4. **Geben Sie den folgenden Befehl im unteren Teil des Fensters ein (siehe Abbildung 9.1):**

```
create table FISCH_NEU (
FISCH_NAME VARCHAR2(10) NOT NULL PRIMARY KEY,
AQUARIUM VARCHAR2(10) NULL,
GEBURTSTAG DATE NULL,
TODESTAG DATE NULL,
FARBEN VARCHAR2(10) NULL,
GATTUNG VARCHAR2(10) NULL,
GESCHLECHT CHAR(1) NULL,
BEMERKUNG VARCHAR2(100))
tablespace USER_DATA;
```

5. **Klicken Sie auf die *Execute*-Schaltfläche, um den Befehl auszuführen.**

   Das SQL-Worksheet kopiert den Befehl in den oberen Abschnitt des Fensters und führt ihn aus. Wenn der Befehl ausgeführt ist, meldet das SQL-Worksheet im oberen Fenster:

```
Anweisung verarbeitet.
```

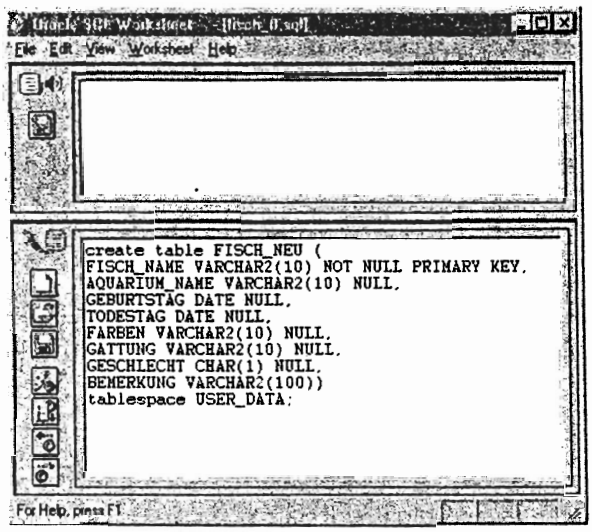


Abbildung 9.1: Eine Tabelle mit dem SQL-Worksheet erstellen.

Wie können Sie eine eigene Tabelle wieder loswerden? Kein Problem! Lassen Sie sie einfach fallen. Der SQL-Code lautet:

```
DROP TABLE tablename;
```

`tablename` steht (natürlich) für den Namen der Tabelle, die Sie löschen wollen.

☒ **Ändere die Personalnummer 10102 in der Tabelle *mitarbeiter*. Die neue Nummer ist 22222.**

Mit diesem Text wird jetzt der Wert im Primärschlüssel der Tabelle **mitarbeiter** durch einen neuen Wert ersetzt. Dadurch entstehen nicht konsistente Werte im Fremdschlüssel der Tabelle **arbeiten**. Die Änderung, die mit der folgenden Transact-SQL-Anweisung durchgeführt werden kann:

```
update mitarbeiter
    set m_nr = 22222
    where m_nr = 10102
```

wird abgewiesen, falls die referentielle Integrität für die beiden Tabellen definiert ist.

☒ **Lösche die Reihe der Tabelle *mitarbeiter* mit der Personalnummer 10102.**

Dieser Fall ist dem vorherigen ähnlich. Durch das Löschen der Reihe im Primärschlüssel der Tabelle **mitarbeiter** entstehen nicht konsistente Werte im Fremdschlüssel der Tabelle **arbeiten**.

Alle vier o.g. Fälle werden beim SQL Server mit der Ausgabe einer entsprechenden Fehlermeldung abgewiesen. Beispiel 3.15 zeigt die Tabellen der Beispieldatenbank mit allen dazugehörigen Primär- und Fremdschlüsseln.

Beispiel 3.15

```
create table abteilung1
    (abt_nr char(4) not null,
    abt_name char(20) not null,
    stadt char(15) null,
    primary key(abt_nr));

create table projekt1
    (pr_nr int char(4) null,
    pr_name char(25) not null,
    mittel float null,
    primary key(pr_nr));

create table mitarbeiter1
    (m_nr integer not null,
    m_name char(20) not null,
    m_vorname char(20) not null,
    abt_nr char(4) null,
    primary key(m_nr),
    foreign key(abt_nr) references abteilung1(abt_nr));

create table arbeiten1
    (m_nr integer not null,
    pr_nr char(4) not null,
    aufgabe char(15) null,
    einst_dat date null,
    primary key(m_nr, pr_nr),
    foreign key(m_nr) references mitarbeiter1(m_nr),
    foreign key(pr_nr) references projekt1(pr_nr));
```

