```
ON posten
TO müller;
```

Einige Datenbanken lassen auch die Definition von Rechten für mehrere Tabellen gleichzeitig zu. Die Syntax lautet dann:

```
GRANT DELETE
ON kunde, betellung, posten
TO müller;
```

Die Mitarbeiterdaten werden so angepasst:

```
UPDATE mitarbeiter
SET abteilung = 3
WHERE name = 'Müller';
```

Übung 14.6

Frau Lehne aus der Verwaltung geht in Mutterschutz. Während dieser Zeit sollen ihre Rechte zurückgenommen werden. Und wie werden Sie die Rechte später wieder zurückgeben?

Der Befehl lautet wie folgt:

```
REVOKE ALL
ON abteilung
FROM lehne;
REVOKE ALL
ON mitarbeiter
FROM lehne;
```

Einige Datenbanken lassen auch die Definition von Rechten für mehrere Tabellen gleichzeitig zu. Die Syntax lautet dann:

```
REVOKE ALL
ON abteilung, mitarbeiter
FROM lehne;
```

und später zur Widerherstellung der Rechte:

```
GRANT SELECT, INSERT, UPDATE
ON abteilung
TO lehne;
GRANT SELECT, INSERT, UPDATE
ON mitarbeiter
TO lehne;
```

Einige Datenbanken lassen auch die Definition von Rechten für mehrere Tabellen gleichzeitig zu. Die Syntax lautet dann:

```
GRANT SELECT, INSERT, UPDATE
ON abteilung, mitarbeiter
TO lehne;
```

17.13 Lösungen zu Kapitel 15

Übung 15.1

Geben Sie mittels einer SQL-Abfrage über die Systemtabellen eine Liste aus, die alle Felder mit den dazugehörigen Tabellen auflistet. Hinweis: Die Tabellen `RDB$RELATION` und `RDB$RELATION_FIELDS` sind über das Feld `RDB$RELATION_NAME` zu verknüpfen.

Der SQL-Befehl lautet:

```
SELECT RDB$RELATION_FIELDS.RDB$FIELD_NAME,
RDB$RELATIONS.RDB$RELATION_NAME
FROM RDB$RELATIONS
   INNER JOIN RDB$RELATION_FIELDS ON
(RDB$RELATIONS.RDB$RELATION_NAME =
RDB$RELATION_FIELDS.RDB$RELATION_NAME)
```

Es werden daraufhin alle angelegten Felder einschließlich deren Tabellennamen ausgegeben. Die Verknüpfung sieht dabei wie folgt aus (siehe Abbildung 17.3):

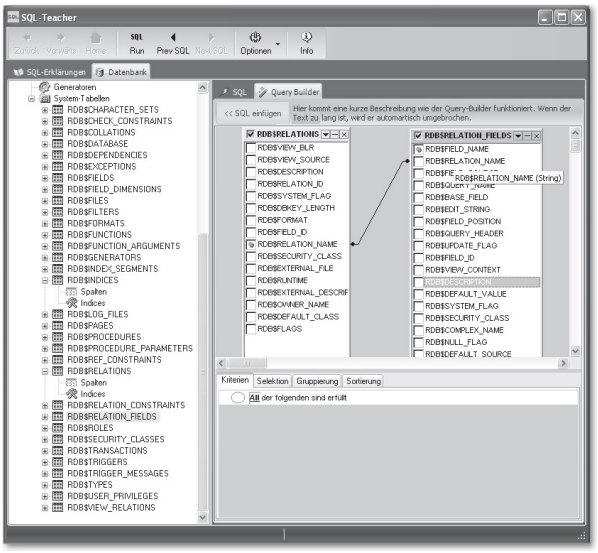

Abbildung 17.3 Verknüpfung der Systemtabellen

In diesem Kapitel wird der Aufbau der Beispieldatenbank Schritt für Schritt gezeigt. Sie können damit die Datenbank des SQL-Teachers auch in anderen Datenbanksystemen nachbauen.

18 Beispieldatenbank

Zuerst benötigen Sie eine leere Datenbank, um dort die Tabellen anzulegen. Sie legen eine neue Datenbank über SQL mit dem Befehl

```
CREATE DATABASE datenbankname;
```

an – in diesem Fall vielleicht:

```
CREATE DATABASE beispieldatenbank;
```

Es ist durchaus möglich, dass Sie in Ihrem Datenbanksystem eine neue Datenbank auch über einen grafischen Assistenten erstellen können.

