

Michael Laube

CONSTRAINT

CREATE INDEX

ORDER BY

WHERE

Einstieg in SQL

Für
Ausbildung
und Beruf

CREATE
TABLE

SELECT * FROM

DELETE FROM

GROUP BY

CREATE VIEW

4. Auflage

- ▶ Datenbanken und SQL – ganz ohne Vorkenntnisse einsteigen
- ▶ SQL-Abfragen, Datenmodellierung, Tipps aus der Praxis
- ▶ Übungen und Musterlösungen für alle Datenbanksysteme



Inkl. Übungsdatenbank zum Download



Rheinwerk
Computing

Kapitel 1

Grundlagen kennenlernen und verstehen

Datenbanken und SQL haben in der Informatik die Aufgabe, Daten auf eine einfache Art und Weise zu verwalten. Zentrale Elemente sind die Abfragesprache SQL selbst, die Tabellen, in denen die Daten gespeichert werden, und schließlich die Datenbanksysteme, in denen die Tabellen und Daten hinterlegt sind.

Sie haben eine gute Wahl getroffen, als Sie sich entschieden haben, die Datenbankabfragesprache SQL zu lernen!

Datenbanken, die die Abfragesprache SQL unterstützen, sind weltweit verbreitet. Sie werden in zahlreichen Unternehmen und Institutionen verwendet, um Daten bzw. Informationen zu strukturieren und effektiv zu verwalten. Denken Sie nur an Versicherungen, Banken, Behörden und viele weitere Institutionen, die darauf angewiesen sind, Informationen sicher und dauerhaft zu verwalten.

Vermutlich interessiert Sie auch, wofür SQL überhaupt steht: *Structured Query Language*.

Structured Query Language (SQL)

S → Structured (engl. für »strukturiert«)

Q → Query (engl. für »Abfrage«)

L → Language (engl. für »Sprache«)

SQL ist eine *strukturierte Abfragesprache* für Datenbanken.



1.1 Die Tabelle als zentrales Element

Im Zentrum der Sprache SQL steht die *Tabelle*. Mit SQL können Sie folgende Hauptfunktionen auf eine Tabelle anwenden:

- Daten in eine Tabelle *einfügen*
- Daten aus einer Tabelle *abfragen*

- Daten in einer Tabelle *aktualisieren*
- Daten aus einer Tabelle *löschen*

In der Fachsprache werden diese Funktionen oftmals mit dem Begriff *CRUD* zusammengefasst, der für die englischen Wörter *Create*, *Read*, *Update* und *Delete* steht. Es sind die Grundfunktionen jedes Datenbanksystems.

Beginnen werden Sie in diesem Buch mit sogenannten **SELECT**-Abfragen, die auf eine Tabelle angewendet werden können. Die in ihrer Grundstruktur einfachen Abfragen ermöglichen es Ihnen, Daten einer Tabelle abzurufen bzw. abzufragen. Außerdem hält SQL alles bereit, was nötig ist, um Daten in Tabellen einzufügen, zu ändern und natürlich auch zu löschen.

Sie merken schon, die Tabelle ist das zentrale Element von SQL.

Tabellen und ihre Struktur

Wahrscheinlich haben Sie bereits einmal in einem anderen Zusammenhang mit Tabellen zu tun gehabt, beispielsweise mit Microsoft Excel. Dann wissen Sie, dass die Struktur einer Tabelle grundsätzlich sehr einfach ist: Eine Tabelle besteht aus *Zeilen* und *Spalten*. Eine Zeile, die auch als *Datensatz* bezeichnet wird, enthält Werte, die wiederum Spalten zugeordnet sind.

Betrachten wir zur Illustration ein Beispiel. In Abbildung 1.1 sehen Sie die Tabelle, die von der Öffentlichkeit vermutlich am meisten diskutiert wird: die Tabelle der ersten Fußball-Bundesliga.

PL. ^	TEAM	SP.	S	U	N	TORE	DIFF.	PUNKTE
1 -	 Bayern München	5	4	1	0	17:4	13	13
2 ^	 Eintracht Frankfurt	5	4	0	1	11:6	5	12
3 ^	 RB Leipzig	5	3	2	0	8:2	6	11
4 v	 Bayer 04 Leverkusen (M, P)	5	3	1	1	14:10	4	10
5 ^	 Borussia Dortmund	5	3	1	1	11:9	2	10
6 ^	 1. FC Heidenheim	5	3	0	2	10:7	3	9
7 v	 SC Freiburg	5	3	0	2	8:7	1	9
8 v	 VfB Stuttgart	5	2	2	1	14:10	4	8

Abbildung 1.1 Diese Tabelle enthält die essenziellen Daten zur schönsten Nebensache der Welt. (Quelle: <https://www.kicker.de/bundesliga/tabelle>)

Hier sehen Sie auf den ersten Blick, dass **EINTRACHT FRANKFURT** am 5. Spieltag hinter **BAYERN MÜNCHEN** steht und dass sich der **SC FREIBURG** auf einem starken 7. Platz befindet. Die Informationen, die uns die Spalten der Tabelle liefern, sind:

- ▶ PL. – der Platz, auf dem sich Ihr Lieblingsclub befindet
- ▶ TEAM – die Bezeichnung des Clubs
- ▶ SP. – die Anzahl der gespielten Spiele
- ▶ S – die Anzahl der Siege
- ▶ U – die Anzahl der unentschieden gespielten Spiele
- ▶ N – die Anzahl der Niederlagen
- ▶ TORE – die Anzahl der bisher erzielten Tore
- ▶ DIFF. – die Tordifferenz
- ▶ PUNKTE – die Anzahl der bis dato geholten Punkte

Die Spaltenbezeichnungen im Tabellenkopf, z. B. PL., TEAM, SP. und S, stehen für die *Eigenschaften* der Tabelle. Sie sorgen dafür, dass der Analyst die Zahl der Spiele und die Zahl der Niederlagen auseinanderhalten kann. Darunter stehen die dazugehörigen Zeilen, die die Tabelle mit Inhalt füllen. Einen *Wert* in einer Tabelle ermitteln Sie, indem Sie den Schnittpunkt der Zeile und der Spalte betrachten. Sie wollen wissen, welcher Club auf Platz 2 ist? Kein Problem: Betrachten Sie Zeile 2 und die Spalte TEAM, und ermitteln Sie den Schnittpunkt. Hier finden Sie den Wert EINTRACHT FRANKFURT.

Zusammenfassung: Definition einer Tabelle

Eine *Tabelle* besteht aus einem Tabellennamen, Zeilen und Spalten.

Der *Name* einer Tabelle fasst die Eigenschaften zusammen, die in einer Tabelle hinterlegt sind, und beschreibt, welche Informationen gespeichert werden.

Die *Eigenschaften* (Spalten) einer Tabelle bilden eine Struktur, in der Informationen geordnet gespeichert werden.

Die Informationen (die Daten) einer Tabelle sind in den Zeilen zu finden. Ein *Wert* innerhalb einer Zeile wird gelesen, indem der Schnittpunkt zwischen einer Zeile und einer Spalte ermittelt wird.



1.2 Eine kleine Historie von SQL

Eine Tabelle habe ich Ihnen bereits vorgestellt. Jetzt widmen wir uns der Frage, wer auf die Idee kam, Tabellen zur Speicherung großer Datenmengen zu nutzen, und auch die Grundlagen für SQL schuf.

Edgar F. Codd entwickelte in den 1970er-Jahren ein mathematisches Modell auf Grundlage der Mengenlehre. Die Struktur einer Tabelle wie der Bundesligatabelle kann auf dieser Grundlage beschrieben werden: Wir betrachten eine Menge. Diese

Menge bezeichnen wir als *Bundesligatabelle* und stattdessen sie mit Elementen wie *Platz* und *Verein* aus. Dies lässt sich in folgender Kurzschreibweise darstellen:

Bundesligatabelle={Pl,Verein,Sp,g,u,v,...}

Mathematisch betrachtet, bedeutet das nichts anderes als eine Definition der Struktur der Bundesligatabelle. In den geschweiften Klammern ist die Menge der Elemente (Spalten) aufgelistet, die für die Informationen der Bundesligatabelle erforderlich sind.

Mit der mathematischen Beschreibung einer Tabelle ist es in der Datenverarbeitung natürlich nicht getan. Denn das Ganze muss noch zum Leben erweckt werden.

Deshalb entwickelte Edgar F. Codd auch eine Sprache, mit der Daten basierend auf einfachsten Operationen abgefragt, eingefügt, gelöscht oder geändert werden können. Das Resultat dieser Überlegungen bildete die Grundlage für das *relationale Datenbankmodell* und die Sprache SQL. Die Arbeit von Edgar F. Codd wurde schließlich gekrönt, indem sie zu einem Standard erhoben wurde. Diese Standards werden heute von der *ISO (International Organization for Standardization)* festgelegt, erweitert und publiziert. In Abschnitt 1.4 erfahren Sie mehr über den SQL-Standard und seine Umsetzung durch unterschiedliche Datenbanksysteme.

1.3 Datenbanksysteme

Unterschiedlichste Anbieter haben zahlreiche Datenbanksysteme auf den Markt gebracht, die SQL unterstützen und Daten bzw. Informationen in Tabellen speichern. Datenbanksysteme, die Informationen in Tabellen speichern, werden als *relationale Datenbanksysteme* bezeichnet. Zu den bekanntesten Vertretern gehören:

- ▶ *IBM DB2*
- ▶ *Oracle DB*
- ▶ *Microsoft SQL Server*
- ▶ *Oracle MySQL*
- ▶ *MariaDB*
- ▶ *PostgreSQL*

Hierbei handelt es sich nur um eine kleine Auswahl von SQL-Datenbanken. Die Auswahl der Datenbanken habe ich bewusst getroffen, um Ihnen einerseits kommerzielle Anbieter und andererseits Open-Source-Vertreter von Datenbanksystemen vorstellen zu können.

Als führendes Datenbanksystem verwenden wir in diesem Buch MySQL. MySQL-Datenbanken sind sehr weit verbreitet. Das liegt unter anderem daran, dass die MySQL-

Datenbank kostenlos angeboten und oft im Verbund mit der Web-Programmiersprache *PHP* verwendet wird. Außerdem steht Ihnen die MySQL-Datenbank für die wichtigsten Betriebssysteme (beispielsweise unterschiedliche Linux-Distributionen, Windows-Systeme, Apple macOS) zur Verfügung. Ein weiterer Vorteil ist die umfangreiche Dokumentation, die Sie jederzeit online unter <http://dev.mysql.com/doc> abfragen können.