Notizen

```
insert into arbeiten values (25348, 'p2',
                             'Sachbearbeiter', '15.2.1988')
insert into arbeiten values (18316, 'p2',null, '1.6.1989')

insert into arbeiten values (29346, 'p2',null, '15.12.1987')
insert into arbeiten values (2581, 'p3',
                             'Projektleiter', '15.10.1989')
insert into arbeiten values (9031, 'p1',
                             'Gruppenleiter', '15.4.1989')
insert into arbeiten values (28559, 'p1',null, '1.8.1988')
insert into arbeiten values (28559, 'p2',
                             'Sachbearbeiter, '1.2.1989')
insert into arbeiten values (9031, 'p3',
                             'Sachbearbeiter', '15.11.1988')
insert into arbeiten values (29346, 'p1',
                             'Sachbearbeiter, '1.4.1989')
```

BEISPIEL

Beispiel 6.2

Laden Sie alle Reihen der Tabelle **projekt**.

```
insert into projekt values('p1', 'Apollo', 120000.0)
insert into projekt values('p2', 'Gemini', 95000.0)
insert into projekt values('p3', 'Merkur', 186500.0)
```

Beispiel 6.3

BEISPIEL

Laden Sie alle Reihen der Tabelle **abteilung**.

```
insert into abteilung values('a1', 'Beratung', 'München')
insert into abteilung values('a2', 'Diagnose', 'München')
insert into abteilung values('a3', 'Freigabe', 'Stuttgart')
```

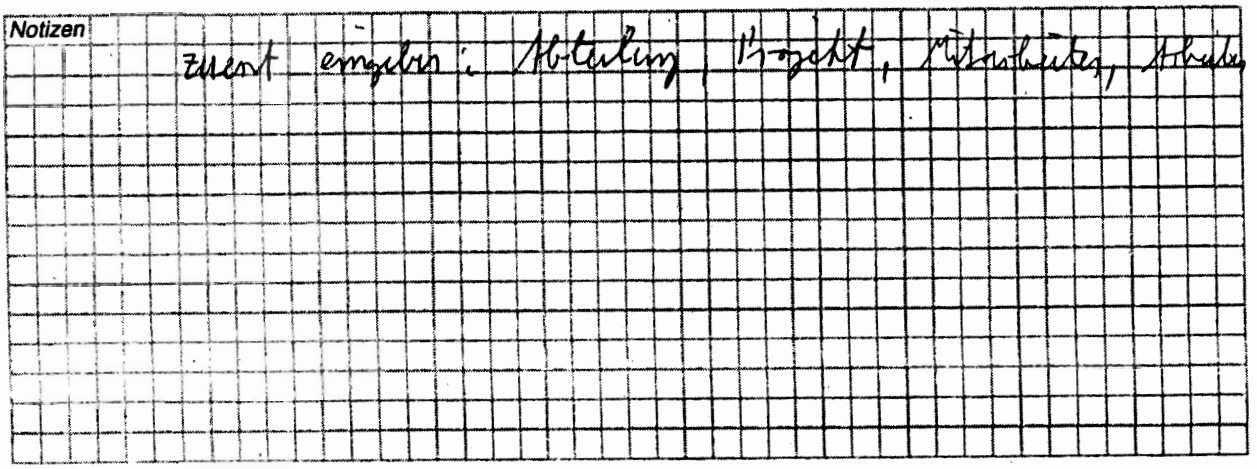
Beispiel 6.4

BEISPIEL

Laden Sie alle Reihen der Tabelle **mitarbeiter**.

```
insert into mitarbeiter values(25348, 'Keller', 'Hans', 'a3')
insert into mitarbeiter values(10102, 'Huber', 'Petra', 'a3')
insert into mitarbeiter values(18316, 'Müller', 'Gabriele', 'a1')
insert into mitarbeiter values(29346, 'Probst', 'Andreas', 'a2')
insert into mitarbeiter values(9031, 'Meier', 'Rainer', 'a2')
insert into mitarbeiter values(2581, 'Kaufmann', 'Brigitte', 'a2')
insert into mitarbeiter values(28559, 'Mozer', 'Sibille', 'a1')
```

Notizen



Die relationale Datenbank für das soeben beschriebene Schema ist in folgenden Tabellen dargestellt.

Tabelle **abteilung**

abt_nr	abt_name	stadt
a1	Beratung	München
a2	Diagnose	München
a3	Freigabe	Stuttgart

Tabelle **mitarbeiter**

m_nr	m_name	m_vorname	abt_nr
25348	Keller	Hans	a3
10102	Huber	Petra	a3
18316	Müller	Gabriele	a1
29346	Probst	Andreas	a2
9031	Meier	Rainer	a2
2581	Kaufmann	Brigitte	a2
28559	Mozer	Sibille	a1