In unserer Übungsdatenbank SQL-Teacher können Sie die folgenden Schritte ebenfalls nachvollziehen. Allerdings ist hier der Datenbankname vorgegeben, deshalb müssen die bestehenden Inhalte der Datenbank zuerst gelöscht werden. Ein `CREATE DATABASE` ist in der Übungssoftware aus technischen Gründen nicht möglich.

Wenn Sie die Datenbank des SQL-Teachers leeren wollen, müssen Sie in einer bestimmten Reihenfolge vorgehen: Sie müssen abhängige Tabellen vor den Vatertabellen löschen. Auch können Sie Domänen erst löschen, wenn Sie vorher die Tabellen löschen, die diese Domänen anwenden. Diese Reihenfolge sollte bei der Ausgangsdatenbank funktionieren:

```
drop table statistik;
drop table posten;
drop table bestellung;
drop table artikel;
drop table hersteller;
drop table kategorie;
drop table mwstsatz;
drop table kunde;
drop table kunde_domaene;
drop table jobticket;
```

```
drop table mitarbeiter;
drop table abteilung;
drop domain d_zahlungsart;
```

Nun können Sie darangehen und die Tabellen und Domänen neu erzeugen. Hier müssen Sie Vatertabellen vor den abhängigen Tabellen und Domänen vor den Tabellen, die sie benutzen, erzeugen. Beim Löschen gehen Sie genau umgekehrt vor.

Sie erzeugen zuerst Tabellen, die von keiner anderen Tabelle abhängig sind, also keinen Fremdschlüssel enthalten. In unserer Beispieldatenbank sind das die Tabellen `abteilung`, `hersteller`, `kategorie`, `kunde`, `mwstsatz` und `statistik`. Sie können Tabellen, die von diesen Tabellen abhängen, erst erzeugen, wenn es diese schon gibt. Für die Tabelle `kunde_domaene` müssen Sie zuerst die Domäne `d_zahlungsart` erzeugen.

Natürlich können Sie die abhängigen Tabellen sofort erzeugen, sobald die benötigten Vatertabellen existieren, also sofort nach der Tabelle `abteilung` die Tabelle `mitarbeiter` etc. Sie können hier auch mit Datentypen und Einschränkungen wie `NOT NULL` und Weitergaben wie `ON DELETE CASCADE` (siehe Kapitel 3, »Datenbankdefinition«) experimentieren. Denken Sie daran, dass Sie jederzeit die ursprüngliche Datenbank wiederherstellen können (folgen Sie dazu den Anweisungen in Abschnitt 1.4, »Übungssoftware SQL-Teacher«).

Die Tabelle `abteilung` ist eine Vatertabelle für die Tabelle `mitarbeiter`. Sie erzeugen die Tabelle `abteilung` mit diesem Befehl:

```
CREATE TABLE abteilung
(
   abteilungsnr INTEGER NOT NULL,
   bezeichnung VARCHAR(50),
   PRIMARY KEY (abteilungsnr)
);
```

Sollte das Datenbanksystem nicht auf `Autocommit` gestellt sein, geben Sie noch `COMMIT;` ein. Im SQL-Teacher stellen Sie den **Autocommit** über das Feld Optionen in der Menüleiste ein.

Sie können dann sofort Datensätze einfügen. Hier wird die wichtigste Abteilung eingetragen, nach diesem Vorbild können Sie eigene Abteilungen eingeben. Denken Sie daran, dass Sie auf jeden Fall für den Primärschlüssel einen neuen Wert angeben:

```
INSERT INTO abteilung VALUES (1, 'Geschäftsführung');
```

Nun können Sie die Tabelle `mitarbeiter` erzeugen:

```
CREATE TABLE mitarbeiter
(
   mitarbeiternr INTEGER NOT NULL,
   name VARCHAR(50),
   vorname VARCHAR(50),
   strasse VARCHAR(50),
   plz CHAR(14),
   ort VARCHAR(50),
   gehalt DECIMAL(10,2),
   abteilung INTEGER,
   telefonnummer VARCHAR(25),
   email VARCHAR(50),
   eintrittsdatum DATE,
   PRIMARY KEY(mitarbeiternr),
   FOREIGN KEY (abteilung)
      REFERENCES abteilung(abteilungsnr)
);
```

