlassifikationen mit dem Alias `Anzahl` im Ausgabeergebnis angezeigt werden. Die drei relevanten Tabellen für diese Abfrage sind somit `Lieferanten`, `Artikel` und `Konditionen`, die wie gewohnt miteinander verknüpft werden. Hier braucht das DBMS keine Spalten und Zeilen miteinander zu vergleichen. Es zählt lediglich die ABCKlassifikationen, ordnet die Summe dem jeweiligen Lieferanten zu und gibt die Summe je Lieferant und ABCKlassifikation aus. Zum Schluss wird die Ausgabe noch gruppiert und aufsteigend nach `ABCKlassifikation` sortiert.

	NAME	ABCKLASSIFIKATION	ANZAHL
1	Schmuck-und-Herrenartikel	A	2
2	Clock-Work GmbH	A	3
3	Schmuck-und-Accessoires GbR	B	1
4	Clock-Work GmbH	B	2
5	Günes-Gözlügü AG	C	2
6	Schmuck-und-Accessoires GbR	C	1
7	Schmuck-und-Herrenartikel	C	1
8	Your Brends Herrenarmbänder	C	1

h. Geben Sie die Namen der Kunden aus, die alle Artikel aus Ihrem Sortiment bestellt haben.

```
1    SELECT name FROM KUNDEN K, BESTELLUNGEN B,
2    BESTELLPOSITIONEN BPOS, ARTIKEL A
3      WHERE K.kundennummer = B.kundennummer
4      AND B.bestellnummer = BPOS.bestellnummer
5      AND BPOS.artikelnummer = A.artikelnummer
6      AND A.artikelnummer >= ALL (
7        SELECT artikelnummer FROM ARTIKEL)
8          GROUP BY name
9          ORDER BY name ASC;
```

Zu Beginn wurden zwei Kunden ausgewählt, die noch keine Bestellung aufgegeben hatten: `Daryl Dixon` und `Saul Goodman`. Um ein sinnvolles Abfrageergebnis zu erhalten, haben die beiden mit einem `INSERT` Befehl jeden Artikel genau einmal *„bestellt“ (die Tabellen Bestellungen und Bestellpositionen wurden also erweitert).* Folgerichtig werden diese beiden Namen im Abfrageergebnis erwartet. Die `INSERT` Befehle und die erweiterten Tabellen sind im Anschluss der Aufgabe zu finden.

Von Zeile 3. bis 5. werden die vier Tabellen `Kunden`, `Bestellungen`, `Bestellpositionen` und `Artikel` wie gewohnt miteinander verknüpft. In Zeile 6. und 7. steht die Bedingung, die mit einer

ALL-Unterabfrage formuliert wurde. Umgangssprachlich bedeutet dieser Befehl: *„Gebe die Kundennamen aus, bei denen die Bestellpositionen größer oder gleich alle Artikelnummern aus der Artikeltabelle sind."* Statt einem >= ALL wäre übrigens auch ein <= ALL möglich. Der Operator muss aber an dieser Stelle notiert werden, weil das DBMS die Abfrage mit einem schlichten = oder mit keinem Operator nicht interpretieren kann. Da der Operator *größer oder gleich alle* bzw. *kleiner oder gleich alle* mathematisch alle Artikelnummern einschließt, werden nur die Kundennamen ausgegeben, die alle Artikel mindestens einmal bestellt haben.

	NAME
1	Daryl Dixon
2	Saul Goodman

i. Geben Sie für jeden Artikel die Artikelnummer, den Artikelnamen und entweder ‚billig', ‚mittel' oder ‚teuer' aus, je nachdem, ob der Preis unter z.B. 10 Euro, zwischen 10 und unter 100 Euro oder darüber (>=100 Euro) ist.

```
1    SELECT artikelnummer, artikelname,
2      CASE
3        WHEN verkaufspreis <= 50 THEN 'billig'
4        WHEN verkaufspreis <= 150 THEN 'mittel'
5        ELSE 'teuer'
6      END AS preiseinstufung
7    FROM ARTIKEL;
8      ORDER BY preiseinstufung ASC;
```

Die SQL-Abfrage der Aufgabe *i.* lässt ein wenig mehr als bisher an herkömmliche Programmiersprache erinnern. Zunächst wird dem DBMS die Anweisung gegeben, die Spalten Artikelnummer und Artikelname unverändert auszugeben. Für eine weitere, dritte Spalte Preiseinstufung, soll folgende Bedingung erfüllt werden:

Das Schlüsselwort CASE signalisiert den Beginn der Anweisung und das END den Schluss. Der Zwischenteil besteht aus *„WHEN … THEN … ELSE …“*, was nichts anderes bedeutet als: *„Wenn dies erfüllt wird, dann tu das, ansonsten tue etwas anderes.“* Es handelt sich also um einen *Wenn-Dann-Fall,* bei dem das DBMS in der Spalte Preiseinstufung automatisch bei jedem Artikel *„billig“* setzt, wenn die Bedingung *„Verkaufspreis kleiner oder gleich 50€“* erfüllt ist. Sollte der Verkaufspreis aber kleiner oder gleich 150€ sein, dann setzt das DBMS *„mittel“*. Da die Bedingung <= 50€ bereits implementiert ist, kommt es nicht zu einer Überschneidung der Bedingungen und es führt nicht zu einer *„mittel“* Ausgabe bei <= 50€. Sollten beide Bedingungen nicht erfüllt sein, dann setzt das DBMS bei jedem Artikel – das teurer als 150€ ist – automatisch *„teuer“*. In Zeile 5. würde die Anweisung

```
1     WHEN verkaufspreis > 150 THEN 'teuer'
```

zwar zum selben Ausgabeergebnis führen, allerdings zeugt diese Alternative nicht von einem gutem SQL-Stil. Mit END AS *(Zeile 6.)* wird der Alias der *„CASE-Spalte“* definiert.

Für diese Abfrage wird nur die Artikeltabelle benötigt. Außerdem wird das Ergebnis aufsteigend nach Preiseinstufung sortiert.

	ARTIKELNUMMER	ARTIKELNAME	PREISEINSTUFUNG
1	12451	James Audrey Cameron Bends	billig
2	13675	James Audrey Mrs Audreys Window	mittel
3	12457	James Audreys Window	mittel
4	13670	James Audrey Lady Fitness	teuer
5	14456	James Audrey Mrs Audreys Chrono One	teuer
6	12459	James Audrey Big Data	teuer
7	13645	James Audrey Big Data Mrs	teuer
8	12450	James Audrey Man Fitness	teuer
9	12440	James Audrey Business Clock	teuer
10	14460	James Audrey TikTok	teuer
11	12456	James Audrey Chrono One	teuer

j. Geben Sie alle Artikelnamen und das Datum aus, an dem sie bestellt wurden. Artikel, die nicht bestellt wurden, sollen auch einmal erscheinen.

```
1    SELECT artikelname, bestelldatum
2    FROM ARTIKEL A
3      LEFT OUTER JOIN BESTELLPOSITIONEN BPOS
4        ON A.artikelnummer = BPOS.artikelnummer
5      LEFT OUTER JOIN BESTELLUNGEN B
6        ON BPOS.bestellnummer = B.bestellnummer
7          GROUP BY artikelname, bestelldatum
8          ORDER BY bestelldatum ASC;
```

Für die zehnte und letzte Abfrage der Praktikumsaufgabe 4 wird eine `LEFT OUTER JOIN` Anweisung verwendet. Das `OUTER` ist im obenstehenden Befehl optional und kann nach Belieben weggelassen werden, da der davorstehende Begriff `LEFT` *(aber auch `RIGHT` oder `FULL`)* bereits eine `OUTER JOIN` Anweisung kennzeichnet.

Wie gewohnt wird zu Beginn mit `SELECT` die Anweisung gegeben, die Spalten `Artikelname` und `Bestelldatum` auszugeben.

Mit dem ersten `JOIN` wird die Artikeltabelle mit der Bestellpositionen-Tabelle an der Stelle verknüpft, an der die Spalt `Artikelnummer` aus der Artikeltabelle, gleich der Spalte `Artikelnummer` aus der Bestellpositionen-Tabelle ist. Im zweiten `JOIN` wird auf gleiche Weise die Artikeltabelle mit der Bestelltabelle verknüpft. Die *links* bzw. *rechts* Richtung mit der die Tabellen verknüpft werden, ist äußerst entscheidend für das Abfrageergebnis. Um die Unterschiede deutlich zu machen, geht die nächste Seite näher auf die möglichen Varianten ein. Die `RIGHT`'s und `LEFT`'s beziehen sich dabei auf die Zeilen 3. und 5. im Abfragecode.

Zum Schluss wird das Abfrageergebnis gruppiert und aufsteigend nach `Bestelldatum` sortiert.

- ✗ **RIGHT und dann LEFT**

Diese Anweisung würde die Datensätze der Artikeltabelle und der rechten Tabelle *(Zusatztabelle, hier Bestellpositionen)* und der linken Tabelle *(Haupttabelle, hier Bestellungen)* ausgeben. Da die linke Tabelle vorrangig behandelt wird, hat dies zur Folge, dass nur alle Artikelnamen ausgegeben werden, die auch bestellt wurden *(vorrangige Tabelle: Bestellungen)*. Die nicht bestellten Artikel werden für das Ausgabeergebnis ignoriert.

- ✗ **RIGHT und dann RIGHT bzw. LEFT und dann RIGHT**

Mit zwei `RIGHT`'s würde sich das DBMS auf zwei Zusatztabellen beziehen und mit einem `LEFT` und dann `RIGHT` auf eine Haupt- und eine Zusatztabelle. In beiden Fällen hätte dies zur Folge, dass im Ausgabeergebnis eine zusätzliche Zeile erscheint, bei der für einen Artikelnamen ein `NULL`-Wert angegeben wird. Da dieser nicht existiert, zeigt dies deutlich, dass diese Verknüpfung auch hier einen falschen Wert liefert.