Tabelle **projekt**

pr_nr	pr_name	mittel
p1	Apollo	120000.0
p2	Gemini	95000.0
p3	Merkur	186500.0

Tabelle **arbeiten**

m_nr	pr_nr	aufgabe	einst_dat
10102	p1	Projektleiter	1.Okt.1988 0:00
10102	p3	Gruppenleiter	1.Jan.1989 0:00
25348	p2	Sachbearbeiter	15.Feb.1988 0:00
18316	p2		1.Jun.1989 0:00
29346	p2		15.Dez.1987 0:00
2581	p3	Projektleiter	15.Okt.1989 0:00
9031	p1	Gruppenleiter	15.Apr.1989 0:00
28559	p1		1.Aug.1988 0:00
28559	p2	Sachbearbeiter	1.Feb.1989 0:00
9031	p3	Sachbearbeiter	15.Nov.1988 0:00
29346	p1	Sachbearbeiter	1.Apr.1989 0:00

Mit Hilfe unseres Beispiels können wir jetzt einige wichtige Eigenschaften des relationalen Modells erklären:

- ☒ Die Reihen innerhalb einer Tabelle können eine beliebige Reihenfolge haben.
- ☒ Die Spalten innerhalb einer Tabelle können eine beliebige Reihenfolge haben.
- ☒ Alle Datenwerte einer Spalte haben genau denselben Datentyp.
- ☒ Jede Spalte hat einen eindeutigen Namen innerhalb einer Tabelle. Spalten, die verschiedenen Tabellen angehören, können durchaus denselben Namen haben. (Beispiel: Die Spalte **m_nr** in der Tabelle **arbeiten** und die Spalte **m_nr** in der Tabelle **mitarbeiter**.)
- ☒ Jeder einzelne Datenwert innerhalb einer Tabelle ist durch einen einzigen Wert dargestellt. Das heißt, im Schnittpunkt einer Reihe und einer Spalte können sich nie mehrere Werte gleichzeitig befinden.

6 Änderung der Tabelleninhalte

In diesem Kapitel erfahren Sie

→ wie Tabelleninhalte geändert werden mit Hilfe von
- INSERT-Anweisungen
- UPDATE-Anweisungen
- DELETE-Anweisungen

6.1 Die INSERT-Anweisung

Mit der INSERT-Anweisung können Reihen oder Teile von Reihen in einer bestehenden Tabelle eingefügt werden. Dabei existieren zwei unterschiedliche Formen dieser Anweisung:

- INSERT INTO tab_name|view_name[(spalte_1,...)]
  {DEFAULT VALUES|VALUES (ausdruck_1,...)}
- INSERT INTO tab_name|view_name [(spalte_1,...)]
  select_anweisung

Mit der ersten Form der INSERT-Anweisung werden eine einzige Reihe oder Teile einer Reihe in der Tabelle **tab_name** (oder mittels View namens **view_name** in der darunterliegenden Tabelle) eingefügt. Mit der zweiten Form werden mehrere Reihen oder Teile mehrerer Reihen mit Hilfe einer SELECT-Anweisung gleichzeitig in der Tabelle eingefügt.

Für beide Formen der INSERT-Anweisung gilt, daß jeder eingefügte Datenwert denselben Datentyp wie die Spalte haben muß, in der er eingefügt wird. Dabei sollen die alphanumerischen Werte und die Datums- und Zeitangaben immer in Apostrophen bzw. Anführungszeichen eingeschlossen werden, während man numerische Werte ohne Apostrophe (Anführungszeichen) schreiben kann.

6.1.1 Einfügen einer Reihe

Bei der ersten Form der INSERT-Anweisung ist die explizite Angabe der Spaltenliste optional. Falls alle Spalten einer Tabelle eingefügt werden sollen, kann auf die Angabe der Liste verzichtet werden.

Die Angabe DEFAULT VALUES fügt die Standard (default-)Werte für alle Tabellenspalten ein. Falls eine Spalte entweder den TIMESTAMP-Datentyp oder die IDENTITY-Eigenschaft hat, werden die entsprechenden Werte eingefügt. Der NULL-Wert wird eingefügt, falls für die Spalte sowohl NULL-Werte erlaubt sind als auch kein Standardwert definiert ist. In allen anderen Fällen wird eine Fehlermeldung ausgegeben.

In den folgenden vier Beispielen werden die Reihen aller Tabellen der Beispieldatenbank eingefügt. Damit wird gleichzeitig die Möglichkeit aufgezeigt, die Reihen einer Datenbank zu laden.

BEISPIEL

Beispiel 6.1

Laden Sie alle Reihen der Tabelle **arbeiten**:

```
insert into arbeiten values (10102,'p1',
                             'Projektleiter','1.10.1988')
insert into arbeiten values (10102,'p3',
                             'Gruppenleiter','1.1.1989')
```

aussehen. Wie auch immer – wenn Sie sind auf einem PC arbeiten, sollten Sie die Sorge um die Struktur Oracle8 überlassen.