Auch hier wollen wir einen Datensatz beispielhaft eingeben, an dem Sie sich für eigene Eingaben orientieren können. Die Eingaben in der Beispieldatenbank sind selbstverständlich alle fiktiv: Die in der Beispieldatenbank aufgeführten Straßen gibt es zwar in den dazugehörigen Orten, aber wir haben andere Postleitzahlen gewählt. Außerdem haben wir alle Namen willkürlich ausgewählt:

```
INSERT INTO mitarbeiter
VALUES (1, 'Ross', 'Hagen', 'Hauptstraße 67', '53123',
   'Bonn', 7500, 1, '43567890',
   'hagen.ross@beispielfirma.de', '1986-02-18');
```

Die Tabelle `mitarbeiter` ist die Vatertabelle für die Tabelle `jobticket`. Die Tabelle ist wie folgt definiert:

```
CREATE TABLE jobticket
(
   ID INTEGER NOT NULL,
   mitarbeiternr INTEGER,
   gueltig_bis DATE,
   FOREIGN KEY (mitarbeiternr)
      REFERENCES mitarbeiter(mitarbeiternr)
);
```

Auch hier können Sie Datensätze nach diesem Muster eingeben:

```
INSERT INTO jobticket (ID, mitarbeiternr, gueltig_bis)
VALUES (1, 1, '2006-12-31');
```

Damit haben Sie alle Tabellen erzeugt, die letztlich von der Tabelle `abteilung` abhängig sind.

Sie können nun die Tabelle `kunde` erzeugen. Die Tabelle ist wie folgt definiert:

```
CREATE TABLE kunde
(
   kundennr INTEGER NOT NULL,
   name VARCHAR(50),
   vorname VARCHAR(50),
   strasse VARCHAR(50),
   plz CHAR(14),
   ort VARCHAR(50),
   telefon_gesch VARCHAR(25),
   telefon_privat VARCHAR(25),
   email VARCHAR(50),
   zahlungsart CHAR(1),
   PRIMARY KEY (kundennr)
);
```

Der folgende Datensatz kann Ihnen als Beispiel für eigene Eingaben dienen. Beachten Sie: Wenn Sie Felder als `NOT NULL` definiert haben, können Sie nicht, wie hier, einen unbekannten Wert mit `NULL` angeben. Im ersten Datensatz der Beispielfirma waren die geschäftliche Telefonnummer und die E-Mail-Adresse unbekannt:

```
INSERT INTO kunde
VALUES (1, 'Loewe', 'Arthur', 'Sebastianstrasse 134',
   '50737', 'Köln', NULL, '19467383', NULL, 'B');
```

Die Tabelle `kunde` ist die Vatertabelle für die Tabelle `bestellung`. Sie können sie folgendermaßen erzeugen:

```
CREATE TABLE bestellung
(
   bestellnr INTEGER NOT NULL,
   kundennr INTEGER,
   bestelldatum DATE,
   lieferdatum DATE,
   rechnungsbetrag DECIMAL (10,2),
   PRIMARY KEY (bestellnr),
```

```
   FOREIGN KEY (kundennr)
      REFERENCES kunde (kundennr)
);
```

An diesem Datensatz können Sie sich für eigene Eingaben orientieren:

```
INSERT INTO bestellung
VALUES (1, 1, '2011-01-02', '2004-01-11', 160);
```

Die Tabelle `bestellung` ist eine Vatertabelle für die Tabelle `posten`. Die Tabelle `posten` ist aber auch von der Tabelle `artikel` abhängig, die wiederum von den Tabellen `hersteller`, `kategorie` und `mwstsatz` abhängig ist. Um fortzufahren, müssen Sie also zunächst diese letzteren drei Tabellen und dann die Tabelle `artikel` erzeugen.