Falls Sie stattdessen lieber mit einer MariaDB-, PostgreSQL- oder einer Microsoft SQL Server-Datenbank lernen möchten, wird es Sie freuen zu erfahren, dass ich auf alle vier genannten Datenbanken eingehe und Ihnen jeweils eine Übungsdatenbank zur Verfügung stelle.

Die Datenbankserver MariaDB, PostgreSQL und Microsoft SQL Server sind ebenfalls sehr gut dokumentiert. Die Dokumentationen finden Sie unter:

- ▶ <https://mariadb.com/kb/en/documentation/>
- ▶ <http://www.postgresql.org/docs>
- ▶ <https://learn.microsoft.com/de-de/sql/sql-server/?view=sql-server-ver16>

MariaDB

MariaDB wurde im Jahr 2015 ins Leben gerufen, und es lohnt sich, einige Worte darüber zu verlieren. MariaDB ist ein sogenannter *Fork* von MySQL, also eine Abspaltung von diesem Projekt. Das MariaDB-Projekt bemüht sich, die volle Kompatibilität mit MySQL zu bewahren – die Unterschiede zwischen den beiden Datenbanken sind wirklich mit der Lupe zu suchen und betreffen hauptsächlich Lizenzfragen. (Fast) alle Erläuterungen, die Sie in diesem Buch zu MySQL finden, gelten daher ebenso für MariaDB. Sie finden das Projekt unter:

<https://mariadb.org>



1.4 SQL – ein Standard und seine Umsetzung

Der SQL-Standard der ISO wurde in der Vergangenheit als *ANSI-SQL-Standard* (ANSI: *American National Standards Institute*) bezeichnet. Der ISO-SQL-Standard soll auch international sicherstellen, dass Ihre Abfragen auf allen konformen Datenbanken zuverlässig funktionieren.

SQL: Ein Standard

Die Abfragesprache SQL ist im Standard ISO/IEC 9075-1:2016 festgelegt. Die Spezifikation kann unter www.iso.org gegen ein Entgelt bezogen werden.



Sie können davon ausgehen, dass unabhängig vom verwendeten Datenbanksystem der SQL-Sprachschatz zum größten Teil so umgesetzt wird, wie er im SQL-Standard beschrieben ist. Der kleinere Teil des Sprachumfangs, der nicht so realisiert wird, wie es der SQL-Standard vorsieht, wird als *SQL-Dialekt* bezeichnet. Alle drei hier behandelten Datenbanken haben einen eigenen Dialekt, der sich durch kleinere Unterschiede zum SQL-Standard-Sprachumfang bemerkbar macht.

Aus diesem Grund stelle ich Ihnen SQL anhand von vier unterschiedlichen Datenbanksystemen vor. Einerseits möchte ich Ihnen damit zeigen, dass SQL ein Standard ist und sich daher das Gelernte auf unterschiedliche Datenbanken anwenden lässt. Andererseits möchte ich Sie dafür sensibilisieren, dass SQL in einem kleinen Umfang abhängig vom jeweiligen Hersteller ist und damit im Detail unterschiedlich sein kann. Da es sich aber nur um kleine Unterschiede handelt, können Sie die hier vermittelten Grundlagen auch nutzen, um z. B. auf einer DB2-Datenbank von IBM oder einer Oracle-Datenbank mit SQL zu arbeiten.



SQL mit Hilfe von künstlicher Intelligenz lernen

Vielleicht kennen Sie schon KI-Helfer wie ChatGPT. Diese Werkzeuge können nicht nur Texte in normaler Sprache erstellen, sondern sie beherrschen auch Programmiersprachen wie SQL. Mit den richtigen Anfragen, den sogenannten *Prompts*, können Sie sich auch SQL-Aufrufe erstellen lassen und Informationen zu Ihrem Datenbanksystem erfragen.

Ich möchte Sie aber bitten, diese Möglichkeiten nur mit Bedacht und nicht verantwortungslos zu nutzen. Sie sollten keinesfalls KI-Code, den Sie nicht vollständig verstehen und geprüft haben, in Ihrer Datenbank nutzen. Denken Sie immer daran, dass die KI-Antworten fehlerhaft sein können und abhängig von den Eingaben sind. ChatGPT und andere Tools beruhen auf Modellen, die mit statistischen Wahrscheinlichkeiten der Sprache arbeiten. Damit lassen sich beeindruckende Ergebnisse erzielen, aber Ihr KI-Helfer »versteht« seine Aufgaben nicht wirklich und ist kein Ersatz für das eigene Lernen und Mitdenken.

Wenn Sie nach der Lektüre dieses Buchs die Grundlagen von SQL und Datenbanken gelernt haben, können Sie mit diesem Wissen die KI-Helfer sicherer nutzen und ihre Antworten prüfen und verstehen.



Zusammenfassung: SQL-Dialekte

SQL ist ein Standard, der durch die ISO (*International Organization for Standardization*) vorgegeben wird. Dennoch gibt es im Sprachschatz jeweils abhängig von der verwendeten Datenbank Abweichungen vom Standard. Diese Unterschiede werden als *SQL-Dialekte* bezeichnet.

1.5 Zu diesem Buch

Grundsätzlich gehen wir in diesem Buch vom SQL-Standard aus. Den Grundsprachschatz, den Sie im Rahmen dieses Einstiegs in SQL erlernen, können Sie universal auf allen SQL-Datenbanken anwenden. Bezogen auf die hier behandelten Datenbanksysteme werde ich Sie stets auf die kleinen Unterschiede aufmerksam machen. Und wenn Sie einmal auf den Fall stoßen, dass ein hier beschriebenes Beispiel bei Ihnen nicht funktioniert, weil Sie ein besonderes Feature einer bestimmten Datenbank einsetzen wollen: Glückwunsch, dann sind Sie wahrscheinlich kein Einsteiger mehr und spezialisieren sich auf ein Datenbanksystem eines Anbieters. Achten Sie in diesem Fall besonders auf die Dokumentation der von Ihnen eingesetzten Datenbank.

In Abschnitt 1.6 gebe ich Ihnen einen Leitfaden an die Hand, der Sie bei der Installation der MySQL-Datenbank unterstützt.

Der Leitfaden zur Installation enthält ausschließlich Hinweise, die für die Installation der jeweiligen Datenbank von Bedeutung sind. Von einer gewissen Affinität zu Windows- und Linux-Systemen gehe ich in diesem Fachbuch aus. Unter Windows können Sie die hier vorgestellten Datenbanken sehr einfach installieren. Ausführlicher werde ich Ihnen die Installation einer MySQL-Datenbank unter Windows beschreiben.

In den Downloadmaterialien zum Buch finden Sie das MySQL-Installationsprogramm in der folgenden Version hinterlegt:

- *MySQL 8.0.39 Community Edition, MySQL Workbench 8.0.38*

Falls Sie eine MariaDB-, PostgreSQL- oder MS SQL Server-Datenbank verwenden möchten, laden Sie die Installationsprogramme bitte direkt beim Hersteller herunter. Die entsprechende Installationsanleitung finden Sie in den Materialien zum Buch auf www.rheinwerk-verlag.de/6020. Beachten Sie, dass sich je nach Version Abweichungen ergeben können. Die Erklärungen in diesem Buch beziehen sich auf die Versionen:

- *MariaDB Server 11.5.2, HeidiSQL 12.8*
- *PostgreSQL 16.4, pgAdmin 4, v8.6*
- *MS SQL Server 2022 Express, SQL Server Management Studio 20.1*

In Abschnitt 1.7 zeige ich Ihnen anschließend, wie Sie die Übungsdatenbank auf den hier besprochenen Systemen anlegen und die Daten importieren.

In Abschnitt 1.8 erfahren Sie, wie Sie die MySQL Workbench nutzen, um mit SQL zu beginnen.

Sie werden in diesem Buch viele Kästen und SQL-Beispiele finden. In Hinweiskästen befinden sich besonders wichtige Tipps, die Sie sich unbedingt anschauen sollten.





Außerdem werde ich am Ende der Abschnitte die wichtigsten Informationen noch einmal für Sie zusammenfassen. Dies wird immer mit einem besonderen Icon gekennzeichnet.

In den SQL-Listings finden Sie sowohl eine kürzere einzeilige Schreibweise als auch die etwas längere, aber übersichtlichere Fassung, die sich über mehrere Zeilen erstreckt:

```
SELECT
    name,
    vorname,
    bonus
FROM
    mitarbeiter
WHERE bonus>500;
```

Listing 1.1 Ein auf mehrere Zeilen verteilter SQL-Befehl

Für die Datenbank bedeutet diese Anweisung genau das Gleiche wie der folgende Befehl:

```
SELECT name,vorname,bonus FROM mitarbeiter WHERE bonus>500;
```

Listing 1.2 Das gleiche Beispiel in der Kurzschreibweise

Wenn Sie nur rasch eine SQL-Anweisung ausprobieren wollen, reicht Ihnen wahrscheinlich die Kurzschreibweise aus. Bei längeren, komplexeren Anweisungen wird dies jedoch schnell unübersichtlich. Dann ist es hilfreich, die einzelnen Schritte in der übersichtlichen Schreibweise aufzuschlüsseln. Dabei ist es sinnvoll, jedes SQL-Schlüsselwort in eine Zeile zu schreiben. Egal, ob in der Kurzschreibweise oder der ausführlichen Darstellung: Die SQL-Schlüsselwörter finden Sie hier im Buch immer fett gedruckt und durch Großbuchstaben hervorgehoben. So können Sie sich rasch einen Überblick über die Beispiele verschaffen.

Was diese Schlüsselwörter genau bedeuten, erfahren Sie in den folgenden Kapiteln. Kümmern wir uns aber zunächst darum, die MySQL-Datenbank zu installieren.