Nehmen Sie an, daß Sie für die Planung der Tabellenstruktur der Datenbank oder eines Teils der Datenbank verantwortlich sind. Sie können die Tabellen in einem großen Tablespace speichern oder auf viele kleine Tablespaces verteilen. Ich empfehle Ihnen, mit der Zahl der Tablespaces vorsichtig umzugehen. Jeder zusätzliche Tablespace erhöht Ihren Aufwand bei der Verwaltung der Datenbank. Wenn Sie mit vielen kleinen Tablespaces arbeiten, können Sie auf folgendes Problem stoßen: Insgesamt ist die Datenbank zwar groß genug, um alle Daten zu speichern, aber sie kann nicht den gesamten Speicherplatz nutzen, weil Sie eine Tabelle in einem Tablespace gespeichert haben, der zu klein geworden ist. Eine Tabelle kann nicht mehrere Tablespaces auf einmal nutzen, es sei denn, Sie haben sie als eine sogenannte *partitionierte Tabelle* gespeichert. (Nur große Tabellen – Tabellen mit mehr als 1 GB Speicherbedarf – sollten als partitionierte Tabellen definiert werden.) Fragen Sie immer Ihren DBA, wo Tabellen gespeichert werden sollen.

Spalten und ihre Eigenschaften

Oracle8 benötigt Tabellen, und Tabellen brauchen Spalten. Selbst wenn eine Tabelle keine Zeilen enthält, muß sie Spalten haben. Wenn Sie Spalten definieren, müssen Sie ihren Namen, ihren Datentyp und ihr Nullwert-Flag festlegen.

Kapitel 2 behandelt die Namensregeln für Spalten, Tabellen, Indexe und andere Datenbankobjekte. Dieser Abschnitt behandelt die Datentypen und das Nullwert-Flag Ihrer Daten.

Spalten in Oracle8 definieren

Spalten werden beim Erstellen einer neuen Tabelle definiert. Jede Spalte hat einen Namen und einen Datentyp, der festlegt, welche Art von Daten in der Spalte gespeichert werden darf. Tabelle 9.1 gibt einen Überblick über die möglichen Datentypen. Sie sollten die Tabelle mit einem Lesezeichen versehen, damit Sie später schnell darauf zurückgreifen können.

Datentyp	Parameter	Beschreibung
VARCHAR2(n)	n=1 bis 4,000	Zeichenkette mit variabler Länge. (n) spezifiziert die maximale Länge. Dieser Datentyp speichert Buchstaben, Ziffern und Symbole des Standard-ASCII-Zeichensatzes (oder EBCDIC oder des Standardzeichensatzes Ihrer Datenbank). Wenn Ihre Daten kürzer als die maximale Größe sind, paßt Oracle8 die Spaltenbreite an die Länge der Daten an. Leerzeichen am Ende der Zeichenkette werden abgeschnitten. VARCHAR2 ist der gebräuchlichste Datentyp.

Datentyp	Parameter	Beschreibung
NUMBER(p,s)	p=1 bis 38, s=0 bis 84	Zahl. (p) spezifiziert die Genauigkeit als Anzahl der Ziffern. Zulässig sind Werte bis zu 127 oder FLOAT. (s) spezifiziert die Anzahl der Dezimalstellen rechts vom Dezimalkomma; z.B. wird eine Zahl mit einem Maximum von *999.99* als NUMBER(5,2) definiert. FLOAT gibt an, daß der Dezimalpunkt an beliebiger Stelle der Zahl stehen kann. Oracle8 rundet längere Daten so, daß sie in das definierte Format passen. Wenn Sie z.B. eine Spalte als NUMBER(5,2) definieren, speichert Oracle8 die Zahl *575.316* als *575.32*. Wenn Sie eine Spalte als NUMBER(3,0) definieren, wird die *575.316* als *575* gespeichert.
DATE	Keine	Datum. Zulässige Daten reichen vom 1. Januar 4712 v. Chr. (-4712) bis zum 31. Dezember 4712 n. Chr. Oracle8 speichert DATE intern als 7-Byte-Zahl. Diese Zahl enthält definitionsgemäß auch die Zeit in Stunden, Minuten und Sekunden.
CHAR(n)	n=1 bis 255	Zeichenkette fester Länge. (n) spezifiziert die maximale Länge. Die Standardlänge ist *1*. Daten, die kürzer als die definierte Maximallänge sind, werden rechts mit Leerzeichen aufgefüllt. Wenn Sie z.B. eine Spalte als CHAR(10) definiert haben, fügt Oracle8 an die Zeichenkette HEINER, die sechs Zeichen lang ist, vier Leerzeichen an. Der Hauptunterschied zwischen CHAR und VARCHAR2 besteht darin, daß CHAR die Daten mit Leerzeichen auffüllt und VARCHAR2 Leerzeichen abschneidet.
NCHAR(n)	n=1 bis 2,000	Siehe CHAR. Derselbe Datentyp, außer daß die Zeichen in einem nationalen Zeichensatz (z.B. chinesische Zeichen) gespeichert werden. Oracle8 unterstützt auf diese Weise viele Sprachen.
NVARCHAR2(n)	n=1 bis 4,000	Siehe VARCHAR2. Derselbe Datentyp, außer daß die Zeichen in einem beliebigen nationalen Zeichensatz gespeichert werden, den Oracle8 unterstützt.
LONG	Keine	Zeichenkette variabler Länge. Die maximale Länge beträgt 2 GB. LONG ist für große Daten bestimmt. Pro Tabelle dürfen Sie nur eine Spalte mit dem Datentyp LONG definieren, und diese Spalte muß am Ende der Tabelle stehen. Der LONG-Datentyp darf in einer SQL-Abfrage nur in der SELECT-Klausel, aber nicht in den anderen Klauseln (WHERE, GROUP BY oder ORDER BY) verwendet werden. Eine Spalte mit dem LONG-Datentyp kann nicht nach bestimmten Zeichen durchsucht werden. Kapitel 3 enthält Beispiele für die Klauseln WHERE, GROUP BY und ORDER BY. In der Dokumentation finden Sie noch viele andere Restriktionen für den LONG-Datentyp.