- ✓ **LEFT und dann LEFT**

Nur eine Verknüpfung mit zwei Haupttabellen liefert das gewünschte Ausgabeergebnis. Auf diese Weise bezieht sich das DBMS gleichrangig auf die Relationen `Bestellungen` und `Bestellpositionen`. Dadurch werden auch die Artikel berücksichtig, die nicht bestellt wurden. Alle Zeilen sind somit gleichgestellt, weil sie aus zwei Haupttabellen *(`LEFT OUTER JOINS`)* entnommen werden. Da die nicht bestellten Artikel nun gleichgestellt mit den bestellten Artikel sind, aber dennoch kein Bestelldatum für sie existiert, erhalten diese Artikelnamen vom DBMS automatisch `NULL`-Werte für das Bestelldatum.

Ohne die `GROUP BY` Anweisung in Zeile 6. würde das Bestelldatum für alle Bestellpositionen angezeigt werden. Dies hätte zur Folge, dass auch die Zeilen ausgegeben werden, bei denen Artikelname und Bestelldatum doppelt vorkommen *(da zwei Kunden an einem Tag dasselbe bestellen können)*. Somit lässt sich das Ausgabeergebnis mit einer `GROUP BY` Anweisung auf nicht wiederholende Zeilen reduzieren.

Das Ausgabeergebnis der Aufgabe *j.* nimmt unter Berücksichtigung der aufgeführten Methode das folgende Erscheinungsbild an:

	ARTIKELNAME	BESTELLDATUM
1	James Audrey Big Data	25.04.15
2	James Audrey Big Data Mrs	25.04.15
3	James Audrey Cameron Bends	25.04.15
4	James Audrey Chrono One	25.04.15
5	James Audrey Mrs Audreys Window	25.04.15
6	James Audreys Window	25.04.15
7	James Audrey Big Data	26.04.15
8	James Audrey Big Data Mrs	26.04.15
9	James Audrey Chrono One	26.04.15
10	James Audrey Mrs Audreys Window	26.04.15
11	James Audrey Big Data Mrs	17.06.15
12	James Audrey Business Clock	(null)
13	James Audrey Lady Fitness	(null)
14	James Audrey Man Fitness	(null)
15	James Audrey Mrs Audreys Chrono One	(null)
16	James Audrey TikTok	(null)

SQL-Befehle zur Anpassung der Relationen

Anpassungsbefehle für die Abfrage *a.*:

Hinzufügen von `Lagerbestand` in die Artikeltabelle

```
ALTER TABLE ARTIKEL
  ADD lagerbestand NUMBER(6);
```

Anpassung der Testdaten von `Lagerbestand`

```
UPDATE ARTIKEL
  SET lagerbestand = 10
  WHERE artikelnummer = 14456
UPDATE ARTIKEL
  SET lagerbestand = 20
  WHERE artikelnummer = 12456;
UPDATE ARTIKEL
  SET lagerbestand = 30
  WHERE artikelnummer = 12459;
UPDATE ARTIKEL
  SET lagerbestand = 40
  WHERE artikelnummer = 12457;
UPDATE ARTIKEL
  SET lagerbestand = 50
  WHERE artikelnummer = 13675;
UPDATE ARTIKEL
  SET lagerbestand = 60
  WHERE artikelnummer = 13645;
UPDATE ARTIKEL
  SET lagerbestand = 70
  WHERE artikelnummer = 12451;
```

Anpassungsbefehle für die Abfrage *h.*:

Daryl Dixon

```
INSERT INTO BESTELLUNGEN VALUES (121235, 141424,
'11.05.2015', 1);
```

```
INSERT INTO BESTELLPOSITIONEN VALUES (121235, 14460, 1);
INSERT INTO BESTELLPOSITIONEN VALUES (121235, 12440, 1);
INSERT INTO BESTELLPOSITIONEN VALUES (121235, 12450, 1);
INSERT INTO BESTELLPOSITIONEN VALUES (121235, 13670, 1);
INSERT INTO BESTELLPOSITIONEN VALUES (121235, 14456, 1);
INSERT INTO BESTELLPOSITIONEN VALUES (121235, 12456, 1);
INSERT INTO BESTELLPOSITIONEN VALUES (121235, 12459, 1);
INSERT INTO BESTELLPOSITIONEN VALUES (121235, 12457, 1);
INSERT INTO BESTELLPOSITIONEN VALUES (121235, 13675, 1);
INSERT INTO BESTELLPOSITIONEN VALUES (121235, 13645, 1);
INSERT INTO BESTELLPOSITIONEN VALUES (121235, 12451, 1);
```

Saul Goodman

```
INSERT INTO BESTELLUNGEN VALUES (121236, 2837472,
'11.05.2015', 1);
```

```
INSERT INTO BESTELLPOSITIONEN VALUES (121236, 14460, 1);
INSERT INTO BESTELLPOSITIONEN VALUES (121236, 12440, 1);
INSERT INTO BESTELLPOSITIONEN VALUES (121236, 12450, 1);
INSERT INTO BESTELLPOSITIONEN VALUES (121236, 13670, 1);
INSERT INTO BESTELLPOSITIONEN VALUES (121236, 14456, 1);
INSERT INTO BESTELLPOSITIONEN VALUES (121236, 12456, 1);
INSERT INTO BESTELLPOSITIONEN VALUES (121236, 12459, 1);
INSERT INTO BESTELLPOSITIONEN VALUES (121236, 12457, 1);
INSERT INTO BESTELLPOSITIONEN VALUES (121236, 13675, 1);
INSERT INTO BESTELLPOSITIONEN VALUES (121236, 13645, 1);
INSERT INTO BESTELLPOSITIONEN VALUES (121236, 12451, 1);
```

```
SELECT * FROM BESTELLPOSITIONEN
```

	BESTELLNUMMER	ARTIKELNUMMER	MENGE
1	121226	12451	3
2	121226	12456	2
3	121226	12459	1
4	121227	12451	4
5	121228	12456	2
6	121229	12459	1
7	121230	12451	1
8	121230	12456	3
9	121230	12457	1
10	121230	12459	3
11	121230	13645	1
12	121230	13675	2
13	121231	12456	4
14	121231	13645	1
15	121231	13675	2
16	121233	12451	2
17	121234	12456	1
18	121234	13645	3
19	121235	12440	1
20	121235	12450	1
21	121235	12451	1
22	121235	12456	1
23	121235	12457	1
24	121235	12459	1
25	121235	13645	1
26	121235	13670	1
27	121235	13675	1
28	121235	14456	1
29	121235	14460	1
30	121236	12440	1
31	121236	12450	1
32	121236	12451	1
33	121236	12456	1
34	121236	12457	1
35	121236	12459	1
36	121236	13645	1
37	121236	13670	1
38	121236	13675	1
39	121236	14456	1
40	121236	14460	1

```
SELECT * FROM BESTELLUNGEN
```

	BESTELLNUMMER	KUNDENNUMMER	BESTELLDATUM	STATUS
1	121226	152341	25.04.15	3
2	121230	2948517	25.04.15	3
3	121233	324567	25.04.15	3
4	121228	223344	26.04.15	2
5	121229	2948517	26.04.15	2
6	121231	2948517	26.04.15	2
7	121232	324567	26.04.15	2
8	121234	11111	26.04.15	2
9	121227	3987634	27.04.15	1
10	121235	141424	11.05.15	1
11	121236	2837472	11.05.15	1

Befehle, um die Anpassungen für *h.* rückgängig zu machen

Mit den folgenden `DELETE` Befehlen werden alle Tupel aus der Bestell- und Bestellpositionen-Tabelle von `Daryl Dixon` und `Saul Goodman` gelöscht, die nur für die Abfrage *h.* relevant waren.

```
DELETE FROM BESTELLPOSITIONEN
  WHERE bestellnummer IN (121235, 121236);
```

```
22 Zeilen gelöscht.
```

```
DELETE FROM BESTELLUNGEN
  WHERE bestellnummer IN (121235, 121236);
```

```
2 Zeilen gelöscht.
```

Beachte, dass die Attributwerte der `Bestellpositionen` vor den Attributwerten der `Bestellungen` gelöscht werden müssen. Andernfalls würde man gegen den *Integritäts-Constraint* verstoßen, da kein Primary Key mehr vorhanden wäre, auf den ein Foreign Key referenziert.

```
SELECT PRAKTIKUM FROM INFORMATIONSSYSTEME
    WHERE praktikumsaufgaben =
```

‘Praktikumsaufgabe 5‘

```
    AND autor = 'Steven Illg';
```

Termin: 26.Mai 2015
Abgabe: 08.Juni 2015

Aufgabe 5.1. Der CREATE VIEW Befehl

Erstellen Sie eine VIEW ***AArtikel,*** *die nur die Artikel mit der ABCKlassifikation ‚A' enthält.*

```
1    CREATE VIEW AArtikel
2      AS SELECT *
3      FROM ARTIKEL
4        WHERE abcklassifikation = 'A';
```

Jeder Befehl, der eine VIEW erstellt, beginnt mit CREATE VIEW. Nach der Anweisung folgt der Name der VIEW, die erstellt werden soll – hier AArtikel, wobei das erste A im Namen die Artikel meint, die zur Klassifikation *„A"* gehören. Im ersten Moment lässt die erste Zeile des CREATE VIEW Befehls an einen CREATE TABLE Befehl erinnern, jedoch wird statt mit einer runden Klammer mit einem AS SELECT angegeben, welche Spalten eine VIEW beinhalten soll. In Zeile 3. wird mit der FROM-Komponente festgelegt, aus welcher – bzw. welchen – Tabellen die Daten kommen. Hier werden mit AS SELECT * FROM ARTIKEL alle Spalten aus der Artikeltabelle in die VIEW AArtikel aufgenommen. Mit der WHERE-Klausel in Zeile 4. werden die Zeilen auf diejenigen Artikel beschränkt, bei denen der Attributwert in der Spalte ABCKlassifikation gleich 'A' entspricht. Dadurch werden nur die Artikel aus der Artikeltabelle in die VIEW aufgenommen, die zur ABCKlassifikation 'A' gehören.