Wir definieren die Tabelle `hersteller` so:

```
CREATE TABLE hersteller
(
   herstellernr INTEGER NOT NULL,
   name VARCHAR(50),
   PRIMARY KEY (herstellernr)
);
```

Hier haben Sie als Beispiel einen Datensatz für diese Tabelle:

```
INSERT INTO hersteller VALUES (1, 'Belinea');
```

Die Tabelle `kategorie` ist wie folgt definiert:

```
CREATE TABLE kategorie
(
   kategorienr INTEGER NOT NULL,
   bezeichnung VARCHAR(50),
   PRIMARY KEY (kategorienr)
);
```

Als Beispiel mag Ihnen dieser Datensatz dienen:

```
INSERT INTO kategorie VALUES (1, 'Monitore');
```

Die dritte Vatertabelle für die Tabelle `artikel` ist die Tabelle `mwstsatz`:

```
CREATE TABLE mwstsatz
(
   mwstnr INTEGER NOT NULL,
   prozent DECIMAL(4,2),
   PRIMARY KEY (mwstnr)
);
```

Da es hier nur zwei Datensätze gibt, geben wir Ihnen beide an:

```
INSERT INTO mwstsatz VALUES (1, 7);
INSERT INTO mwstsatz VALUES (2, 19);
```

Nun können Sie die Tabelle `artikel` erzeugen. Die Tabelle ist wie folgt definiert:

```
CREATE TABLE artikel
(
   artikelnr INTEGER NOT NULL,
   bezeichnung VARCHAR(50),
   hersteller INTEGER,
   nettopreis DECIMAL(10,2),
   mwst INTEGER,
   bestand INTEGER,
   mindestbestand INTEGER,
   kategorie INTEGER,
   bestellvorschlag CHAR(1) DEFAULT '0',
   PRIMARY KEY (artikelnr),
   FOREIGN KEY (mwst)
      REFERENCES mwstsatz (mwstnr),
   FOREIGN KEY (hersteller)
      REFERENCES hersteller (herstellernr),
   FOREIGN KEY (kategorie)
      REFERENCES kategorie(kategorienr)
);
```

Hier ist der erste Datensatz unserer Beispieldatenbank für diese Tabelle:

```
INSERT INTO artikel
VALUES (1, '106075', 1, 137.93, 2, 100, 10, 1, '1');
```

Damit können Sie nun die Tabelle `posten` erzeugen:

```
CREATE TABLE posten
(
   bestellnr INTEGER NOT NULL,
   artikelnr INTEGER,
   bestellmenge INTEGER,
   liefermenge INTEGER,
   FOREIGN KEY (bestellnr)
      REFERENCES bestellung(bestellnr),
   FOREIGN KEY (artikelnr)
      REFERENCES artikel(artikelnr)
);
```

Hier haben Sie als Beispiel den ersten Datensatz der beiliegenden Datenbank:

```
INSERT INTO posten VALUES (1, 1, 1, 1);
```

Als letzte Tabelle können Sie die Tabelle `statistik` erzeugen. Sie steht für sich allein, und sie ist nötig, wenn Sie Trigger aus Abschnitt 12.2, »Trigger (CREATE TRIGGER)«, nachvollziehen möchten:

```
CREATE TABLE statistik
(
   kundenanzahl INTEGER,
   artikelanzahl INTEGER
);
```

Wenn Sie noch keine eigenen Eingaben gemacht haben, sieht der Datensatz so aus. Anderenfalls müssen Sie entsprechende Änderungen vornehmen:

```
INSERT INTO statistik (kundenanzahl, artikelanzahl)
VALUES (1,1);
```

In dieser Tabelle wird der Anfangswert der Kunden- und Artikelanzahl gespeichert.

Zum Schluss können Sie noch die Tabelle `kunde_domaene` erzeugen. Das ist beinahe dieselbe Tabelle wie die Tabelle `kunde`, nur dass Sie hier für die Spalte `zahlungsart` die Domäne `d_zahlungsart` verwenden, die Sie vorher erzeugen müssen.

Hier also die Domäne `d_zahlungsart`:

```
CREATE DOMAIN d_zahlungsart
AS CHAR(1)
DEFAULT 'R'
CHECK (VALUE IN ('R', 'B', 'N', 'V', 'K'));
```

Nun können Sie die Tabelle `kunde_domaene` erzeugen:

```
CREATE TABLE kunde_domaene
(
   kundennr INTEGER NOT NULL,
   name VARCHAR(50),
   vorname VARCHAR(50),
   strasse VARCHAR(50),
   plz CHAR(14),
   ort VARCHAR(50),
   telefon_gesch VARCHAR(25),
```