Datentyp	Parameter	Beschreibung
		Mein Ratschlag: Benutzen Sie diesen Datentyp nur, wenn Sie unbedingt große Datenblöcke speichern wollen und diese nicht nach Text durchsuchen müssen. Sonst wirken sich die Einschränkungen dieses Datentyps zu störend aus. Benutzen Sie VARCHAR2, wenn Sie den Text durchsuchen wollen. LONG ist ein älterer Datentyp, der irgendwann vollständig durch die Datentypen CLOB und NCLOB für große Objekte abgelöst werden wird.
RAW(n)	n=1 bis 255	Rohe Binärdaten variabler Länge. (n) spezifiziert die maximale Länge der Daten. Oracle8 benutzt den RAW-Datentyp für kleine Grafiken und formatierte Textdateien, wie z.B. Word-Dokumente. RAW ist ein älterer Datentyp, der irgendwann vollständig durch die Datentypen BLOB, CLOB, NCLOB und BFILE für große Objekte abgelöst werden wird.
LONGRAW	Keine	Rohe Binärdaten variabler Länge. Die maximale Länge beträgt 2 GB. Oracle8 benutzt den LONGRAW-Datentyp für größere Grafiken, formatierte Textdateien, wie z.B. Word-Dokumente, Audio-, Video- und andere Nichttext-Daten. Die Datentypen LONG und LONGRAW dürfen nicht zusammen in einer Tabelle definiert werden. LONGRAW ist ein älterer Datentyp, der irgendwann vollständig durch die Datentypen BLOB, CLOB, NCLOB und BFILE für große Objekte abgelöst werden wird.
BLOB, CLOB, NCLOB	Keine	Drei Arten großer binärer Objekte (LOB = *Large Binary Object*); Oracle8 benutzt diese Datentypen für größere Grafiken, formatierte Textdateien, wie z.B. Word-Dokumente, Audio-, Video- und andere Nichttext-Daten. Die Maximalgröße beträgt 4 GB. Es gibt verschiedene Varianten von LOBs, die von der Art der benutzten Bytes abhängen. Oracle8 speichert diese LOBs intern in der Datenbank. Siehe auch BFILE. Es gibt spezielle Funktionen zum Lesen, Speichern und Schreiben von LOBs, die aber nicht Thema dieses Buches sind. Wenn Sie diese Datentypen benutzen wollen, lesen Sie in der Oracle8-Dokumentation nach.
BFILE	Keine	Großes Binärobjekt (LOB = *Large Binary Objects*), das außerhalb der Datenbank gespeichert wird. Die Maximalgröße beträgt 4 GB. Dieser externe LOB-Datentyp wird zwar in der Datenbank verwaltet, aber tatsächlich außerhalb der Datenbank in einer separaten Datei gespeichert. Oracle8 kann BFILEs lesen und abfragen, aber nicht schreiben. Es gibt spezielle Funktionen zum Lesen, Speichern und Schreiben von LOBs, die aber nicht Thema dieses Buches sind. Wenn Sie diese Datentypen benutzen wollen, lesen Sie in der Oracle8-Dokumentation nach.

Tabelle 9.1: Übersicht über die Datentypen

Die gebräuchlichsten Datentypen sind VARCHAR2, DATE und NUMBER. Der CHAR-Datentyp wird meistens für codierte Daten mit konstanter Länge, wie z.B. die Abkürzungen der US-Bundesstaaten, verwendet.

Die anderen Datentypen dienen speziellen Zwecken und werden seltener benutzt (es sei denn, Sie erstellen eine Datenbank mit Musiktiteln, die auf Anforderung abgespielt werden können).

Null, leer oder nicht leer

Wenn Sie Spalten definieren, müssen Sie festlegen, ob die Spalten leer sein dürfen (siehe den Einschub *Was ist ein Nullwert?*) oder immer Daten enthalten müssen. Es gilt folgende Regel: Der Primärschlüssel (der eindeutiger Bezeichner für die Zeilen einer Tabelle) darf keine Nullwerte enthalten. Abgesehen davon ist alles erlaubt. Ich rate Ihnen, Nullwerte überall außer bei dem Primärschlüssel zuzulassen. Selbst wenn Sie meinen, daß eine Spalte immer Daten enthalten wird, gibt es Ausnahmen. Wenn Sie festlegen, daß eine Spalte keinen Nullwert enthalten darf, stehen Sie vor einem Problem, wenn Sie eine neue Zeile einfügen wollen, für die es in der betreffenden Spalte tatsächlich keinen Wert gibt.