Am Ende der Aufgabe ist eine weitere Alternative (1) aufgeführt, die für die VIEW AArtikel auch die Namen der Lieferanten berücksichtigt. Ein CREATE VIEW Befehl sieht generell folgendermaßen aus:

```
1    CREATE VIEW <viewname>
2      AS SELECT <spaltenname1>, …, <spaltennameN>
3      FROM <tabellenname1>, …, <tabellennameN>
4        [Bedingungen];
```

Erstellen Sie eine weitere `VIEW` **`NamenBeziehung`**, *die für die Konditionen-Tupel statt der Artikelnummer und Lieferantennummer, die Bezeichnung der Artikel und den Namen des Lieferanten enthält.*

```
1   CREATE VIEW NamenBeziehung
2     AS SELECT name, artikelname
3     FROM ARTIKEL A, LIEFERANTEN L, KONDITIONEN KON
4       WHERE A.artikelnummer = KON.artikelnummer
5       AND KON.lieferantennummer = L.lieferantennummer;
```

In Zeile 1. wird die Anweisung gegeben, eine neue `VIEW` zu erstellen, die `NamenBeziehung` heißen soll. Mit `AS SELECT` werden, wie oben bereits beschrieben, die Spalten `Name` *(für die Lieferanten)* und `Artikelname` in die `VIEW` aufgenommen. In Zeile 3. gibt die `FROM`-Komponente die Tabellen `Artikel`, `Lieferanten` und `Konditionen` als Informationsquellen für diese `VIEW` an. Letztlich werden die drei relevanten Relationen wie gewohnt in Zeile 4. und 5. miteinander verknüpft. Umgangssprachlich bedeutet dieser Befehl also: *„Erstelle eine VIEW NamenBeziehung, in der die Attribute (Lieferanten)-Name und Artikelname aus den Tabellen Artikel, Lieferanten und Konditionen enthalten sind."*

Mit einer zusätzlichen Zeile könnte man eine `ORDER BY` Anweisung definieren, die jedoch nicht zwingend notwendig ist, da man diese nicht immer wünscht und nachträglich im `SELECT` Befehl vornehmen kann.

In Hinblick auf die *Generalisierung* in Aufgabe 5.4. ist zu erwähnen, dass die `VIEW NamenBeziehung` nach der Bearbeitung der Aufgabe nicht mehr funktionsfähig ist. Anstelle der `Lieferantennummer` in Zeile 5. müsste die neu angepasste `Partnernummer` stehen und anstelle der Lieferantentabelle müsste die Geschäftspartnertabelle gewählt werden – es sei denn, man würde sich auf die `VIEW Lieferanten` beziehen, die gleichbedeutend mit der Relation `Lieferanten` ist. Da sich alle Partnernummern aus der Konditionstabelle auf die Partnernummern der Lieferanten-`View` beziehen, ist die Möglichkeit von einer `VIEW` in einer anderen `VIEW` nicht ausgeschlossen. Man kann Relationen wie gewohnt mit anderen `VIEW`s verknüpfen, sofern in der `VIEW` ein Foreign Key vorhanden ist, der auf einen Primary Key referenziert.

Erstellen Sie eine dritte VIEW, die für jede Bestellnummer die ***Anzahl der Bestellpositionen*** *enthält.*

```
1    CREATE VIEW AnzahlBestellPos
2      AS SELECT bestellnummer,
3      COUNT(bestellnummer) AS Anzahl
4      FROM BESTELLPOSITIONEN
5        GROUP BY bestellnummer;
```

Auch hier wird die VIEW AnzahlBestellPos, was so viel wie *„Anzahl der Bestellpositionen"* bedeuten soll, mit einem CREATE VIEW Befehl erzeugt. Diese VIEW umfasst insgesamt zwei Attribute, die in Zeile 2. und 3. hinzugefügt werden. Ferner wird in Zeile 4. angegeben, dass alle Daten aus der Tabelle Bestellpositionen kommen sollen. Die erste Spalte ist das Attribut Bestellnummer, die zweite besteht aus der Summe der einzelnen Bestellnummern, die in der Tabelle Bestellpositionen enthalten sind – in SQL lautet dies also: COUNT(bestellnummer). Diese Spalte erhält den Alias Anzahl. Zum Schluss wird die VIEW nach Bestellnummer gruppiert.

Hinweis zur GROUP BY Anweisung

Wie der Namen bereits ahnen lässt, kann man mit einer GROUP BY Anweisung ausgewählte Daten nach Attributen gruppieren. Gruppierungen sind vor allem in Kombination mit einer *Aggregatfunktionen (eine Zusammenlagerung von Daten und Objekten)* interessant. Einige von ihnen wurden bereits vorgestellt: COUNT, MIN, MAX, SUM, AVG, aber auch bisher nicht verwendete wie: GROUPING, FIRST, LAST, MEDIAN und viele Weitere. Abgesehen von einer Aggregatfunktion ist ein GROUP BY auch in Kombination mit einem JOIN zwingend zu empfehlen.

Aufgabe 5.2. `SELECT * FROM <viewname>`

Lassen Sie sich mit `SELECT` die Tupel der drei `VIEWS` ausgeben.

```
SELECT * FROM AArtikel
```

	ARTIKELNUMMER	ARTIKELNAME	PRODUKTGRUPPE	VERKAUFSPREIS	ABC	LAGERBESTAND
1	14460	James Audrey TikTok	Herrenuhren	149	A	15
2	12440	James Audrey Business Clock	Herrenuhren	349	A	25
3	12450	James Audrey Man Fitness	Herrenuhren	149	A	35
4	12456	James Audrey Chrono One	Herrenuhren	249	A	20
5	12459	James Audrey Big Data	Herrenuhren	149	A	30

```
SELECT * FROM NamenBeziehung
```

	NAME	ARTIKELNAME
1	Schmuck-und-Herrenartikel	James Audrey Business Clock
2	Your Brends Herrenarmbänder	James Audrey Cameron Bends
3	Schmuck-und-Herrenartikel	James Audrey Cameron Bends
4	Schmuck-und-Accessoires GbR	James Audrey Cameron Bends
5	Clock-Work GmbH	James Audrey Big Data
6	Clock-Work GmbH	James Audrey Big Data Mrs
7	Schmuck-und-Accessoires GbR	James Audrey Lady Fitness
8	Günes-Gözlügü AG	James Audrey Mrs Audreys Window
9	Günes-Gözlügü AG	James Audreys Window
10	Clock-Work GmbH	James Audrey Chrono One
11	Schmuck-und-Herrenartikel	James Audrey Man Fitness
12	Clock-Work GmbH	James Audrey TikTok
13	Clock-Work GmbH	James Audrey Mrs Audreys Chrono One

```
SELECT * FROM AnzahlBestellPos
```

	BESTELLNUMMER	ANZAHL
1	121226	3
2	121227	1
3	121228	1
4	121229	1
5	121230	6
6	121231	3
7	121233	1
8	121234	2

Aufgabe 5.3. INSERT, UPDATE, DELETE

Sind die Befehle INSERT, UPDATE und DELETE für diese VIEWs möglich? Stellen Sie das Ergebnis (ja / nein) tabellarisch dar.

Befehle für die Testfälle der VIEWs		
AArtikel	**IN.**	`INSERT INTO AArtikel VALUES (10900, 'James Audrey Luxus-Watch', 'Herrenuhren', '349,00', 'A', 100);`
	UP.	`UPDATE AArtikel SET verkaufspreis = '300,00' WHERE artikelnummer = 10900;`
	DEL.	`DELETE FROM AArtikel WHERE artikelnummer = 10900;`
NamenBeziehung	**IN.**	`INSERT INTO NamenBeziehung VALUES ('Clock-Work GmbH', 'James Audrey Luxus-Watch');`
	UP.	`UPDATE NamenBeziehung SET name = 'Schmuck-und-Herrenartikel' WHERE artikelname = 'James Audrey TikTok';`
	DEL.	`DELETE NamenBeziehung WHERE artikelname = 'James Audrey TikTok';`
AnzahlBestellPos	**IN.**	`INSERT INTO AnzahlBestellPos VALUES (121234, 4);`
	UP.	`UPDATE AnzahlBestellPos SET Anzahl = 4 WHERE bestellnummer = 121234;`
	DEL.	`DELETE FROM AnzahlBestellPos WHERE bestellnummer = 121234;`

IN. = INSERT; **UP.** = UPDATE; **DEL.** = DELETE

VIEWs								
AArtikel			NamenBeziehung			AnzahlBestellPos		
IN.	UP.	DEL.	IN.	UP.	DEL.	IN.	UP.	DEL.
ja	ja	ja	nein	nein	nein	nein	nein	nein

Die Fehlermeldungen bei `NamenBeziehung` und `AnzahlBestellPos`:

`NamenBeziehung`

INSERT *„Kann keine Spalte, die einer Basistabelle zugeordnet wird, verändern“*
UPDATE *„Datenmanipulationsoperation auf dieser View nicht zulässig“*
DELETE *„Datenmanipulationsoperation auf dieser View nicht zulässig“*

`AnzahlBestellPos`

INSERT *„Virtuelle Spalte hier nicht zulässig“*
UPDATE *„Datenmanipulationsoperation auf dieser View nicht zulässig“*
DELETE *„Datenmanipulationsoperation auf dieser View nicht zulässig“*

Hinweis zur Änderbarkeit von `VIEW`s

Wenn man die `INSERT`, `UPDATE` und `DELETE` Befehle für die `VIEW`s `AArtikel`, `NamenBeziehung` und `AnzahlBestellPos` betrachtet, kommt relativ schnell die Frage auf, warum die Veränderungen nur für die `VIEW` `AArtikel` zulässig sind. Die Begründung ist leichter, als sie auf Anhieb aussehen mag: `NamenBeziehung` und `AnzahlBestellPos` erfüllen nicht alle erforderlichen Bedingungen, die eine Veränderung von `VIEW`s zulassen.

Bei der `VIEW NamenBeziehung` ist weder `INSERT`, `UPDATE` noch `DELETE` möglich, da die `FROM`-Komponente mehr als einen Relationsnamen enthält. Um den Lieferanten- und den zugehörigen Artikelnamen ausgeben zu lassen, muss die `VIEW` auf drei Relationen zugreifen: `Artikel`, `Konditionen` und `Lieferanten`.

Auch bei der `VIEW AnzahlBestellPos` sind keine Veränderungen möglich, obwohl hier nur auf eine Relation zugegriffen wird. Hier wird aber gegen eine andere Regel verstoßen: Veränderungen sind nur dann möglich, wenn keine berechneten Spalten in der `VIEW` enthalten sind. Jedoch wird mit `COUNT(bestellnummer)` eine Berechnung vorgenommen.