1.6 MySQL unter Windows installieren

Um MySQL unter Windows zu installieren, führen Sie die folgenden Schritte aus:

Schritt 1: Laden Sie entweder den MySQL-Installer der MySQL 8.0.39 Community Edition von der Webseite zu diesem Buch unter www.rheinwerk-verlag.de/6020 herunter oder die aktuelle Version von der Website <https://dev.mysql.com/downloads/installer>. Achten Sie dabei darauf, dass das Auswahlfeld unter dem Punkt **SELECT**

OPERATION SYSTEM auf MICROSOFT WINDOWS eingestellt ist, um den MSI-Installer herunterladen zu können. Hier werden zwei Downloads angeboten: zum einem ein Web-Installer und zum anderen ein vollständiger Installer, der alles enthält, was Sie zur Installation brauchen. Wählen Sie den vollständigen Installer aus. Zu erkennen ist er einerseits am Namen und andererseits an der Größe der Installationsdatei von mehr als 310 MB. Die Anmeldeinformation auf der folgenden Seite können Sie gestrost ignorieren. Klicken Sie einfach weiter unten auf der Seite auf NO THANKS, JUST START MY DOWNLOAD, um die Datei ohne Anmeldung herunterzuladen.

Schritt 2: Öffnen Sie als Nächstes das Downloadverzeichnis, und führen Sie einen Doppelklick auf die soeben heruntergeladene Installationsdatei (mit der Endung *.msi*) aus. Falls Ihnen ein Dialogfenster mit einer Warnung angezeigt wird, in der Sie gefragt werden, ob Sie die Datei tatsächlich ausführen wollen, bestätigen Sie sie mit JA. Sollten während des Installationsprozesses weitere Warnmeldungen auftauchen, bestätigen Sie auch diese entsprechend.

Schritt 3: Im Fenster CHOOSING A SETUP TYPE sehen Sie die fünf möglichen Installationsvarianten (siehe Abbildung 1.2). Wählen Sie die Variante CUSTOM. Diese ermöglicht, nur die Softwarekomponenten auszuwählen, die Sie für den Einstieg in SQL benötigen. Klicken Sie auf NEXT, um zum nächsten Installationsschritt zu gelangen.

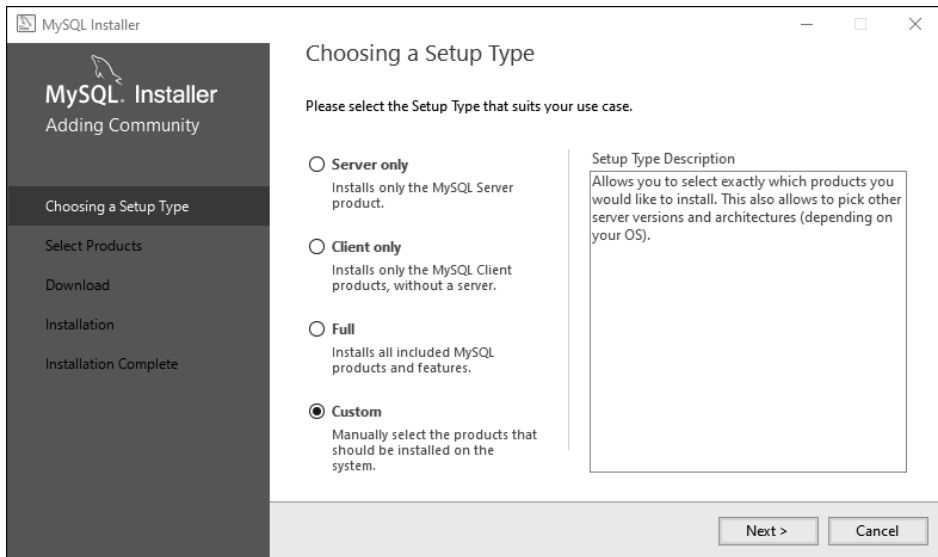


Abbildung 1.2 Installationstyp auswählen

Schritt 4: Im Fenster SELECT PRODUCTS (siehe Abbildung 1.3) sehen Sie eine Auswahl der Softwarekomponenten, die Sie installieren können.

Im linken Bereich mit der Überschrift AVAILABLE PRODUCTS ist ein Navigationsbaum zu sehen. Klicken Sie einfach auf den Knoten mit dem Pluszeichen, um den Baum so

weit aufzuklappen, dass die benötigten Softwarekomponenten auswählbar sind. Wählen Sie **MYSQ L SERVERS • MYSQ L SERVERS • MYSQ L SERVER 8.0 • MYSQ L SERVER 8.0.39–X64**. Klicken Sie im Anschluss auf den grün gewordenen Pfeil nach rechts, der sich zwischen den beiden Fenstern befindet, um das Produkt in das Fenster **PRODUCTS TO BE INSTALLED** zu verschieben.

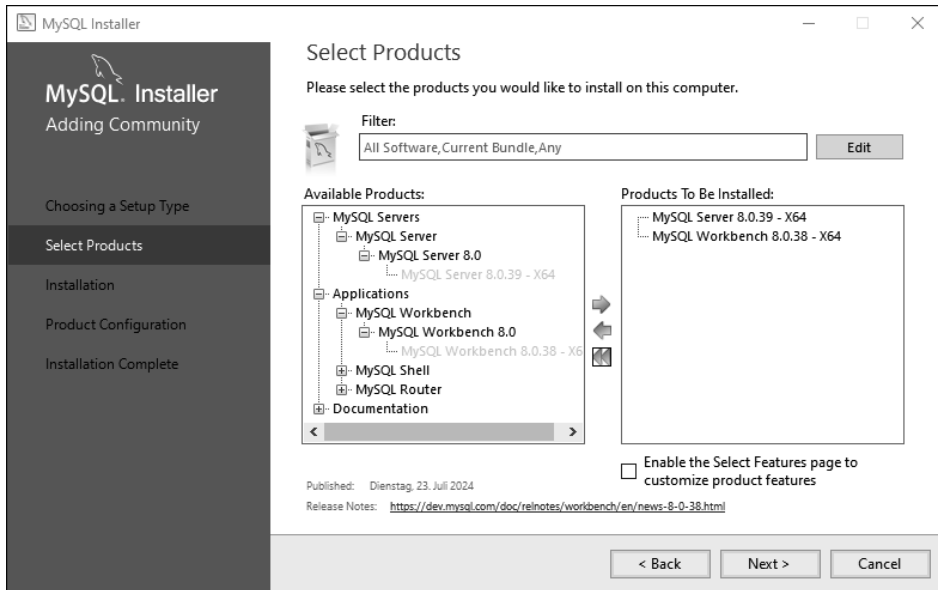


Abbildung 1.3 Zu installierende Komponenten auswählen

Wiederholen Sie diesen Schritt für **APPLICATIONS • MYSQ L WORKBENCH • MYSQ L WORKBENCH 8.0 • MYSQ L WORKBENCH 8.0.39–X64**.

Sollten neuere Versionen von MySQL Server und MySQL Workbench zur Verfügung stehen, wählen Sie bitte diese anstelle der hier genannten.

Klicken Sie anschließend auf den Button **NEXT**, um mit der Installation fortzufahren.

Schritt 5: Im Anschluss sehen Sie das Fenster **INSTALLATION**. Darin werden nochmals die Softwareprodukte angezeigt, die Sie zur Installation ausgewählt haben. Mit einem Klick auf **EXECUTE** werden sie installiert. Sobald die Installation erfolgreich war, werden vor den Softwarekomponenten grüne Häkchen angezeigt, und die **STATUS**-Spalte zeigt **COMPLETE** an (siehe Abbildung 1.4). Klicken Sie erneut auf den Button **NEXT**, um zum nächsten Schritt zu gelangen.

Schritt 6: Jetzt befinden Sie sich im Fenster **PRODUCT CONFIGURATION**. Hier werden die Softwareprodukte aufgelistet, die zur Konfiguration bereitstehen. Da Sie nichts weiter tun können, klicken Sie direkt auf **NEXT**.

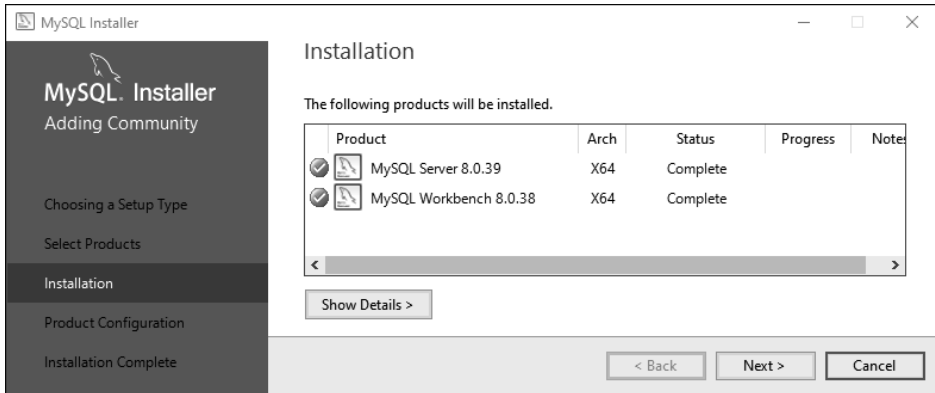


Abbildung 1.4 Die Installation der MySQL Server- und MySQL Workbench-Software wird bestätigt.

Schritt 7: Nach einem erneuten Klick auf NEXT öffnet sich das Fenster TYPE AND NETWORKING (siehe Abbildung 1.5). Der Konfigurationsdialog enthält einige voreingestellte Netzwerkeinstellungen. Ich empfehle Ihnen, diese beizubehalten und direkt auf NEXT zu klicken, um fortzufahren.