Die Eigenschaft, daß eine Spalte keinen Nullwert enthalten darf, können Sie durch einen CREATE TABLE-Befehl mit einer NOT NULL-Einschränkung (engl. *constraint*) für die betreffende Spalte definieren (siehe nächster Abschnitt).

In Kapitel 17 finden Sie Informationen darüber, wie Sie die Nullwert-Eigenschaft einer Spalte ändern können (Nullwerte zulassen oder verbieten).

Was ist ein Nullwert?

Wenn eine Spalte einer Zeile keine Daten enthält, sagt Oracle8, daß die Spalte *null* ist oder einen *Nullwert* enthält. Ein Nullwert wird benutzt, wenn Daten unbekannt oder nicht vorhanden sind. Wenn ich z.B. die Fische in meinem Aquarium mit einer Tabelle verwalten will, speichere ich in der Spalte TODESTAG einen Nullwert, falls der betreffende Fisch noch lebt oder ich nicht weiß, wann er gestorben ist.

In Oracle8 kann können Spalten jedes Datentyps mit einem Nullwert-Flag versehen werden. Oracle8 läßt nur in den folgenden beiden Fällen keine Nullwerte in Spalten zu:

- ✔ in Primärschlüsselspalten
- ✔ in Spalten, für welche die NOT NULL-Einschränkung definiert ist

Ein weiteres Beispiel für die Definition von Nullwerten: Stellen Sie sich vor, daß Ihr Scheckbuchregister eine Spalte enthält, in die Sie ein kleines *X* eintragen, wenn Sie Ihren Kontoauszug mit dem Register abstimmen. Diese Spalte soll den

Um die Integrität einer Datenbank zu gewährleisten, beachtet man die Grundsätze der **Normalformenlehre**. Damit verhindert man, daß unnütze und sich widersprechende Informationen gespeichert werden. Wenn Sie komplexe Datenbankmodelle entwickeln wollen, sollten Sie sich u.a. damit auseinandersetzen.

In Kurzform folgende Grundsätze:

1. **Kein Feld darf mehrfach vorkommen.**
2. **Jedes Feld, das nicht zum Primärschlüssel gehört, muß von dem Primärschlüssel abhängig sein, sonst muß es ausgelagert werden, und**
3. **jedes Feld, das von einem anderen als einem Primärschlüsselfeld abhängig ist, muß ausgelagert werden.**

Die Nummern in der folgenden Tabelle „Produkte" kennzeichnen, welche Regel bei dem Feld mißachtet wurde:

ProduktNr *Produktbez* *ProduktgruppenNr* *Gruppenbez*(3) *Kunde1*(1,2) *Kunde2*(1,2) *Kunde3*(1,2)

Diese Tabelle versucht alle Daten aufzunehmen und wird dadurch problematisch. Es läßt sich schlecht vorhersehen, wie viele Kunden ein Produkt bestellen. Für jeden möglichen Kunden ein bzw. mehrere Felder vorzusehen ist also sehr inflexibel. Das Feld „Kunde" nur einmal aufzunehmen wäre falsch, da pro Kunde immer wieder dieselben Produktinformationen eingetragen werden müßten.

● Diese Aufzeichnungen sollten Sie immer anfertigen, bevor Sie die Tabellen in Access anlegen. Gehen Sie in Gedanken und anhand schriftlicher Vorlagen durch, ob die Informationen zur Erstellung Ihrer Formulare und Berichte ausreichen. Sobald Daten in den Tabellen stehen, können Änderungen an der Tabellenstruktur schwierig werden und sogar zu Datenverlust führen. Möchte man nachträglich die Länge eines Feldes verkürzen oder sogar dessen Datentyp verändern, kann Ihr Datenbestand Schaden nehmen. Konzeptionelle Änderungen haben oft Wechselwirkungen, die nur schwer zu überblicken sind.