Weitere Regeln stehen im IN1 Skript von Herr Prof. Dr. W. Gerken (Kapitel 3, Folie 90)

Aufgabe 5.4. Die Generalisierung

Die Aufgabe 5.4. wird aus einer reinen Erläuterung mit visueller Unterstützung in Form von Ausgabeergebnissen bestehen. Aus diesem Grund wurde die Aufgabenstellung in *a.* bis *f.* zerlegt, damit jede Unteraufgabe einen kurzen, erläuternden Text erhalten kann, um den Bearbeitungsverlauf leichter nachvollziehen zu können. Am Ende der gesamten Aufgabe werden alle SQL-Statements dokumentiert vorzufinden sein. Für eine erleichterte Orientierung verweisen innerhalb der Erläuterungen kleine Zahlen – wie zum Beispiel `(1)` – auf den entsprechenden SQL-Code, der rechtsbündig dieselbe kleine Zahl im Anschluss erhält.

Sie haben jetzt u.a. die Kunden- und Lieferantenrelation. Beide enthalten gemeinsame Attribute wie `Name`, `Straße`, `PLZ` *etc. ...*

a. Erstellen Sie eine neue Tabelle `Geschäftspartner`, die nur diese gemeinsamen Attribute enthält und zuzüglich das Schlüsselattribut `Partnernummer`.

Es kann möglich sein, dass man für die Aufgabe 5.4.*a.* zunächst einige Veränderungen an der Datenbank vornehmen muss, bevor man sie realisieren kann. Zum einen kommen explizit die Attributbezeichnungen in Frage, wie das Ändern von `Lieferantenname` in `Name (2)`. Zum anderen könnte ein neues Attribut `Name` für Vor- und Nachname zur Kundentabelle hinzufügt werden müssen `(3)`, um anschließend die Attribute `Vorname` und `Nachname` in das neue Attribut `Name` zu überführen `(4)`. Wenn Vor- und Nachname in einem Attribut `Name` zusammengefasst werden, sollte man sich gegen `NULL`-Werte entscheiden.

Sollten jedoch alle Voraussetzungen erfüllt sein, dann kann man im nächsten Schritt die neue Relation `Geschäftspartner` mit einem `CREATE TABLE` Befehl erzeugen `(5)`.

b. Speichern Sie die gemeinsamen Attribute von Kunden und Lieferanten in der neuen Geschäftspartnertabelle ab.

Für die Umsetzung der Aufgabe 5.4.*b.* wurde die vorgeschlagene `INSERT`-Variante aus der Aufgabenstellung von Prof. Dr. W. Gerken verwendet. Hierfür musste zunächst die Sequence `qp_seq` mit einem `CREATE SEQUENCE` Befehl `(6)` erstellt werden. Anschließend wurden zwei `INSERT` Befehle formuliert, die alle Attributwerte aus der Kunden- `(7)` und Lieferantentabelle `(8)` in die neu angelegte Tabelle `Geschäftspartner` übernommen haben. Insgesamt waren vier von fünf Attribute betroffen: `Name`, `Straße`, `PLZ` und `Ort`. Jedes Tupel hat in `Geschäftspartner` dank der Sequence automatisch ein fünftes, fortlaufendes Attribut `Partnernummer` erhalten.

c. Jetzt können Sie in der Kunden- und Lieferantentabelle einen Foreign Key auf `Geschäftspartner` hinzufügen, die Kunden- und Geschäftspartnerdaten über diese FK's dem richtigen Geschäftspartner zuordnen, die gemeinsamen Attribute löschen und die Tabellennamen in `Kundendaten` und `Lieferantendaten` ändern.

Die Aufgabe 5.4.*c.* war aufgrund der manuellen Durchführung und das selbstständige Zuordnen der automatisch zugewiesenen Partnernummern von `Geschäftspartner` in `Kunden` bzw. `Lieferanten` etwas mühselig; aufgrund der verhältnismäßig geringen Datenmenge und Anzahl an Kunden und Lieferanten in der Datenbank jedoch machbar.

Zuerst benötigte man in der Relation `Kunden` und `Lieferanten` ein geeignetes Attribut für den Foreign Key. Also wurde kurzerhand das Attribut `Partnernummer` jeweils mit einem `ALTER TABLE` Befehl zur Kunden- `(9)` und Lieferantentabelle `(10)` hinzugefügt. Damit im weiteren Verlauf noch immer die Datenintegrität gewährleistet sein kann, wurde bereits jetzt die `Partnernummer` für die Kunden- `(11)` und die Lieferantentabelle `(12)` als Foreign Key festgelegt.

Im dritten Schritt wurden alle Partnernummern manuell mit `UPDATE` Befehlen `(13)` jedem Kunden und Lieferanten zugeordnet, die sie von

der Sequence qp_seq in Geschäftspartner erhalten hatten. Insgesamt waren es 23 UPDATE Befehle, von denen achtzehn für die Kunden und fünf für die Lieferanten zur Buche kamen.

Nachdem nun jeder Kunde und Lieferant seine Partnernummer hatte, konnte man bedenkenlos die doppelten Attribute aus den Relationen Kunden (14) und Lieferanten (15) löschen. Dies war mit zwei einfachen ALTER TABLE und DROP Anweisungen schnell erledigt.

Zum Abschluss wurden die Relationen Kunden (16) und Lieferanten (17) umbenannt. Da die Attribute Kundennummer und Lieferantennummer auch gelöscht wurden, mussten sie in den relevanten Relationen Bestellungen (18) und Konditionen (19) in Partnernummer umbenannt und ebenfalls mit UPDATE Befehlen (20) aktualisiert werden.

SELECT * FROM GESCHÄFTSPARTNER

	PARTNERNUMMER	NAME	STRAßE	PLZ	ORT
1	41	Rick Grimes	Walking-Road 23	01934	Dead Ville
2	42	Carl Grimes	Walking-Road 23	01934	Dead Ville
3	43	Hershel Greene	Farm im Grünen 101	22115	Farmers Hood
4	44	Daryl Dixon	Merler-Landstraße 20	22111	Philadelphia
5	45	Carol Paletier	Klosterweg 13	20224	Atlanta
6	46	Abraham Ford	Karottenallee 4	30821	Washington D.C.
7	47	Walter White	Erlenmeyer-Straße 12a	42351	Albuquerque
8	48	Skyler White	Erlenmeyer-Straße 12a	42351	Albuquerque
9	49	Hank Shrader	An der Polizeiwache 80	39874	(null)
10	50	Jesse Pinkman	Bei-der-Schulbibliothek 12	39874	(null)
11	51	Gustavo Fring	Pollo Hermanos 15	38911	New Mexico
12	52	Saul Goodman	Bei-den-Anwälten 34	42351	Albuquerque
13	53	Sheldon Cooper	Big-Banger-Kreisel 42b	77345	Pasadena
14	54	Leonard Hofstadter	Big-Banger-Kreisel 42b	77345	Pasadena
15	55	Rajesh Koothrappali	Universitätsallee 69	78090	(null)
16	56	Howard Wolowitz	Neil-Armstrong-Platz 8	76548	Altadena
17	57	Amy Farrah Fowler	Charles-Darwin-Weg 112	77355	Pasadena
18	58	Bernadette Rostenkowski	Lamarck-Straße 2	78090	(null)
19	59	Günes-Gözlügü AG	Sonnenallee 112	77345	Pasadena
20	60	Clock-Work GmbH	Additionsstraße 16	42351	Albuquerque
21	61	Schmuck-und-Accessoires GbR	Ticari Weg 24a	01934	(null)
22	62	Schmuck-und-Herrenartikel	Ticari Weg 24b	01934	(null)
23	63	Your Brends Herrenarmbänder	Schlangenhügel 10	20224	Atlanta

d. Es sollten noch mindestens zwei spezifische Kunden- und Lieferantenattribute übrig bleiben. Wenn Sie die nicht haben, denken Sie sich welche aus. Damit haben Sie das OO-Konstrukt „Generalisierung / Spezialisierung" realisiert.

In der Relation `Lieferantendaten` waren letzten Endes noch die beiden Attribute `Partnernummer` und `Telefonnummer` übrig geblieben, daher brauchte man hier keine Veränderungen vorzunehmen.

In der Relation `Kundendaten` war nur das Attribut `Partnernummer` übrig, daher musste man dieser Relation mit einem `ALTER TABLE` Befehl `(21)` mindestens ein weiteres Attribut *(hier entschied man sich für `Geburtsdatum`)* hinzufügen, damit die minimale Menge an Attributen auch hier erreicht war. Im Endeffekt wäre jedes kundenspezifische Attribut möglich gewesen, außer `Telefonnummer`. Würde man sich für genau dieses Attribut entscheiden, dann hätte es genauso gut wieder in der Geschäftspartnertabelle Platz gefunden, als gemeinsames Attribut mit den `Lieferanten`.

e. Aber nur fast: Generalisierung / Spezialisierung ist eine (0,1)-(1,1)-Beziehung. Sorgen Sie also dafür, dass jedes Kundendaten-Tupel bzw. Lieferantendaten-Tupel zu genau einem Geschäftspartner-Tupel in Beziehung steht!

Für eine *(0,1)-(1,1)-Beziehung* müssen die Partnernummer-Foreign-Keys aus der Tabelle `Kundendaten` (22) bzw. `Lieferantendaten` (23) zusätzlich mit einem `UNIQUE CONSTRAINT` versehen werden *(ein zusammengesetzter Primary Key wäre ebenfalls möglich)*. Durch diese weitere Bedingung wird gegeben, dass jede Partnernummer nur ein einziges Mal in die Kunden- bzw. Lieferantendatenrelation eingegeben werden kann. Folgerichtig würde genau dann eine *(0,1)-(1,1)-Beziehung* bestehen.

```
SELECT * FROM KUNDENDATEN
```

	PARTNERNUMMER	GEBURTSDATUM
1	41	(null)
2	42	(null)
3	43	(null)
4	44	(null)
5	45	(null)
6	46	(null)
7	47	(null)
8	48	(null)
9	49	(null)
10	50	(null)
11	51	(null)
12	52	(null)
13	53	(null)
14	54	(null)
15	55	(null)
16	56	(null)
17	57	(null)
18	58	(null)

```
SELECT * FROM LIEFERANTENDATEN
```

	PARTNERNUMMER	TELEFONNUMMER
1	59	017612345612
2	60	(null)
3	61	017961212345
4	62	(null)
5	63	017661255882

f. Nur laufen Ihre bisherigen Queries, die `Kunden` und `Lieferanten` verwenden, jetzt leider nicht mehr! Das können Sie aber beheben, indem Sie zwei `VIEWs` `Kunden` und `Lieferanten` anlegen, die Ihre ursprünglichen Daten enthalten. Führen Sie zum Test die Query 2.h. aus der Praktikumsaufgabe 3 erneut aus.