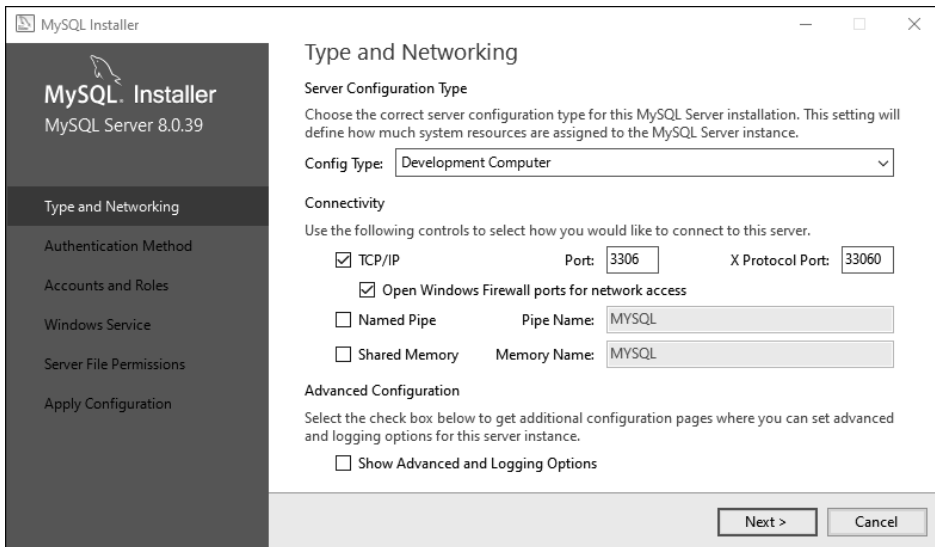


Abbildung 1.5 Typ und Netzwerkeinstellungen konfigurieren

Schritt 8: Im Fenster AUTHENTICATION METHOD (siehe Abbildung 1.6) können Sie zwischen zwei Optionen wählen. Belassen Sie es auch hier bei der Standardauswahl, und klicken Sie auf den Button NEXT, um die Installation fortzuführen.

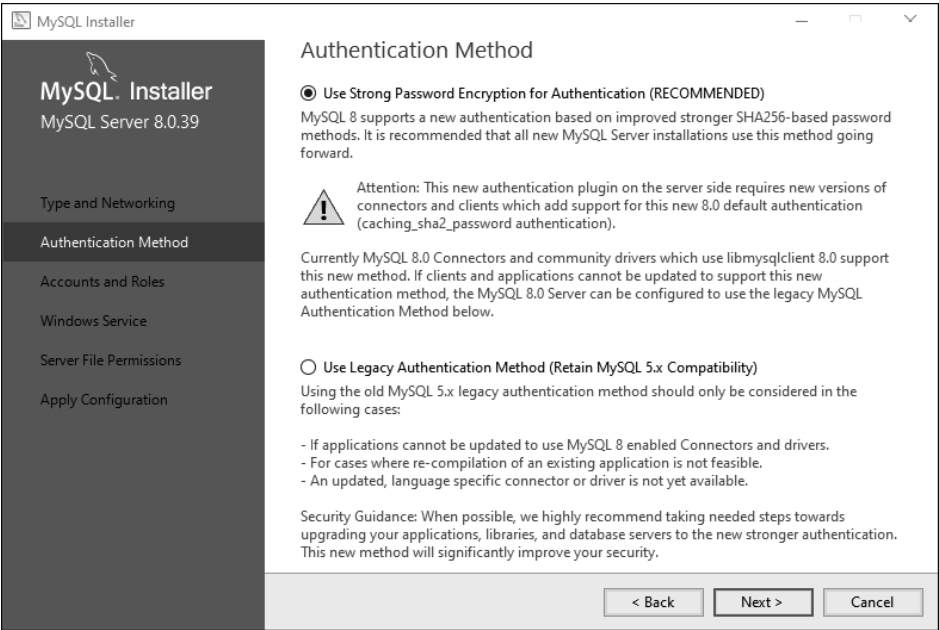


Abbildung 1.6 Die Authentifizierungsmethode auswählen

Schritt 9: Jetzt müssen Sie noch ein Passwort für unseren Superuser (*root*) festlegen. Geben Sie ein Passwort Ihrer Wahl in das Texteingabefeld für **MYSQL ROOT PASSWORD** ein, und bestätigen Sie es, indem Sie es noch einmal in das Texteingabefeld **REPEAT PASSWORD** eingeben.

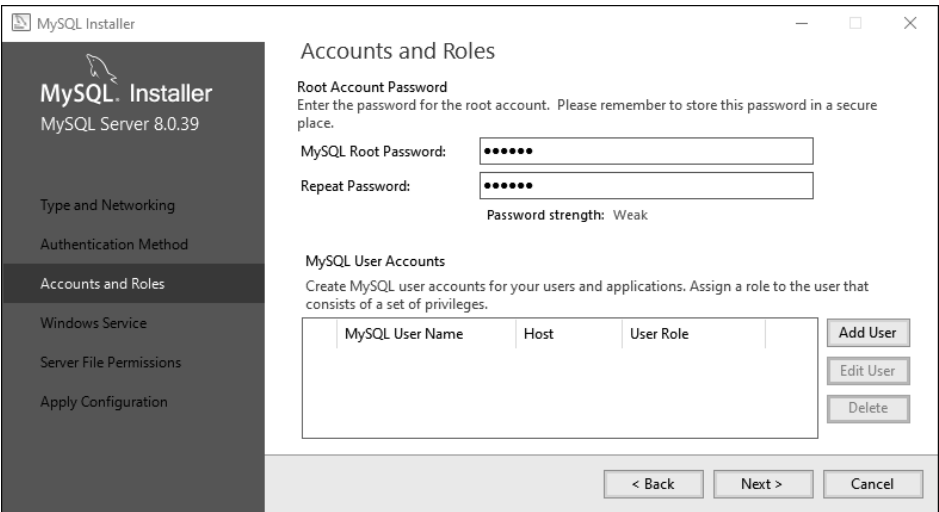


Abbildung 1.7 Ein Passwort für den Superuser (»root«) vergeben

Der Einfachheit halber empfehle ich Ihnen für den Übungsbetrieb, ein simples Passwort auszuwählen. Klicken Sie wieder auf den Button NEXT, um zum nächsten Installationsschritt zu gelangen.

Hinweis zur Nutzung des Administrators »root«

Im Produktivbetrieb einer Datenbank würden Sie aus Sicherheitsgründen natürlich niemals mit dem administrativen Nutzer *root* arbeiten, wenn Sie SQL-Abfragen durchführen. Für den Übungsbetrieb können Sie jedoch getrost den Administrator *root* verwenden. Mehr zu Benutzern, Rollen und Berechtigungen erfahren Sie in Kapitel 6.

Schritt 10: In Abbildung 1.8 sehen Sie das Konfigurationsfenster WINDOWS SERVICE. Der MySQL-Datenbankserver schlägt vor, sich als Windows-Dienst einzurichten. Idealerweise belassen Sie es dabei und klicken auf den Button NEXT, um fortzufahren.

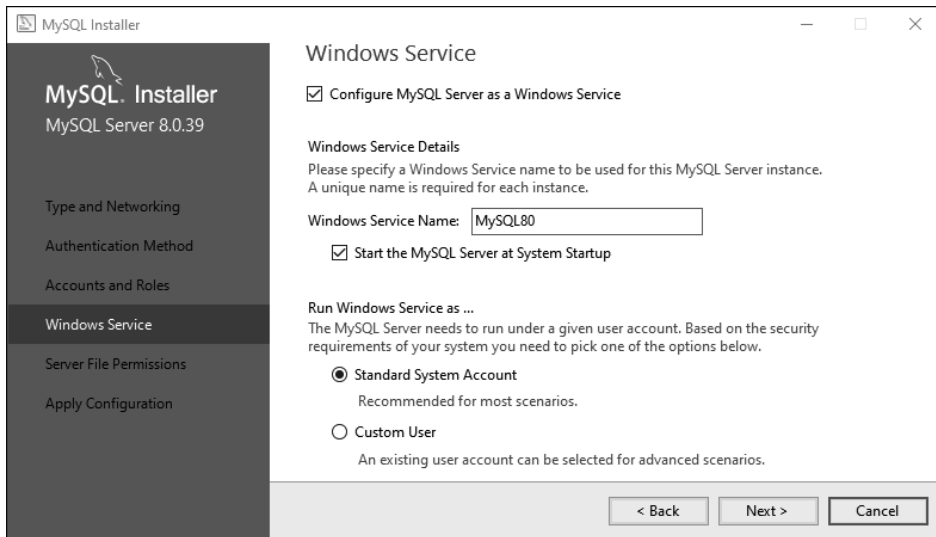


Abbildung 1.8 MySQL Server als Windows-Dienst einrichten

Schritt 11: Als Nächstes bestätigen Sie im Konfigurationsfenster SERVER FILE PERMISSIONS (Abbildung 1.9) eine Berechtigungsänderung für das Datenverzeichnis der MySQL-Datenbank, die durch das Installationsprogramm vorgenommen wird. Bitte belassen Sie diese Einstellung so wie vorgegeben, und klicken Sie auf den Button NEXT, um fortzufahren.

Schritt 12: Im Fenster APPLY CONFIGURATION müssen Sie mit einem Klick auf EXECUTE nur noch der Konfiguration zustimmen.

Gleich im Anschluss bestätigt Ihnen das Installationsprogramm mit grünen Häkchen, welche Konfigurationsschritte bereits ausgeführt worden sind (siehe Abbil-

dung 1.10). Sobald alle Schritte durchgeführt wurden, können Sie auf **FINISH** klicken, um die Konfiguration abzuschließen.