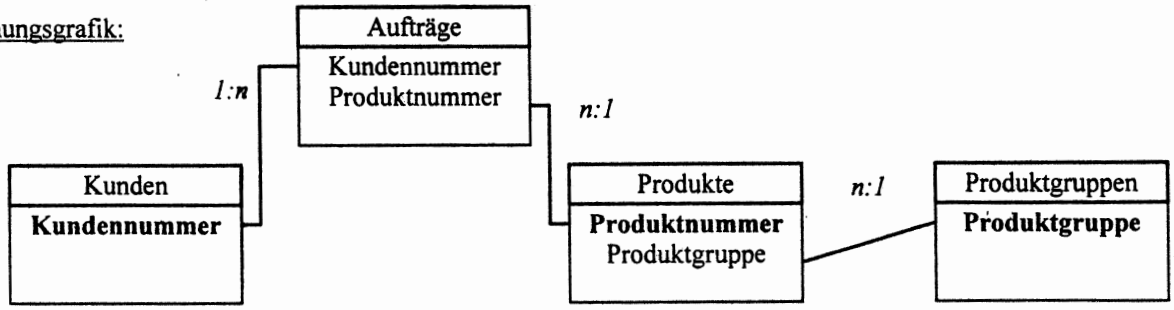
Streng nach der Normalformenlehre gedacht, müßte sogar noch eine weitere Tabelle für die Postleitzahlen und Ortschaften existieren, damit die gleiche Ortschaft nicht mehrmals in der Kundentabelle stehen muß. Hierbei sind jedoch praktische Überlegungen höher zu bewerten als die theoretische Notwendigkeit. Der Fachmann nennt das „Denormalisieren".

Generell bereitet es anfangs Schwierigkeiten, die „richtige" Aufteilung der Tabellen zu finden. Hier spielt auch die Erfahrung eine wichtige Rolle. Die ein oder andere Frage zu unserem Modell klärt sich sicher bei dem nächsten Thema.

ⓘ Ein Feld, das in einer Tabelle Primärschlüssel ist und in einer weiteren Tabelle vorkommt, nennt man dort **Sekundärschlüssel.**

ⓘ Durch unsere Aufteilung müssen die Daten (z. B. Adressenänderungen) nur an einer Stelle gepflegt werden.

Beziehungsgrafik:



Aufnahmestruktur der Tabellen umsetzen

Das Ergebnis dieses Arbeitsschrittes nennen wir **Aufnahmestruktur**. Sie gehört ebenso dringend wie das Beziehungsschema zu einer sorgfältigen Dokumentation.

1. Richten Sie die vier Tabellen ein, die Sie hier auf der linken und rechten Seite finden. Das zuerst aufgeführte Feld jeder Tabelle ist das Primärschlüsselfeld.
2. Definieren Sie weitere Eigenschaften, wenn Sie Ihnen sinnvoll erscheinen. Legen Sie für die Telefonnummer mit Hilfe des Eingabeformatassistenten ein Eingabeformat fest, z. B. „(0421) 123 456". Überlegen Sie sich Gültigkeitsbereiche für die Nummernfelder.

Tabelle „Aufträge":

Feldname	Datentyp	Größe
AuftragNr	[redacted]	4
ProduktNr	Text	2
KundenNr	Zahl (Long)	4
Menge	Zahl (Byte)	1
BestDat	Datum/Zeit	8
Rabatt	Ja/Nein	1

Tabelle „Kunden":

Feldname	Datentyp	Größe
KundenNr	Zähler (Long)	4
Firma	Text	40
Name	Text	25
Vorname	Text	20
Anrede	Text	15
Straße	Text	30
PLZ	Text	5
Ort	Text	40
Telefon	Text	20
Fax	Text	20

Beispieldaten erfassen

Füllen Sie die Tabellen mit den hier und auf der rechten Seite abgebildeten Daten:

Tabelle: Aufträge

Auftrags-Nr.	Produkt-Nr.	Kunden-Nr.	Menge	Datum	5% Rabatt
001	10	002	4	05.06.94	Nein
002	30	001	5	03.06.94	Ja
003	21	003	1	07.06.94	Nein
004	30	004	3	09.06.94	Ja
005	31	004	2	05.06.94	Ja
006	10	004	4	05.06.94	Ja
007	21	003	1	10.06.94	Nein
008	30	001	2	20.06.94	Ja
009	20	002	1	28.06.94	Nein

Tabelle: Produktgruppen

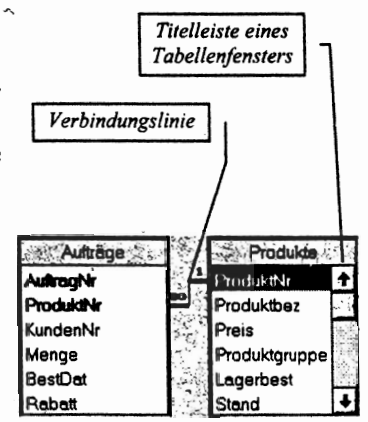
Produktgr	Gruppenbezeichnung
1	Stühle
2	Tische
3	Regale

In Felder mit dem Datentyp „Zähler" können Sie selbst keine Daten eingeben.

Beziehungen definieren

Zuletzt müssen die Beziehungen zwischen den Tabellen in Access angelegt werden.