Für die Aufgabe 5.4.*f.* wurden zwei weitere `VIEWs` `Lieferanten (24)` und `Kunden (25)` mit einem `CREATE VIEW` Befehl erstellt, die exakt dieselben Informationen beinhalten, wie die ursprünglichen Relationen `Kunden` und `Lieferanten`. Für die beiden `VIEWs` wurden die Attribute `Geburtsdatum` *(in der `Kunden-View`)* und `Telefonnummer` *(in der `Lieferanten-View`)* berücksichtigt. Allerdings wurde für diese Aufgabe darauf verzichtet, für jeden einzelnen Kunden ein eigenes Geburtsdatum auszudenken.

Die Query 2.*h.* `(26)` aus der Praktikumsaufgabe 3 liefert nach minimaler Modifikation von `Kundennummer` in `Partnernummer` nun dasselbe Abfrageergebnis, wie in der ersten Maiwoche.

```
SELECT * FROM KUNDEN
```

	PARTNERNUMMER	NAME	STRAßE	PLZ	ORT	GEBURTSDATUM
1	41	Rick Grimes	Walking-Road 23	01934	Dead Ville	(null)
2	42	Carl Grimes	Walking-Road 23	01934	Dead Ville	(null)
3	43	Hershel Greene	Farm im Grünen 101	22115	Farmers Hood	(null)
4	44	Daryl Dixon	Merler-Landstraße 20	22111	Philadelphia	(null)
5	45	Carol Paletier	Klosterweg 13	20224	Atlanta	(null)
6	46	Abraham Ford	Karottenallee 4	30821	Washington D.C.	(null)
7	47	Walter White	Erlenmeyer-Straße 12a	42351	Albuquerque	(null)
8	48	Skyler White	Erlenmeyer-Straße 12a	42351	Albuquerque	(null)
9	49	Hank Shrader	An der Polizeiwache 80	39874	(null)	(null)
10	50	Jesse Pinkman	Bei-der-Schulbibliothek 12	39874	(null)	(null)
11	51	Gustavo Fring	Pollo Hermanos 15	38911	New Mexico	(null)
12	52	Saul Goodman	Bei-den-Anwälten 34	42351	Albuquerque	(null)
13	53	Sheldon Cooper	Big-Banger-Kreisel 42b	77345	Pasadena	(null)
14	54	Leonard Hofstadter	Big-Banger-Kreisel 42b	77345	Pasadena	(null)
15	55	Rajesh Koothrappali	Universitätsallee 69	78090	(null)	(null)
16	56	Howard Wolowitz	Neil-Armstrong-Platz 8	76548	Altadena	(null)
17	57	Amy Farrah Fowler	Charles-Darwin-Weg 112	77355	Pasadena	(null)
18	58	Bernadette Rostenkowski	Lamarck-Straße 2	78090	(null)	(null)

```
SELECT * FROM LIEFERANTEN
```

	PARTNERNUMMER	NAME	STRAßE	PLZ	ORT	TELEFONNUMMER
1	59	Günes-Gözlügü AG	Sonnenallee 112	77345	Pasadena	017612345612
2	60	Clock-Work GmbH	Additionsstraße 16	42351	Albuquerque	(null)
3	61	Schmuck-und-Accessoires GbR	Ticari Weg 24a	01934	(null)	017961212345
4	62	Schmuck-und-Herrenartikel	Ticari Weg 24b	01934	(null)	(null)
5	63	Your Brends Herrenarmbänder	Schlangenhügel 10	20224	Atlanta	017661255882

Ausgabeergebnis der Query 2.*h.* aus der Praktikumsaufgabe 3

	ARTIKELNAME	NAME
1	James Audrey Big Data	Gustavo Fring
2	James Audrey Big Data Mrs	Gustavo Fring
3	James Audrey Cameron Bends	Gustavo Fring
4	James Audrey Chrono One	Gustavo Fring
5	James Audrey Mrs Audreys Window	Gustavo Fring
6	James Audreys Window	Gustavo Fring

Hinweis zur Datenintegrität

Schlüsselintegrität

Unter dem Begriff der *Integrität* oder *Konsistenz* von Daten, versteht man die Widerspruchsfreiheit der Datenbestände. Eine Datenbank ist konsistent, wenn die gespeicherten Daten fehlerfrei erfasst sind und den gewünschten Informationsgehalt korrekt wiedergeben. Die *Datenintegrität* ist verletzt, wenn Mehrdeutige oder widersprüchliche Sachverhalte auftreten.

Nach Definition sind Relationen Mengen von Tupel, die allein durch ihre Werte unterschieden werden. Die Tupel müssen folgerichtig eindeutig identifizierbar sein, was durch *eindeutige Schlüssel* ermöglicht wird. Im relationalen Datenbankmodell wird dies durch die *Schlüsselkandidaten* bzw. *Primärschlüssel* gewährleistet *(siehe Näheres zu den Schlüsseln auf Seite 36 und 37).*

Gegenstands-Integritätsbedingung

Sie folgt direkt aus der *Schlüsselintegritätsbedingung* und besagt, dass kein Primärschlüsselwert ohne Wert existieren darf *(also keine zulässigen `NULL`-Werte oder { } für Primärschlüssel)*. Falls Schlüssel ohne Wert möglich wären, könnten mehrere Tupel `NULL` als Schlüsselwert besitzen. Dies widerspricht der *Schlüsselintegrität*, da diese Tupel nicht mehr eindeutig identifizierbar wären.

Referentielle Integrität

Beziehungen zwischen zwei Relationen werden immer so hergestellt, dass ein Primärschlüssel der einen Relation in die zweite bzw. derselben Relation als Fremdschlüssel aufgenommen werden kann. Dies verlangt, dass aktuelle Fremdschlüsselwerte sich immer nur auf Primärschlüsselwerte von existierenden Tupel beziehen können. *(siehe Näheres zu den Beziehungen auf Seite 21).*

Aufgabe 5.5. Aktualisierung des Glossars aus 2.3.

Bitte erstellen Sie eine aktualisierte Version Ihres Glossars.

RELATION	ATTRIBUT	BESCHREIBUNG	KEY
Kundendaten		**Daten der Kunden**	
	`Partnernummer`	Partnernummer des Kunden	P-FK
	`Geburtsdatum`	Geburtsdatum des Kunden	
Bestellungen		**Daten der Bestellungen**	
	`Bestellnummer`	Bestellnummer	PK
	`Partnernummer`	Partnernummer des Kunden	FK
	`Bestelldatum`	Datum des Bestellauftrags	
	`Status`	Bearbeitungsstatus	
Bestellpositionen		**Daten über eine Bestellung**	
	`Bestellnummer` `Artikelnummer`	Bestellposition einer Bestellung Artikel einer Bestellung	P-FK
	`Menge`	Bestellmenge einer Bestellung	
Artikel		**Daten über die Artikel**	
	`Artikelnummer`	Artikelnummer	PK
	`Artikelname`	Bezeichnung des Artikels	
	`Produktgruppe`	Produktgruppe des Artikels	
	`Verkaufspreis`	Verkaufspreis des Artikels	
	`ABCKlassifikation`	Klassifizierung nach A, B, C	
	`Lagerbestand`	Artikelmenge im Lager	
Konditionen		**Daten über die Konditionen**	
	`Partnernummer` `Artikelnummer`	Partnernummer des Lieferanten Artikel des Lieferanten	P-FK
	`Einkaufspreis`	Bezugspreis des Artikels	
Lieferantendaten		**Daten der Lieferanten**	
	`Partnernummer`	Partnernummer des Lieferanten	P-FK
	`Telefonnummer`	Name des Lieferanten	
Geschäftspartner		**Daten Kunden/Lieferanten**	
	`Partnernummer`	Partnernummer für beide	PK
	`Name`	Name des Kunden/Lieferanten	
	`Straße`	Straße des Kunden/Lieferanten	
	`PLZ`	PLZ des Kunden/Lieferanten	
	`Ort`	Ort des Kunden/Lieferanten	

P-FK = Primary und Foreign Key

SQL-Befehle-Katalog für die Aufgaben 5.1. und 5.4.

Alternativbefehl für die Praktikumsaufgabe 5.1.

`CREATE VIEW` Befehl, bei dem in der `VIEW AArtikel`, auch die zugehörigen Lieferantennamen ausgegeben werden:

```
1     CREATE VIEW AArtikel
2       AS SELECT ARTIKEL.*, L.name
3       FROM ARTIKEL A, LIEFERANTEN L, KONDITIONEN KON
4         WHERE L.partnernummer = KON.partnernummer
5         AND KON.artikelnummer = A.artikelnummer
6         AND abcklassifikation = 'A';
```

(1)

```
SELECT * FROM AArtikel
```

	ARTIKELNUMMER	ARTIKELNAME	PRODUKTGRUPPE	VERKAUFSPREIS	ABC	LAGERBESTAND	LIEFERANTENNAME
1	14460	James Audrey TikTok	Herrenuhren	149	A	15	Clock-Work GmbH
2	12440	James Audrey Business Clock	Herrenuhren	349	A	25	Schmuck-und-Herrenartikel
3	12450	James Audrey Man Fitness	Herrenuhren	149	A	35	Schmuck-und-Herrenartikel
4	12456	James Audrey Chrono One	Herrenuhren	249	A	20	Clock-Work GmbH
5	12459	James Audrey Big Data	Herrenuhren	149	A	30	Clock-Work GmbH

Alle erwähnten SQL-Befehle für die Praktikumsaufgabe 5.4.