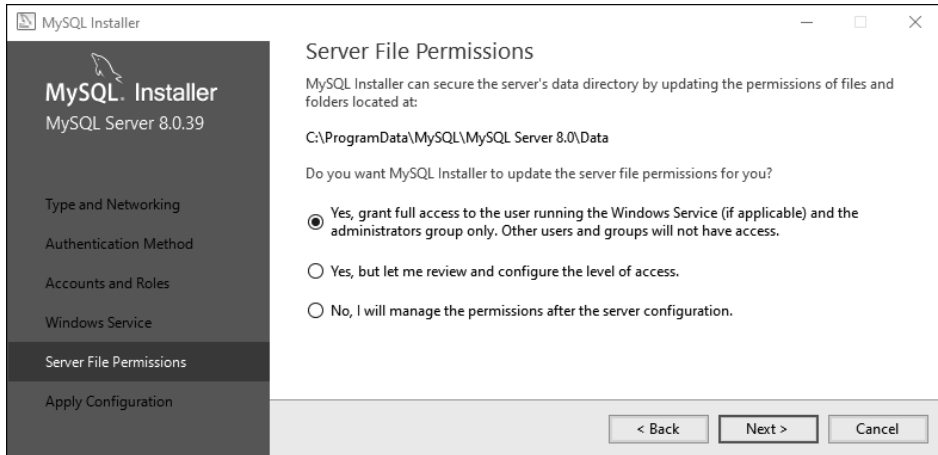


Abbildung 1.9 Bestätigung der Berechtigungseinstellung

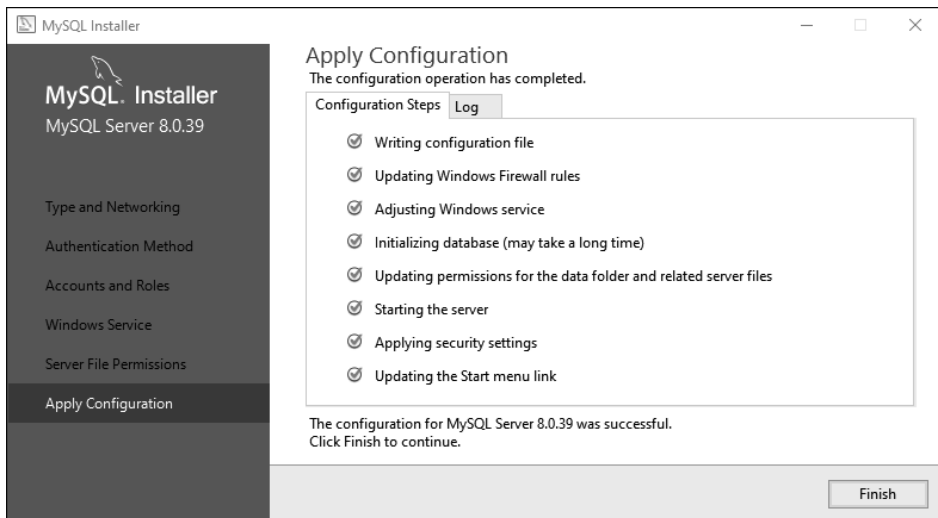


Abbildung 1.10 Bestätigung der Konfiguration

Abschließend erhalten Sie eine Bestätigung über die erfolgreiche Installation und Konfiguration von MySQL Server. Klicken Sie abermals auf **NEXT**.

Jetzt haben Sie es endlich geschafft! MySQL Server und die MySQL Workbench wurden installiert, und sämtliche erforderlichen Einstellungen wurden vorgenommen.

Aktivieren Sie, falls es noch nicht ausgewählt sein sollte, im nächsten Fenster das Kontrollkästchen **START MYSQL WORKBENCH AFTER SETUP** (siehe Abbildung 1.11).

Nun müssen Sie nur noch auf **FINISH** klicken, um die Installation abzuschließen und die MySQL Workbench zu starten.

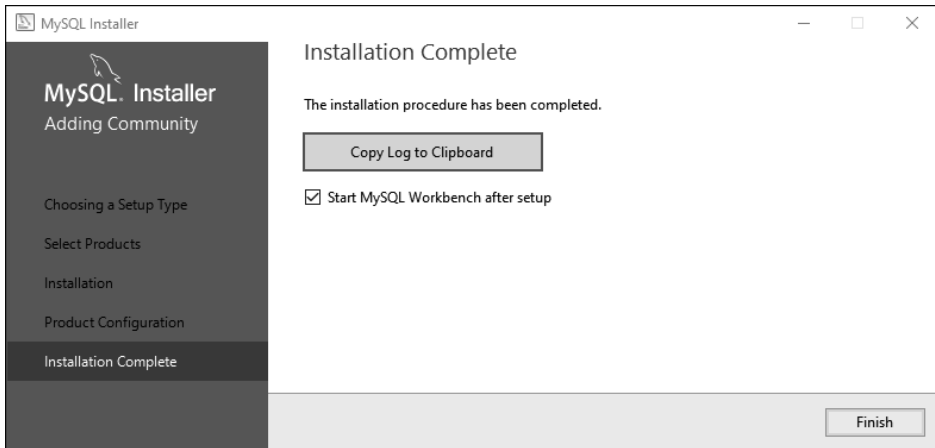


Abbildung 1.11 Bestätigung der Konfiguration

Im folgenden Willkommensfenster der MySQL Workbench sollten Sie darauf achten, im linken Navigationsbereich nur das Symbol mit dem Delphin auszuwählen (siehe Abbildung 1.12).

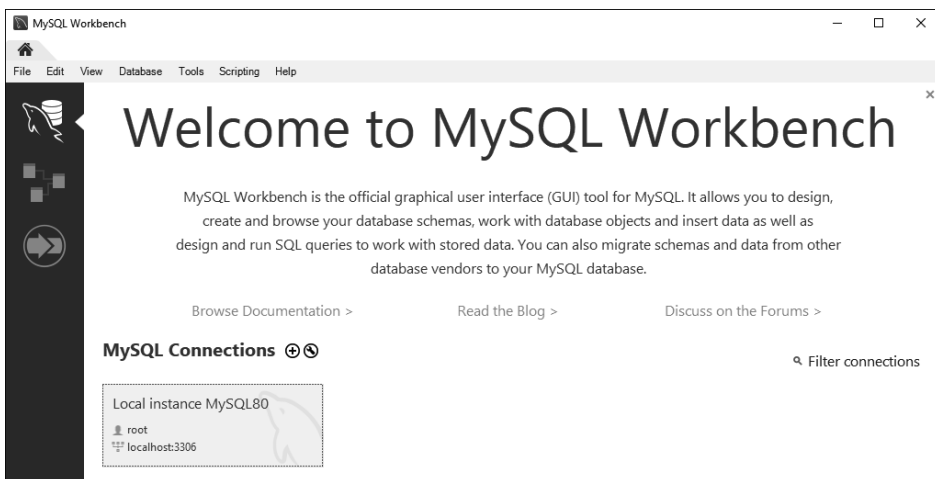


Abbildung 1.12 Die MySQL Workbench nach dem Start

Im Hauptfenster wird Ihnen eine grau hinterlegte Fläche mit der Bezeichnung **LOCAL INSTANCE** angezeigt, auf die eine MySQL-Versionbezeichnung folgt. Wenn Sie auf diese Fläche klicken, öffnet sich nach der Eingabe des Passworts für den Benutzer **root** die Editoransicht. Sie ist im Prinzip ein Texteditor, in dem Sie SQL-Anweisungen eingeben und ausführen können.

1.7 Die MySQL-Übungsdatenbank anlegen

An dieser Stelle gehe ich davon aus, dass Sie, wie in Abschnitt 1.6 beschrieben, die MySQL-Datenbank installiert haben.

Die Übungsdatenbank trägt den selbsterklärenden Namen *uebungsdatenbank*. Beachten Sie, dass der Name vollständig kleingeschrieben ist. Der deutsche Umlaut »ü« wurde durch die Buchstaben »ue« ersetzt, da sich ansonsten Probleme mit Zeichensätzen ergeben können. Daher gehört es in der Programmierung zum guten Stil, Bezeichner stets ohne Umlaute zu verwenden.

Eine wichtige Bemerkung vorab: Um die Übungsdatenbank anzulegen, benötigen Sie *Administratorrechte* für die verwendete Datenbank. Wenn Sie der Installationsanleitung in Abschnitt 1.6 gefolgt sind, besitzen Sie bereits Administratorrechte. Während der Installation haben Sie für den Benutzer *root* ein Passwort vergeben.



Administratoren unter MySQL

Unter MySQL heißt der Nutzer, der über administrative Rechte verfügt, *root*.

Um die Übungsdatenbank erfolgreich anzulegen, benötigen Sie Administratorrechte für die Datenbank oder entsprechende Berechtigungen, die es Ihnen erlauben, Datenbanken oder Tabellen anzulegen und natürlich Daten dort einzufügen.

Kommen wir jetzt zu den Vorbereitungen, die für das Importieren der Übungsdatenbank notwendig sind. Laden Sie die Importskripte für die Übungsdatenbanken herunter. Beachten Sie, dass die Downloaddatei die Importskripte für alle vier Datenbanken enthält, die ich hier vorstelle.



Download der Importskripte

Sie finden die Importskripte auf der Webseite des Rheinwerk Verlags zu diesem Buch unter www.rheinwerk-verlag.de/6020.

Wählen Sie dort den Tab mit der Bezeichnung MATERIALIEN ZUM BUCH. Die Importskripte stehen jeweils als *ZIP*-Datei zum Download bereit. Bitte laden und entpacken Sie diese Datei. Das Entpacken der *ZIP*-Datei kann beispielsweise in Ihrem *Downloads*-Ordner erfolgen.

In dem soeben entpackten Verzeichnis *mysql* finden Sie die folgenden Dateien:

- ▶ *importskript.sql*
- ▶ *importskript_arbeitszeit.sql*
- ▶ *importskript_kreditinstitut.sql*

Wir benötigen zunächst nur die Datei *importskript.sql*, mit der wir die Übungsdatenbank anlegen.

Abbildung 1.13 zeigt das geöffnete Fenster der MySQL Workbench. Klicken Sie auf das VERZEICHNIS ÖFFNEN-Symbol unterhalb der Menüüberschriften.

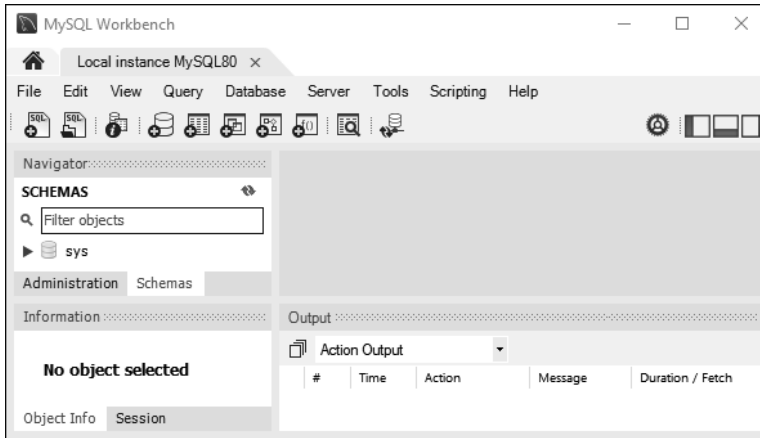


Abbildung 1.13 Die MySQL Workbench

Wählen Sie im Ordner *Downloads\mysql* die Datei *importsript.sql* aus (siehe Abbildung 1.14), und öffnen Sie sie mit einem Klick auf ÖFFNEN.