1. Schließen Sie alle geöffneten Tabellen, und klicken Sie auf das -Symbol. Markieren Sie alle vier Tabellen, klicken Sie auf *Hinzufügen,* und bestätigen Sie mit *Schließen.* Ordnen Sie die Tabellen so an, wie auf Seite 25 unten gezeigt.
2. Ziehen Sie mit der Maus das Feld „KundenNr" aus der Tabelle „Kunden" auf das gleichnamige Feld in der Tabelle „Aufträge".
3. Aktivieren Sie das Kontrollkästchen *Mit referentieller Integrität.* Sie können dann z. B. keinen Kunden löschen, der noch Aufträge in der Tabelle „Aufträge" aufweist. Klicken Sie aber die Option *Löschweitergabe* an, werden alle Datensätze der Detailtabelle mitgelöscht. Auf jeden Fall können Sie in „Aufträge" keine Kundennummer eintragen, die in der Tabelle „Kunden" nicht angelegt wurde.
4. Klicken Sie auf *Erstellen,* und legen Sie alle weiteren Beziehungen (Seite 25) mit referentieller Integrität an, bevor Sie das Fenster schließen und die Speicherung mit *Ja* bestätigen.



Die Verbindungslinie zeigt die erfolgreiche Definition der Beziehung an. Ordnen Sie die Tabellenfenster durch Ziehen mit der Maus an der jeweiligen Titelleiste übersichtlich an.

Tabelle „Produkte“:

Feldname	Datentyp	Größe
ProduktNr	Text	2
Produktbez	Text	30
Lagerbest	Zahl (Integer)	2
Produktgruppe	Text	2
Preis	Währung	8
Stand	Datum/Zeit	8
Bild	OLE-Objekt	-

Tabelle „Produktgruppen“:

Feldname	Datentyp	Größe
Produktgruppe	Text	2
Gruppenbezeichnung	Text	30

- Die Feldgrößen brauchen Sie nur für Textfelder selbst einzustellen. Felder, die bis zu vier Nachkommastellen haben und mit denen häufig gerechnet wird, bekommen besser den Datentyp „Währung“ anstatt „Single“ oder „Double“. Die **Festkommaarithmetik** des Datentyps „Währung“ arbeitet schneller und ohne Rundungsfehler. Das Feld „PLZ“ ist kein Feld mit dem Datentyp „Zahl“, weil es keine **führenden Nullen** speichert.

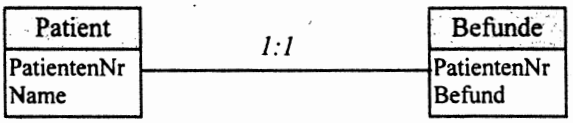
Tabelle: Kunden

Kunden	Firma	Name	Vorname	Anrede	Straße	PLZ	Ort	Telefon	Fax
[redacted]	Elektro Ebers	Ebers	Eberhardt	m	Erlenallee 23	24233	Eisenburg	02411/19786	02411/19787
002		Böse	Benno	m	Bärenweg 12	97845	Burghausen	09871/112344	09871/112345
003		Fröhlich	Frieda	w	Findorfstr. 234	45666	Feuerbach	0234/1234	0234/1234
004	Mode Meier	Mobius	Martina	w	Malvengarten 77	65443	Marienstadt	06345/7070	06345/7071
000									

Tabelle: Produkte

Prod-Nr.	Produktbezeichnung	Bestand	Art	Preis	Stand
10	Küchenstuhl	7	1	85,00	31.05.94
13	Liegestuhl	8	1	145,00	31.05.94
20	Küchentisch	12	2	450,00	31.05.94
21	Gartentisch	2	2	227,00	31.05.94
30	Bücherregal	12	3	230,00	31.05.94
31	Küchenregal	4	3	175,00	31.05.94

Neben 1:n-Beziehungen lassen sich auch **1:1-Beziehungen** aufbauen. Dies ist beispielsweise in der untenstehenden Abbildung sinnvoll, um aus Datenschutzüberlegungen die Daten separat abzulegen.



Interessant wäre es auch, eine **m:n-Beziehung** aufzubauen, die im Prinzip zwischen den Tabellen „Produkte“ und „Kunden“ besteht. Ein Kunde kann mehrere Produkte bestellen, und ein Produkt kann von mehreren Kunden bestellt werden. Diese Verknüpfung ist in einer relationalen Datenbank jedoch nur über eine dritte Tabelle darstellbar, in unserem Fall durch die Tabelle „Aufträge“. Damit wurde die m:n-Beziehung in zwei 1:n-Beziehungen aufgeteilt.

Schalten Sie die Option *Aktualisierungsweitergabe* ein, wenn Sie z. B. die Produktnummern nachträglich ändern wollen und Access das automatisch in den betroffenen Datensätzen in der Tabelle „Aufträge“ tun soll.

- Die Tabelle auf der 1-Seite muß einen Primärschlüssel haben, damit eine Beziehung definiert werden kann. Primär- und Sekundärschlüsselfeld müssen nicht den gleichen Namen, aber unbedingt den **gleichen Datentyp** haben. Der Datentyp eines der Felder kann nur verändert werden, wenn die Beziehung gelöscht wurde.