Umbenennung von `lieferantenname` in `name`:

```
1     ALTER TABLE LIEFERANTEN
2       RENAME COLUMN lieferantenname TO name;
```

(2)

Hinzufügen von `name` in `Kunden`:

```
1     ALTER TABLE KUNDEN
2       ADD name VARCHAR(50);
```

(3)

Überführung von vorname und nachname in name:

```
1    UPDATE KUNDEN
2      SET name = 'Rick Grimes'
3      WHERE kundennummer = 11111;
4    UPDATE KUNDEN
5      SET name = 'Daryl Dixon'
6      WHERE kundennummer = 141424;
7    UPDATE KUNDEN
8      SET name = 'Sheldon Cooper'
9      WHERE kundennummer = 324567;
10   UPDATE KUNDEN
11     SET name = 'Howard Wolowitz'
12     WHERE kundennummer = 3987634;
```

(4)

Der UPDATE Befehl sieht bei den anderen *vierzehn* Kunden – abgesehen von den Attributwerten für Name und Kundennummer – identisch wie die vier oben stehenden Beispiele aus. Einfachheitshalber wird auf die endlos lang erscheinende Wiederholungsreihe verzichtet.

CREATE TABLE Befehl zur Erstellung von GESCHÄFTSPARTNER:

```
1    CREATE TABLE GESCHÄFTSPARTNER (
2      partnernummer NUMBER(6),
3      name VARCHAR2(30),
4      straße VARCHAR2(30) NOT NULL,
5      plz VARCHAR2(5) NOT NULL,
6      ort VARCHAR2(45),
7      CONSTRAINT Partner_PK
8        PRIMARY KEY (partnernummer)
9    );
```

(5)

Erstellung der Sequence qp_seq:

```
1    CREATE SEQUENCE qp_seq;
```

(6)

Fortlaufende partnernummer in GESCHÄFTSPARTNER:

```
1    INSERT INTO GESCHÄFTSPARTNER
2      (partnernummer, name, straße, plz, ort)
3      (SELECT
4      qp_seq.nextval, name, straße, plz, ort
5    FROM KUNDEN);
```

(7)

```
18 Zeilen eingefügt.
```

```
1    INSERT INTO GESCHÄFTSPARTNER
2      (partnernummer, name, straße, plz, ort)
3      (SELECT
4      qp_seq.nextval, name, straße, plz, ort
5    FROM LIEFERANTEN);
```

(8)

```
5 Zeilen eingefügt.
```

Hinzufügen der partnernummer in KUNDEN:

```
1    ALTER TABLE KUNDEN
2      ADD partnernummer NUMBER(6);
```

(9)

Hinzufügen der `partnernummer` in `LIEFERANTEN`:

```
1     ALTER TABLE LIEFERANTEN
2       ADD partnernummer NUMBER(6);
```

(10)

Hinzufügen des Foreign Keys in `KUNDEN` und `LIEFERANTEN`:

```
1     ALTER TABLE KUNDEN
2       ADD FOREIGN KEY (partnernummer)
3       REFERENCES GESCHÄFTSPARTNER (partnernummer);
```

(11)

```
1     ALTER TABLE LIEFERANTEN
2       ADD FOREIGN KEY (partnernummer)
3       REFERENCES GESCHÄFTSPARTNER (partnernummer);
```

(12)

Hinzufügen der Kunden- und Lieferantenpartnernummer:

```
1     UPDATE LIEFERANTEN
2       SET partnernummer = 23
3       WHERE name = 'Your Brends Herrenarmbänder';
4     UPDATE LIEFERANTEN
5       SET partnernummer = 22
6       WHERE name = 'Schmuck-und-Herrenartikel';
```

(13)

… und drei weitere Befehle

```
7     UPDATE KUNDEN
8       SET partnernummer = 41
9       WHERE name = 'Rick Grimes';
10    UPDATE KUNDEN
11      SET partnernummer = 42
12      WHERE name = 'Carl Grimes';
```

```
13    UPDATE KUNDEN
14      SET partnernummer = 43
15      WHERE name = 'Hershel Greene';
16    UPDATE KUNDEN
17      SET partnernummer = 44
18      WHERE name = 'Daryl Dixon';
```

(13)

… *und vierzehn weitere Befehle*

Löschvorgang der doppelten Attribute aus der Kunden- und Lieferantentabelle *(mit* `kunden-` *und* `lieferantennummer`*):*

```
1     ALTER TABLE KUNDEN
2       DROP (kundennummer, name,
3             straße, plz, ort);
```

(14)

```
1     ALTER TABLE LIEFERANTEN
2       DROP (lieferantennummer, name,
3             straße, plz, ort);
```

(15)

Umbenennung von `KUNDEN` und `LIEFERANTEN`:

```
1     ALTER TABLE KUNDEN
2       RENAME TO KUNDENDATEN;
```

(16)

```
1     ALTER TABLE LIEFERANTEN
2       RENAME TO LIEFERANTENDATEN;
```

(17)

Umbenennung der `kunden-` und `lieferantennummer`:

```
1    ALTER TABLE BESTELLUNGEN
2      RENAME COLUMN kundennummer
3      TO partnernummer;
```

(18)

```
1    ALTER TABLE KONDITIONEN
2      RENAME COLUMN lieferantennummer
3      TO partnernummer;
```

(19)

Anpassungen für `partnernummer` in `BESTELLUNGEN`:

```
1    UPDATE BESTELLUNGEN
2      SET partnernummer = 41
3      WHERE bestellnummer = 121234;
4    UPDATE BESTELLUNGEN
5      SET partnernummer = 46
6      WHERE bestellnummer = 121226;
7    UPDATE BESTELLUNGEN
8      SET partnernummer = 56
9      WHERE bestellnummer = 121227;
10   UPDATE BESTELLUNGEN
11     SET partnernummer = 49
12     WHERE bestellnummer = 121228;
```

(20)

… *und fünf weitere Befehle*

Anpassungen für partnernummer in KONDITIONEN:

```
13   UPDATE KONDITIONEN
14     SET partnernummer = 60
15     WHERE bestellnummer = 12456;
16   UPDATE KONDITIONEN
17     SET partnernummer = 61
18     WHERE bestellnummer = 13670;
19   UPDATE BESTELLUNGEN
20     SET partnernummer = 63
21     WHERE einkaufspreis = '22';
23   UPDATE BESTELLUNGEN
24     SET partnernummer = 62
25     WHERE einkaufspreis = '25,45';
26   UPDATE BESTELLUNGEN
27     SET partnernummer = 61
28     WHERE einkaufspreis = '19,99';
```

(20)

… *und acht weitere Befehle*

Hinzufügen des Attributs geburtsdatum in KUNDENDATEN:

```
1    ALTER TABLE KUNDENDATEN
2      ADD geburtsdatum DATE;
```

(21)

Ermöglichung der *(0,1)-(1,1)-Beziehung* bei Kundendaten:

```
1    ALTER TABLE KUNDENDATEN
2      MODIFY partnernummer UNIQUE;
```

(22)

Ermöglichung der *(0,1)-(1,1)-Beziehung* bei `Lieferantendaten`:

```
1    ALTER TABLE LIEFERANTENDATEN
2      MODIFY partnernummer UNIQUE;
```

(23)

Erstellung der beiden `VIEW`s `KUNDEN` und `LIEFERANTEN`:

```
1    CREATE VIEW LIEFERANTEN
2      AS SELECT GESCHÄFTSPARTNER.*, LD.telefonnummer
3      FROM GESCHÄFTSPARTNER G, LIEFERANTENDATEN LD
4        WHERE  G.partnernummer = LD.partnernummer;
```

(24)

```
1    CREATE VIEW KUNDEN
2      AS SELECT GESCHÄFTSPARTNER.*, KD.geburtsdatum
3      FROM GESCHÄFTSPARTNER G, KUNDENDATEN KD
4        WHERE  G.partnernummer = KD.partnernummer;
```

(25)

Angepasste Abfrage *2.h.* aus der Praktikumsaufgabe 3:

```
1    SELECT artikelname, name
2    FROM KUNDEN K, BESTELLUNGEN B,
3    BESTELLPOSITIONEN BPOS, ARTIKEL A
4      WHERE K.partnernummer = B.partnernummer
5      AND B.bestellnummer = BPOS.bestellnummer
6      AND BPOS.artikelnummer = A.artikelnummer
7      AND name = '%Gustavo%'
8        GROUP BY artikelname, name
9        ORDER BY artikelname ASC;
```

(26)

```
SELECT PRAKTIKUM FROM INFORMATIONSSYSTEME
    WHERE zusatzmaterial =
    'Anhang'
    AND autor = 'Steven Illg';
```

Übungsblätter mit Lösungen

Übungsblatt Nr. 1

1. Aufgabe

Gegeben seien die folgenden Tabellen
`Student` *(MatrNr, Name, StGang, StudBeginn, Ort),*
`Fach` *(FachNr, StGang, Bezeichnung)* und
`Klausur` *(MatrNr, FachNr, Punkte, Datum).*

a. Formulieren Sie den `CREATE TABLE` Befehl zum Erzeugen der Tabelle `Klausur`.

```
CREATE TABLE KLAUSUR (
  MatrNr NUMBER(8),
  FachNr NUMBER(4),
  Punkte NUMBER(2),
  CHECK (Punkte IN (1,2,3,4,5,6,7,8,9,10,11,
                    12,13,14,15)),
  Datum DATE,
  CONSTRAINT KLAUSUR_PK
    PRIMARY KEY (MatrNr, FachNr)
  CONSTRAINT KLAUSUR_FK
    FOREIGN KEY (MatrNr)
    REFERENCES STUDENT (MatrNr)
    FOREIGN KEY (FachNR)
    REFERENCES FACH (FachNr)
);
```

b. Formulieren Sie mit SQL: Welche Studierenden *(Name)* wohnen nicht in Hamburg und studieren im Studiengang `'AI'`?

```
SELECT name FROM STUDENT
  WHERE StGang = 'AI'
  AND ort NOT = 'Hamburg';
```

c. Wie viele Studierende studieren im Studiengang `'WI'`?

```
SELECT COUNT(*) FROM STUDENT
  WHERE StGang = 'WI';
```

d. Welche Studierende *(MatrikelNr)* haben am `'9.7.2013'` an der Klausur mit der `FachNr 47` teilgenommen?

```
SELECT MatrNr FROM KLAUSUR
  WHERE FachNr = 47
  AND Datum = '09.07.2013';
```

e. Wie heißen diese Studierenden?

```
SELECT name FROM STUDENT S, KLAUSUR K
  WHERE FachNr = 'DB'
    AND Datum = '09.07.2013'
    AND K.MatrNr = S.MatrNr;
```

2. Aufgabe

Formulieren Sie in der Relationsalgebra folgende Queries für die Tabellen aus Aufgabe 1.:

a. Die Namen der Studenten im Studiengang `'TI'` ausgeben.

```
(STUDENT [StGang = 'TI'] ) [name]
```

b. Welche `'AI'`-Studenten, die `2013` ihr Studium begonnen haben, haben die Klausur im Fach mit der Bezeichnung *„DB“ (für „Datenbanken“)* mit `13 Punkten` bestanden?

```
( (STUDENT [StGang='AI', StudBeginn=2013])
    [MatrNr=MatrNr] ( (KLAUSUR [Punkte=13])
    [FachNr=FachNr] (FACH [Bezeichnung='DB']) )
) [Name]
```

3. Aufgabe

Bei einem Datenbanksystem wird zwischen dem *externen, konzeptionellen* und *internen Modell* unterschieden. Welches Modell ist bei den folgenden Änderungen der Relationen

`Fakultät` *(FkNr, Bezeichnung, Dekan)* und
`Mitarbeiter` *(PersNr, Name, FkNr)*

jeweils betroffen – wenn überhaupt?

a. Eine Fakultät kann ab sofort mehrere Dekane haben.