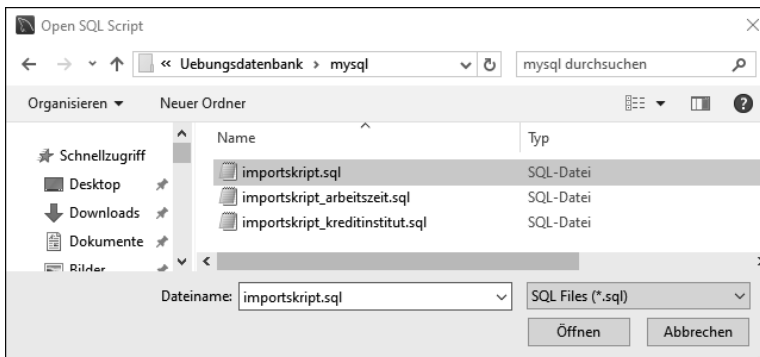


Abbildung 1.14 Das Importskript auswählen und öffnen

Stellen Sie nun sicher, dass sich der Cursor im geöffneten Importskript an der ersten Position der ersten Zeile befindet (siehe Abbildung 1.15). Klicken Sie danach auf den gelben Blitz, um alle Anweisungen, die im Importskript enthalten sind, auszuführen und die Übungsdatenbank anzulegen.

In Abbildung 1.16 sehen Sie im Fenster ACTION OUTPUT, dass die Anweisungen, die für den Import erforderlich sind, ausgeführt wurden.

Nun müssen wir die Liste der Datenbanken aktualisieren – diese werden im MySQL-Sprachgebrauch auch *Schema* genannt. Klicken Sie hierfür auf den Reiter SCHEMAS, der sich unterhalb des NAVIGATOR-Fensters befindet. Nun sehen Sie im NAVIGATOR-

Fenster rechts neben der Überschrift SCHEMAS ein Aktualisierungssymbol. Wenn Sie darauf klicken, wird Ihnen die importierte Datenbank mit ihren Tabellen angezeigt.

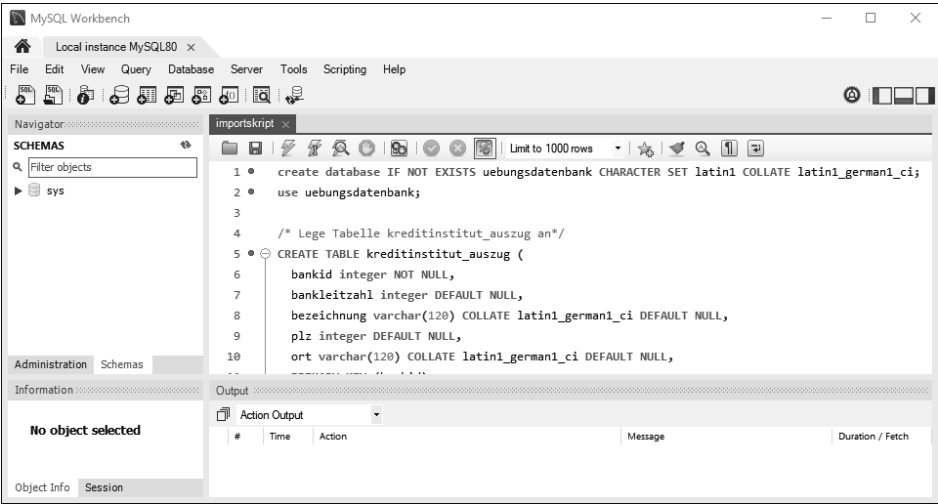


Abbildung 1.15 Das geöffnete Importskript in der MySQL Workbench

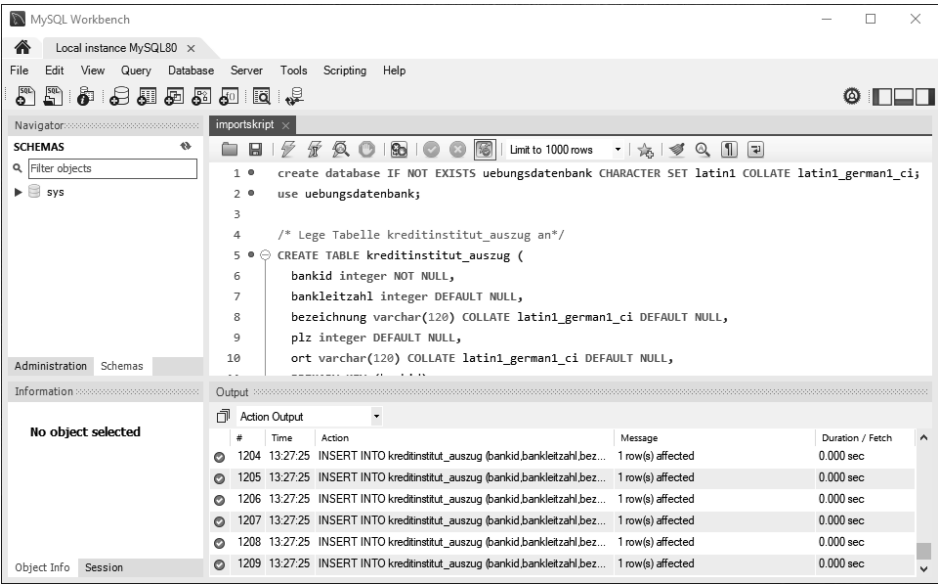


Abbildung 1.16 Das Importskript wurde ausgeführt.

Die aktive Datenbank wird im NAVIGATOR-Fenster wie in Abbildung 1.17 immer in Fettschrift dargestellt. Stellen Sie zu Beginn Ihrer Arbeit stets sicher, dass Sie im NAVIGATOR-Fenster auf die *uebungsdatenbank* geklickt haben und diese fettgedruckt angezeigt wird.

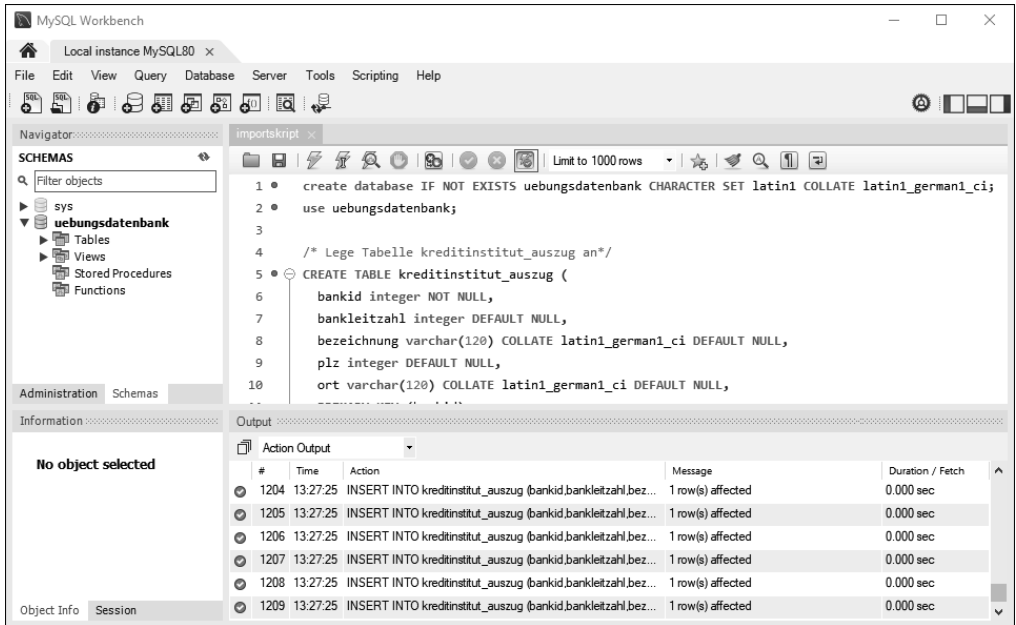


Abbildung 1.17 Das Navigator-Fenster wurde aktualisiert.

Wenn Sie zu einem späteren Zeitpunkt mit mehreren Datenbanken auf Ihrem MySQL-Datenbanksystem arbeiten, können Sie mit der MySQL Workbench auch auf sie entsprechend zugreifen.

Schließen Sie nun das geöffnete Importskript, indem Sie auf das Kreuz neben dem Tab IMPORTSKRIPT klicken. Alternativ können Sie auch die Tastenkombination **Strg** + **W** nutzen, um das Editorfenster zu schließen.

1.8 Eine erste Abfrage an die Datenbank senden

Wir werden nun eine erste einfache Abfrage der Tabelle *mitarbeiter* ausführen. Da wir in Abschnitt 1.7 das Editorfenster geschlossen haben, müssen Sie ein neues Editorfenster in der MySQL Workbench aufrufen. In Abbildung 1.18 sehen Sie die Menüleiste der MySQL Workbench. Unter den Menüpunkten sehen Sie ein paar Icons:

- ❶ Das erste Icon von links enthält ein Pluszeichen. Sie verwenden es, wenn Sie beabsichtigen, eine neue Datei zu erstellen, in der Sie Ihre SQL-Anweisungen notieren und ausführen.
- ❷ Das zweite Icon zeigt eine geöffnete Mappe. Hiermit öffnen Sie eine vorhandene SQL-Datei. Sie haben es bereits kennengelernt, als wir das Importskript geöffnet haben.

- ❸ Das dritte Icon enthält ein »i« wie »Information«. Wenn Sie z. B. im Navigationsbaum eine Datenbank markieren und auf dieses Icon klicken, so erhalten Sie ausführliche Informationen wie die Dateigröße oder in Ihrer Datenbank enthaltene Tabellen.
- ❹ Das vierte Icon symbolisiert eine Datenbank. Mit ihm können Sie eine neue Datenbank (ein neues Schema) anlegen.
- ❺ Das fünfte Symbol zeigt eine Tabelle mit Pluszeichen. Mit ihm legen Sie eine neue Tabelle für eine ausgewählte Datenbank an.

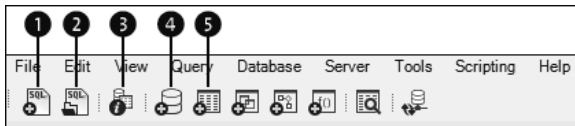


Abbildung 1.18 Die Menüleiste der MySQL Workbench

Nun wollen wir eine SQL-Abfrage an die Datenbank richten. Hierzu benötigen wir ein neues Editorfenster, das Sie mit einem Klick auf das erste Icon erzeugen. Alternativ können Sie auch das Tastaturkürzel `[Strg] + [T]` verwenden.

Am oberen Rand des Editorfensters sehen Sie nun wie in Abbildung 1.19 eine weitere Leiste mit Icons.

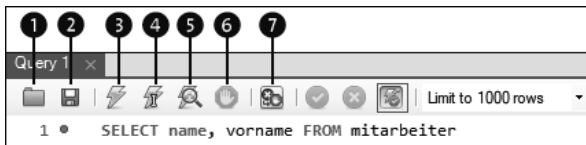


Abbildung 1.19 Eine SELECT-Abfrage ausführen

- ❶ Mit dem Ordnersymbol-Icon öffnen Sie eine gespeicherte *.sql*-Datei, in der Sie SQL-Anweisungen notiert haben.
- ❷ Mit dem Disketten-Icon speichern Sie Ihre aktuell bearbeiteten SQL-Anweisungen in einer Datei mit der Endung *.sql*.
- ❸ Das Blitz-Icon dient zur Ausführung von SQL-Anweisungen. Diese Version des Blitzes führt alle SQL-Anweisungen aus, die im SQL-Editor notiert wurden, es sei denn, Sie haben explizit eine Anweisung mit der Maus markiert.
- ❹ Das zweite Blitz-Icon (mit einem Cursor-Symbol) führt die SQL-Anweisung aus, die sich unterhalb des Cursors befindet.
- ❺ Das Blitz-Icon mit einer Lupe führt die SQL-Anweisung aus und liefert Ihnen eine Beschreibung dessen, was die Datenbank mit dieser Anweisung an Arbeitspaketen zu bewältigen hat. Dieser Punkt ist für Sie im Augenblick nicht von Bedeutung.

- ⑥ Das Symbol mit der Hand bricht eine länger andauernde SQL-Anweisung ab. Diese Funktion werden Sie erst einmal nicht benötigen. Sie werden sie aber zu schätzen lernen, wenn Sie mit großen Datenvolumen zu tun haben.
- ⑦ Über das Symbol eines Abbruchzeichens mit einer stoppenden Hand bestimmen Sie, ob SQL-Anweisungen weiter ausgeführt werden sollen, wenn eine SQL-Anweisung einen Fehler während der Ausführung zurückliefert.

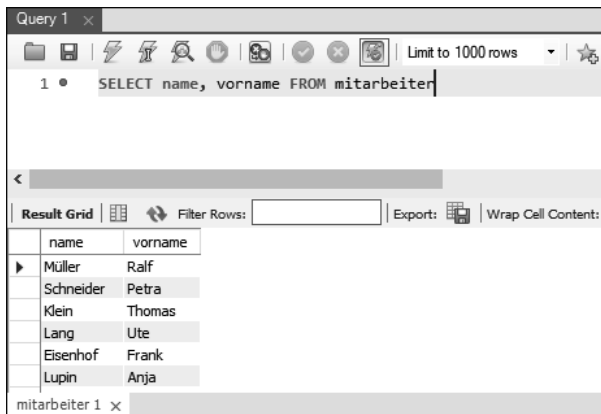
Steigen wir gleich ein, und probieren wir eine Anweisung in einem Beispiel aus. Geben Sie dazu die SQL-Anweisung aus Listing 1.3 in Ihren Editor ein. Sie müssen an dieser Stelle noch nicht verstehen, was diese **SELECT**-Anweisung bewirkt, denn es kommt erst einmal nur darauf an, wie Sie SQL-Anweisungen eingeben und ausführen, um ein Ergebnis von der Datenbank zu erhalten.

```
SELECT name,vorname FROM mitarbeiter;
```

Listing 1.3 Eine SQL-Anweisung an die Datenbank richten

Jetzt müssen wir die im SQL-Editor notierte SQL-Anweisung nur noch an die Datenbank senden, um ein Ergebnis zu erhalten. Hierzu nutzen Sie die Icon-Leiste über dem Eingabeeditor, die Sie ja bereits kennengelernt haben. Nutzen Sie das Blitz-Icon ③ aus Abbildung 1.19, um die SQL-Anweisung, die im SQL-Editor notiert ist, an die Datenbank zu richten. Anstelle des Symbols können Sie alternativ auch eine Tastenkombination verwenden, um Ihre SQL-Anweisungen an das MySQL-Datenbanksystem zu senden: Halten Sie die Taste `[Strg]` gedrückt, und betätigen Sie die `[↵]`-Taste, um die SQL-Anweisung im Editor auszuführen.

Führen Sie also Ihre erste SQL-Anweisung aus, indem Sie sie an den MySQL-Datenbankserver senden. Sie erhalten dann von der Datenbank eine Ergebnistabelle zurück (siehe Abbildung 1.20).



The screenshot shows a SQL query editor window titled "Query 1". The query entered is "SELECT name, vorname FROM mitarbeiter;". Below the query, there is a toolbar with various icons, including a lightning bolt icon (used for executing the query). The results are displayed in a table with the following data:

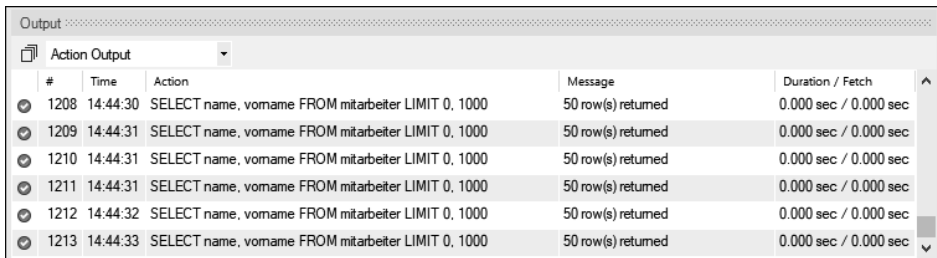
name	vorname
Müller	Ralf
Schneider	Petra
Klein	Thomas
Lang	Ute
Eisenhof	Frank
Lupin	Anja

At the bottom of the window, the text "mitarbeiter 1 x" is visible.

Abbildung 1.20 Ergebnistabelle nach der Ausführung einer SQL-Anweisung

Unterhalb der Ergebnistabelle sehen Sie ein Ausgabefenster wie in Abbildung 1.21. In diesem Fenster wird Ihnen das Feedback der Datenbank zu den jeweiligen SQL-Anweisungen mitgeteilt. Der Haken in der ersten Spalte zeigt Ihnen, dass die jeweilige SQL-Anweisung erfolgreich ausgeführt wurde. Insgesamt hält das Ausgabefenster folgende Informationen für Sie bereit:

- den Zeitpunkt der Ausführung (TIME)
- die SQL-Anweisung, die ausgeführt wurde (ACTION)
- eine Nachricht, die z. B. ausgibt, wie viele Zeilen zurückgeliefert wurden (MESSAGE)
- die Dauer (DURATION / FETCH), die zur Ausführung der SQL-Anweisung benötigt wurde



The screenshot shows the 'Output' window in MySQL Workbench, specifically the 'Action Output' tab. It displays a table with five columns: a checkbox, '#', 'Time', 'Action', 'Message', and 'Duration / Fetch'. There are six rows of data, all showing successful execution of a SELECT query with 50 rows returned and a duration of 0.000 sec.

	#	Time	Action	Message	Duration / Fetch
<input checked="" type="checkbox"/>	1208	14:44:30	SELECT name, vomame FROM mitarbeiter LIMIT 0, 1000	50 row(s) returned	0.000 sec / 0.000 sec
<input checked="" type="checkbox"/>	1209	14:44:31	SELECT name, vomame FROM mitarbeiter LIMIT 0, 1000	50 row(s) returned	0.000 sec / 0.000 sec
<input checked="" type="checkbox"/>	1210	14:44:31	SELECT name, vomame FROM mitarbeiter LIMIT 0, 1000	50 row(s) returned	0.000 sec / 0.000 sec
<input checked="" type="checkbox"/>	1211	14:44:31	SELECT name, vomame FROM mitarbeiter LIMIT 0, 1000	50 row(s) returned	0.000 sec / 0.000 sec
<input checked="" type="checkbox"/>	1212	14:44:32	SELECT name, vomame FROM mitarbeiter LIMIT 0, 1000	50 row(s) returned	0.000 sec / 0.000 sec
<input checked="" type="checkbox"/>	1213	14:44:33	SELECT name, vomame FROM mitarbeiter LIMIT 0, 1000	50 row(s) returned	0.000 sec / 0.000 sec

Abbildung 1.21 Statusmeldungen zu den SQL-Anweisungen



Gebräuchliche Tastaturkürzel der MySQL Workbench

Ich verwende gern Tastaturkürzel, da sie ein effektiveres Arbeiten ermöglichen. Aus diesem Grund stelle ich Ihnen die hier verwendeten Tastaturkürzel kurz zusammengefasst vor:

[Strg] + [O] – ein bestehendes Skript öffnen

[Strg] + [T] – einen neuen Tabulator (Editorfenster) öffnen

[Strg] + [S] – ein Skript speichern

[Strg] + [W] – ein Editorfenster schließen

[Strg] + [↵] – eine SQL-Anweisung ausführen

1.9 Kommentarfunktion

Sie sollten Ihre SQL-Anweisungen, die Sie dauerhaft speichern möchten, ausreichend mit erklärenden Notizen versehen. Besonders bei komplexeren SQL-Anweisungen ist das sehr hilfreich: Nur so können Sie auch nach längerer Zeit noch nachvollziehen, wie Sie die Anweisung programmiert und auf welchen Wegen Sie das Ergebnis herbeigeführt haben.