   `Konzeptionelles Modell`

b. Für die Datensätze der Relation `Mitarbeiter` wird zusätzlich ein Index auf das Attribut `Name` festgelegt.

   `Internes Modell`

c. Benutzer Harry, der bisher nur auf die Mitarbeiter-Tabelle zugreifen konnte, möchte in seiner Datenansicht auch die Bezeichnung der Fakultäten für die Mitarbeiter sehen.

   `Externes Modell`

d. Der Dekan der Fakultät `'LS'` ist nicht mehr Mitarbeiter mit Personalnummer `4712`, sondern nun der Mitarbeiter `1567`.

   `Keines der drei Modelle`

Näheres steht im IN1 Skript von Herr Prof. Gerken (Kapitel 2, Folie 33-35)

4. Aufgabe

Gegeben seien die beiden folgenden Relationen

R (A	B	C	D)
1	3	8	5
1	4	7	3
2	5	7	9
3	1	6	1
3	3	7	1
5	6	3	9
7	2	5	8

und

S (E	F)
1	2
8	3
9	4
4	1
3	6
9	7
8	4

Welches Ergebnis liefern die Ausdrücke der Relationsalgebra:

a. `(R[C<E]S)[B,F]`

B	F
3	3
3	4
3	7
4	3
4	4
4	7
5	3
5	4
5	7
1	3
1	4
1	7
6	3
6	4
6	7
2	3
2	4
2	7

b. `((R[D<4])[B=E]S)[A,F]`

A	F
1	1
3	2
3	6

5. Aufgabe

Gegeben seien die Tabellen

`Student` *(MatrNr, Name, Fachbereich),*
`Klausur` *(MatrNr, FachNr, Semester, Note)* und
`Fach` *(FachNr, Bezeichnung)*

a. Wie lautet der Primärschlüssel der Tabelle Klausur?

Aufgrund der Wiederholungsmöglichkeit handelt es sich um einen zusammengesetzt PK aus `MatrNr`, `FachNr` und `Semester`.

b. Wie nennt man das Attribut `MatrNr` in `Klausur` in Bezug auf `Student` und das Attribut `FachNr` in Bezug auf `Fach`?

Fremdschlüssel *(bzw. Foreign Key)*

c. Was ist zu beachten, wenn man ein Klausur-Tupel speichert?

In `Student` muss die `MatrNr` und in `Fach` muss die `FachNr` bereits vorhanden sein.

d. Was passiert, wenn man ein Student- oder ein Fach-Tupel löscht?

Bei **`ON DELETE CASCADE`** werden die zugehörigen Klausur-Tupel mit gelöscht.

Bei **`ON DELETE RESTRICT`** `[`**`DEFAULT`**`]` wird das Löschen abgelehnt, falls noch zugehörige Klausur-Tupel vorhanden sind.

Übungsblatt Nr. 2

1. Aufgabe

Gegeben sei die Relation `ProfStud` *(ProfNr, MatrNr, Vorlesung, Skript, Abschluss)*. Es gelte:

i. Ein Student belege maximal eine Vorlesung je Professor.
ii. Ein Professor kann mehrere Vorlesungen halten.
iii. Eine Vorlesung kann von mehreren Professoren gleichzeitig angeboten werden.
iv. Zu jeder Vorlesung gibt es ein Skript, egal von welchem Professor die Vorlesung gehalten wird.
v. Ein Student kann nicht mehrere Abschlüsse *(z.B. Dipl.-Ing. und Bachelor of Science)* gleichzeitig anstreben.

a. Welches ist der Primärschlüssel dieser Relation?

Es existieren die funktionalen Abhängigkeiten:

(ProfNr, MatrNr) → *Vorlesung,*
MatrNr → *Abschluss und*
Vorlesung → *Skript.*

Der Primary Key ist also zusammengesetzt aus *(ProfNr, MatrNr)*.

b. Zeigen Sie, dass die Relation weder in zweiter, noch in dritter Normalform vorliegt.

Liegt nur in erster Normalform vor, da *MatrNr* → *Abschluss* keine volle funktionale Abhängigkeit vom Primary Key ist.

c. Ermitteln Sie eine Zerlegung, die in der dritten Normalform ist.

`Vorlesung` *(ProfNr, MatrNr, Vorlesung)*
`Skript` *(Vorlesung, Skript)*
`Abschlüsse` *(MatrNr, Abschluss)*

2. Aufgabe

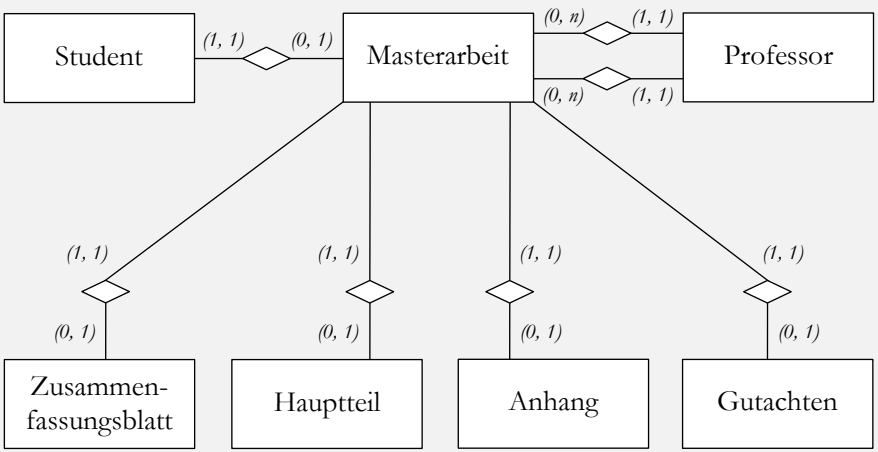
Gegeben sei die Relation `R` *(A, B, C, D, E, F)* mit den funktionalen Abhängigkeiten *(B, C)* → *A*, *(A, D)* → *E* und *E* → *F*. Füllen Sie die Relation mit Beispiel-Tupel, die diesen Abhängigkeiten entsprechen.

A	B	C	D	E	F
⇓	⇗	⇒	⇘	⇒	⇐
⇐	⇓	⇙	⇖	⇑	⇓
⇓	⇗	⇒	⇘	⇒	⇐
⇐	⇓	⇙	⇖	⇑	⇓
⇐	⇓	⇙	⇖	⇑	⇓
⇐	⇓	⇙	⇖	⇑	⇓

Attributwerte sind rein exemplarisch!

3. Aufgabe

Ein Student bearbeitet höchstens eine Masterarbeit, für die es einen Erst- und Zweitgutachter gibt. Eine Masterarbeit besteht aus einem Zusammenfassungsblatt, dem Hauptteil, dem Anhang und einem Gutachten. Erstellen Sie hierfür ein ERM mit *(min, max)*-Notation.



4. Aufgabe

Gegeben sei die folgende Tabelle `TAB`*:*

A	B	C	D
X	2	100	HH
X	1	101	KI
Y	2	100	HH
Y	3	101	KI
Y	2	102	KI

a. Wie lautet der Primärschlüssel dieser Tabelle?

Der Primärschlüssel ist *(A, C)*

b. In welcher Normalform liegt die Tabelle vor? Mit Begründung!

Da $C \rightarrow D$ gilt, liegt die Tabelle in erster Normalform vor.

c. Zerlegen Sie die Tabelle so, dass die dritte Normalform erfüllt ist. *Nur das Schema mit Tabellenbezeichnungen und Attributen angeben.*

$\texttt{TAB}_1$ *(A, B, C)*
$\texttt{TAB}_2$ *(C, D)*

5. Aufgabe

Es gebe die zwei Entitätstypen `Vereine` und `Personen`. Eine Person kann Mitglied in keinem oder auch mehreren Vereinen sein. Ein Verein muss mindestens ein Mitglied haben und kann natürlich auch mehrere Personen als Mitglieder haben.

a. Wie lautet das ERM in *(min, max)*-Notation?

b. Welche Tabellen entstehen, wenn Sie dieses ERM in eine relationale Datenbank abbilden? *Schreibweise: Tabellenbezeichnung (Attribut$_1$, Attribut$_2$, … Attribut$_N$) und den Primärschlüssel unterstreichen.*

`Vereine` *(VereinsNr, Name, …)*
`Personen` *(PersNr, Name, Vorname, …)*
`Mitglied` *(VereinsNr, PersNr, Mitglied_seit, …)*

Übungsklausur Informationssysteme I *vom 01.Juni 2015*

1. Aufgabe

Gegeben sind die folgenden Tabellen einer relationalen Datenbank

Bestellungen	
BestellNr	**Bestelldatum**
100	6.06.2012
101	6.06.2012
102	8.06.2012
103	8.06.2012
104	15.06.2012

Bestellpositionen		
BestellNr	**ArtNr**	**Menge**
101	91	2000
101	92	500
102	93	1000
102	91	1500
102	92	600
103	93	400

Wie lautet der SQL-Code für:

1.1. Zu welcher Bestellung gibt es keine Bestellpositionen?

```
SELECT BestellNr FROM Bestellungen
  WHERE BestellNr NOT IN (
    SELECT BestellNr
    FROM Bestellpositionen);
```

1.2. Wie viele Bestellungen gab es am 8.06.2012?

```
SELECT COUNT(*) FROM Bestellungen
  WHERE Bestelldatum = '8.06.2012'
```

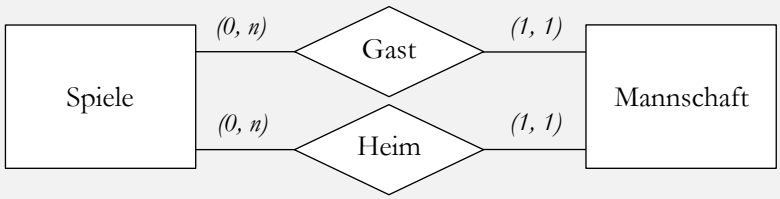
1.3. Wie lautet das Ergebnis des SQL-Befehls:

```
SELECT BestellNr, COUNT(*)
  FROM Bestellpositionen
    GROUP BY BestellNr;
```

BestellNr	COUNT(*)
101	2
102	3
103	1

2. Aufgabe

Gegeben sei das folgende ERM:



Primärschlüssel von *„Spiele"* ist *SpielNr* und von *„Mannschaft"* der *Kurzname*. Für jedes Spiel sind außerdem das Ergebnis *(jeweils Tore der Heim- und Gastmannschaft)* und das Spieldatum gespeichert. Pro Saison gibt es nur ein Spiel pro Paarung.

a. Formulieren Sie den `CREATE TABLE` Befehl für die Relation *„Spiele"*. Die Datentypen sollen geeignet gewählt werden.

```
CREATE TABLE Spiele (
  SpieleNr INTEGER,
  Gast CHAR(20) NOT NULL,
  Punkte_Gast NUMBER(2),
  Punkte_Heim NUMBER(2),
  Ergebnis VARCHAR(10),
  Spieldatum DATE,
  PRIMARY KEY SpielNr,
  FOREIGN KEY Gast REFERENCES Mannschaft,
  FOREIGN KEY Heim REFERENCES Mannschaft
);
```

b. Was können Primärschlüssel dieser Relation sein, wenn es keine *SpielNr* geben würde?

Schlüsselkandidaten:

`Heim, Gast,` *(+ `Saison`, falls Statistik über mehrere Jahre)*

3. Aufgabe

3.1. Welchen Zweck hat ein `TRIGGER`?

Ein `TRIGGER` wird bei Aufruf von `INSERT`-, `UPDATE`- oder `DELETE`-Befehlen automatisch aufgerufen und kann dadurch für die Einhaltung der Datenintegrität sorgen.

3.2. Finden Sie die im folgenden `TRIGGER` vorhandenen *Fehler (die Korrektur wurde im Text bereits in **BOLD** hinzugefügt)*

```
CREATE OR REPLACE TRIGGER Spiele_ok
  BEFORE INSERT ON Spiele
    FOR EACH ROW
    BEGIN
      IF:~~OLD~~NEW.Heim = :~~OLD~~NEW.Gast
      THEN RAISE_APPLICATION_ERROR
      (-20001, 'Heim = Gast geht nicht!');
      END IF;
    END;
```

Hinweis:

DER PL/SQL-Befehl `RAISE_APPLICATION_ER-ROR...` gibt eine Fehlermeldung aus und führt zum Transaktionsabbruch.

4. Aufgabe

Gegeben seien die drei folgenden Relationen und ihre funktionalen Abhängigkeiten:

R	*(A, B, C, D, E)*	*(A, B)* → *C; B* → *D; D* → *E*
S	*(D, E, F)*	*D* → *E; E* → *F*
T	*(G, H, I)*	*G* → *H; G* → *I*

a. Wie lauten die Primärschlüssel dieser Relationen?

Die Primärschlüssel sind: R = *(A, B);* S = *D* und T = *G*

b. In welcher Normalform liegen diese Relationen vor? *(Begründung)*

R: Liegt nur in der 1. Normalform vor, da keine volle funktionale Abhängigkeit unter den Attributen besteht.

S: Liegt nur in der 2. Normalform vor, da eine transitive Abhängigkeit durch das Attribut *E* besteht.

T: Liegt in der 3. Normalform vor.

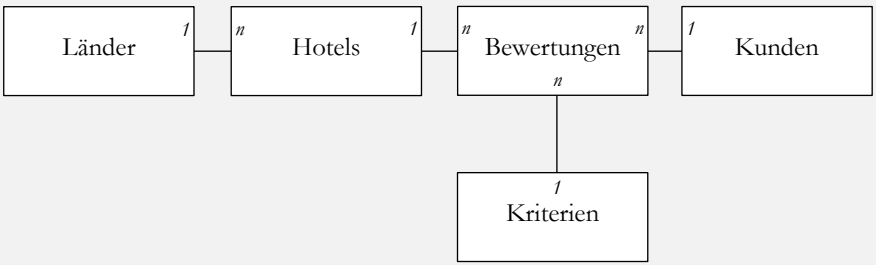
c. Überführen Sie diese Relationen wenn notwendig in die 3 NF.

R_1 *(A, B, C)*	R_2 *(B, D)*	R_3 *(D, E)*
S_1 *(D, E)*	S_2 *(E, F)*	

5. Aufgabe

Zu einem Internet-Portal für Hotelbuchungen gibt es eine Datenbank mit den *Bewertungen* der Gäste. *Länder* werden durch ein *Nationalitätskennzeichen* und den *Ländernamen* beschrieben. Den Hotels sind eine *HotelID* sowie der *Name* und die Anzahl der *Sterne* zugeordnet. In einem Land gibt es mehrere dort vorhandene Hotels. Die *Bewertungskriterien* bestehen aus einem *Schlüssel* und einem *Text*. Die *Kunden,* von denen der *Name* und die *E-Mail-Adresse* bekannt sind, haben zusätzlich eine *KundenID*. Die Kunden bewerten Hotels anhand der Bewertungskriterien jeweils mit den *Noten* 1 bis 6. Zur Vereinfachung wird angenommen, dass ein Kunde ein Hotel nur einmal bewerten kann.

a. Bilden Sie diesen Sachverhalt durch ein ERM-Modell ab.



Die Rauten <> wurden zur Vereinfachung des ERM's weggelassen

b. Stellen Sie dazu ein Relationsschema auf; Schreibweise z.B. *Kunden (KundenID, …)*. Außerdem die Primärschlüssel unterstreichen und die Fremdschlüssel markieren *(beides: Doppelt unterstrichen)*.

`Länder` *(Nationalkennzeichen, Name)*
`Hotels` *(HotelID, Name, Sterne, Nationalkennzeichen)*
`Kunden` *(KundenID, Name, eMail)*
`Kriterien` *(Schlüssel, Text)*
`Bewertungen` *(HotelID, KundenID, Schlüssel, Note)*

Aufgekommene Fragen während der Bearbeitung

Praktikumsaufgabe 1

☞ Ist das Set *(also die Menge an Tupel)* auf eine Anzahl limitiert?

Nein, die Menge der Tupel ist quasi unendlich. Allerdings besteht die Möglichkeit, eine maximale Anzahl vorzugeben, um die Menge der Tupel zu limitieren. Ein SQL-Statement wäre zum Beispiel `LIMIT [OFFSET] ROW`.

☞ Wie werden der Primary Key und die Limitierungen festgelegt?

Beides erfolgt über SQL-Statements. Betrachte hierfür eingehender den `CREATE TABLE` oder `ALTER TABLE` Befehl in den Praktikumsaufgaben 2 und 3.

☞ Ist es möglich, die festgelegten Einschränkungen einzusehen?

Ja, mit einem Doppelklick auf die Relationsbezeichnung. Danach erscheint eine zusätzliche Registerkartei im Reiter, die Unterregisterkarteien enthält. Dort kann man allgemein die Implementierung der Spalten, aber auch die Daten, die Constraints, Trigger, Indizes *(Primary Keys)* und so weiter einsehen.

☞ Existieren außer `ORDER BY` weitere Befehle zur Sortierung?

Nein, `ORDER BY [ASC | DESC]` ist die Methoden der Wahl, um eine Ausgabe auf- bzw. absteigend zu sortieren.

☞ Ist die *Kunde-Bestellung-Beziehung* eine 1:n- oder m:n-Beziehung?

Dies hängt allein vom Datenbankentwurf und dem Zweck der Datenbank ab. Mit einer Minimal-Maximal-Notation *(min-max)* wären beide Versionen vorstellbar.

Praktikumsaufgabe 2

☞ Englische oder deutsche Relations- bzw. Attributbezeichnungen?

In der Regel benutzt man englische Bezeichner. Dennoch kann es unter Umständen sinnvoll sein, auch deutsche Bezeichnungen zu verwenden. Entscheidend sind hierbei die Adressaten der Datenbank, um die es geht.

☞ Kann die `PLZ` anders als mit `VARCHAR2` mit einer `0` beginnen?

Nein, mit einem Textdatentyp trifft man die richtige Wahl.

☞ Wann entscheidet man sich für `NUMBER`, wann für `INTEGER`?

Die beiden Datentypen weisen grundlegende Unterschiede im Speicherplatzverbrauch und im Wertebereich auf. Betrachte hierfür die Seite 37 mit den Hinweisen zu den Datentypen.

Praktikumsaufgabe 3

☞ Kann man sich die Tabellen bzw. Abfragen ausdrucken lassen?

Ja, wie gewohnt unter *Datei → Drucken.*

Praktikumsaufgabe 4

☞ -

Praktikumsaufgabe 5

☞ Gibt es eine sicherere Methode, die alle Partnernummern den richtigen Geschäftspartnern in der Lieferanten- und Kundendaten Relation zuordnet?

> Es besteht die Möglichkeit, eine komplexe Datenbankprogrammierung vorzunehmen. Für Anfänger ohne Vorerfahrung kann es aber sinnvoller sein, wenn man sich konzentriert an die verwendete Variante hält, damit keine Fehler unterlaufen.

Hinweisverzeichnis

Abbildungsverzeichnis

Stichwortverzeichnis