Ihre Kommentare können dabei über verschiedenste Dinge informieren. Als Beispiel liste ich Ihnen einige Informationen auf, die Ihren SQL-Code bereichern und Ihnen in Zukunft die Arbeit damit vereinfachen werden:

- ▶ die anfordernde Stelle
- ▶ die Funktion der Abfrage
- ▶ das Datum der Fertigstellung
- ▶ die Durchführung von Änderungen und der Grund der Änderung
- ▶ der Name des Programmierers

Grundsätzlich verfügt SQL über zwei unterschiedliche Arten von Kommentaren. So gibt es einzeilige Kommentare, die mit einem doppelten Bindestrich (--) eingeleitet werden, auf den der Text des Kommentars folgt. Daneben gibt es Kommentare, die sich über mehrere Zeilen erstrecken können. Diese Kommentare beginnen mit den Zeichen /* und enden mit den Zeichen */. Alles, was dazwischen steht, wird vom Datenbanksystem ignoriert.

Kommentare in der Praxis nutzen

Sehen wir uns an, wie Sie einen einzeiligen Kommentar in der Praxis verwenden. Hierzu setzen wir einfach vor der **SELECT**-Abfrage einen einzeiligen Kommentar wie in Listing 1.4 ein. Wenn Sie die Anweisung in Ihrem SQL-Client ausführen, ignoriert das Datenbanksystem den Kommentar und liefert Ihnen stattdessen nur das Ergebnis der **SELECT**-Abfrage als Ergebnistabelle zurück.

```
-- Spaltenauswahl: name, vorname der Tabelle mitarbeiter  
SELECT name,vorname FROM mitarbeiter;
```

Listing 1.4 Einen einzeiligen Kommentar im SQL-Code anwenden

Einzeilige Kommentare sind zu Dokumentationszwecken nur eingeschränkt verwendbar, da Ihnen nur eine Zeile zur Verfügung steht. Aber dafür gibt es ja die mehrzeiligen Kommentare. Die nächste **SELECT**-Anweisung dokumentieren wir in Listing 1.5 mit einem mehrzeiligen Kommentar, um sie mit verschiedenen Informationen anzureichern. Erläutert wird so, wer der Autor war, wann der Befehl erstellt wurde, und es folgen noch einige weitere nützliche Informationen.

```
/*  
Autor: Michael Laube  
Erstellungsdatum: 01.08.2024  
Funktion: Liste mit den Namen und Vornamen der Mitarbeiter  
Funktion geprüft: ja  
Funktion erfüllt: ja
```


Nutzergruppen der Abfrage: Personalabteilung

*/

```
SELECT name,vorname FROM mitarbeiter;
```

Listing 1.5 Einen mehrzeiligen Kommentar im SQL-Code anwenden

Die Datenbank ignoriert bei der Ausführung der SQL-Anweisung auch den mehrzeiligen Kommentar. Wenn Sie den gesamten Inhalt einer *.sql*-Datei auskommentieren, passiert einfach gar nichts. Bei der Ausführung der leeren Anweisung wird Ihnen kein Ergebnis zurückgeliefert, da keine SQL-Anweisung vorhanden ist.



Zusammenfassung: SQL-Anweisungen kommentieren

Kommentare werden zu Dokumentationszwecken verwendet. In SQL haben Sie zwei Möglichkeiten, Kommentare zu Ihren SQL-Anweisungen zu verfassen:

- ▶ Die erste Möglichkeit besteht darin, einzeilige Kommentare zu verfassen, die mit einem doppelten Bindestrich (--) eingeleitet werden.
- ▶ Die zweite Möglichkeit ist, mehrzeilige Kommentare zu verfassen. Diese beginnen stets mit einem Slash, gefolgt von einem Sternchen (/*), und enden mit einem Sternchen, gefolgt von einem Slash (*/).

1.9.1 Übungen

Übung 1

Geben Sie die SQL-Anweisung wie in Listing 1.6 gezeigt in Ihre MySQL Workbench ein. Vor der **SELECT**-Anweisung soll folgender einzeiliger Kommentar notiert werden:

Alle Spalten der Tabelle mitarbeiter abfragen

```
SELECT * FROM mitarbeiter;
```

Listing 1.6 Alle Spalten einer Tabelle abfragen

Führen Sie die Abfrage in Ihrem SQL-Client aus, um die Auswirkungen des einzeiligen Kommentars auf das Ergebnis der Abfrage zu ermitteln.

Übung 2

In dieser Übung sollen sämtliche Mitarbeiter ermittelt werden, die zur Abteilung *einkauf* gehören. Bitte schreiben Sie die SQL-Anweisung wie in Listing 1.7 angegeben in Ihr Editorfenster:

```
SELECT
    name,vorname,abteilung
FROM
```

```
    mitarbeiter
WHERE
    abteilung='einkauf';
```

Listing 1.7 Eine weitere SQL-Abfrage, die mit einem mehrzeiligen Kommentar versehen werden muss

Vor der **SELECT**-Anweisung notieren Sie einen Kommentar, der so strukturiert ist:

Programmierer: Ihr Name

Datum: Das aktuelle Datum

Funktion: Filtern aller Mitarbeiter, die zur Abteilung Einkauf gehören

Beachten Sie auch die Zeilenumbrüche.

Führen Sie die Abfrage aus, und ermitteln Sie, welche Auswirkungen der mehrzeilige Kommentar auf das Ergebnis der Abfrage hat.

Lösung zu Übung 1

```
-- Alle Spalten der Tabelle mitarbeiter abfragen
SELECT * FROM mitarbeiter;
```

Listing 1.8 Verwendung eines einzeiligen Kommentars in SQL

Der einzeilige Kommentar hat keine Auswirkungen auf das Ergebnis der Abfrage. Die Datenbank liefert Ihnen wie gewohnt eine Ergebnistabelle gemäß Ihrer **SELECT**-Abfrage zurück.

Lösung zu Übung 2

```
/*
Programmierer: Ihr Name
Datum: Das aktuelle Datum
Funktion: Filtern aller Mitarbeiter, die zur Abteilung Einkauf gehören
*/
SELECT
    name,vorname,abteilung
FROM
    mitarbeiter
WHERE
    abteilung='einkauf';
```

Listing 1.9 Verwendung eines mehrzeiligen Kommentars in SQL

Der mehrzeilige Kommentar hat bei der Ausführung der **SELECT**-Anweisung keinerlei Auswirkungen auf das Ergebnis der Abfrage.

Kapitel 10

Operationen auf Tabellen in Beziehungen anwenden

Ein relationales Datenbanksystem stellt sicher, dass die Beziehungen zwischen den Tabellen auch stets berücksichtigt werden. Wie wir uns dies zunutze machen, sehen Sie in diesem Kapitel.

In diesem Kapitel führen wir das Beispiel aus Kapitel 9, »Datenmodelle in Tabellen überführen«, fort und beschäftigen uns weiter mit der Ausbildungsdatenbank.

10.1 Zeilen in Tabellen einfügen, die in Beziehung zueinander stehen

Um Abhängigkeiten zwischen Tabellen zu beschreiben, verwenden wir auch hier die vereinfachte Ausdrucksweise der Eltern-Kind-Beziehung. Tabellen, die sich über eine *Fremdschlüsseldefinition* in der **CREATE TABLE**-Anweisung in Abhängigkeit zu einer anderen Tabelle befinden, bezeichnen wir als *Kindtabellen*. Tabellen, die sich nicht in einer solchen Abhängigkeit befinden, bezeichnen wir als *Elterntabellen*. In diesem Fall wären also die Tabellen *auszubildender* und *ausbildungsberuf* Beispiele für Elterntabellen, während *adresse* eine Kindtabelle darstellt, die in Beziehung zu einer anderen Tabelle steht.

10.1.1 Zeilen in die Tabelle »auszubildender« einfügen

Als Erstes fügen wir Zeilen in die Tabelle *auszubildender* ein. Die Tabelle *auszubildender* hat keinerlei Abhängigkeiten (*Foreign Key Constraints*), die wir beachten müssen, wenn wir neue Zeilen einfügen. Es ist lediglich darauf zu achten, dass für jeden Auszubildenden ein eindeutiger Primärschlüsselwert in der **INSERT**-Anweisung notiert wird. Listing 10.1 enthält sechs **INSERT**-Anweisungen, mit denen wir sechs neue Auszubildende in die Tabelle *auszubildender* eintragen:

```
INSERT INTO auszubildender (ausid,name,vorname,geburtsdatum)
VALUES (1,'Müller','Ralf','2001-04-01');
INSERT INTO auszubildender (ausid,name,vorname,geburtsdatum)
VALUES (2,'Klein','Sabine','2002-05-10');
```

```
INSERT INTO auszubildender (ausid,name,vorname,geburtsdatum)
VALUES (3,'Lang','Peter','2001-03-11');
INSERT INTO auszubildender (ausid,name,vorname,geburtsdatum)
VALUES (4,'Berg','Frank','2002-07-20');
INSERT INTO auszubildender (ausid,name,vorname,geburtsdatum)
VALUES (5,'Erde','Sabine','2001-01-23');
INSERT INTO auszubildender (ausid,name,vorname,geburtsdatum)
VALUES (6,'Grün','Justus','2001-04-15');
```

Listing 10.1 Zeilen in die Tabelle »auszubildender« einfügen

10.1.2 Zeilen in die Tabelle »ausbildungsberuf« einfügen

Als Nächstes fügen wir Zeilen in die Tabelle *ausbildungsberuf* ein. Die Tabelle *ausbildungsberuf* verfügt ebenfalls über keinerlei Abhängigkeiten (*Foreign Constraints*), die wir beachten müssen. Auch hier ist lediglich darauf zu achten, dass für jeden Ausbildungsberuf ein eindeutiger Primärschlüsselwert in der **INSERT**-Anweisung eingetragen wird. Listing 10.2 enthält sechs **INSERT**-Anweisungen, mit denen wir sechs neue Ausbildungsberufe in die Tabelle *ausbildungsberuf* eintragen:

```
INSERT INTO ausbildungsberuf (berufsid,berufsbezeichnung)
VALUES (1,'Energieelektroniker');
INSERT INTO ausbildungsberuf (berufsid,berufsbezeichnung)
VALUES (2,'Mechatroniker');
INSERT INTO ausbildungsberuf (berufsid,berufsbezeichnung)
VALUES (3,'Buchhalter');
INSERT INTO ausbildungsberuf (berufsid,berufsbezeichnung)
VALUES (4,'Industriekaufmann');
INSERT INTO ausbildungsberuf (berufsid,berufsbezeichnung)
VALUES (5,'Schlosser');
INSERT INTO ausbildungsberuf (berufsid,berufsbezeichnung)
VALUES (6,'Elektriker');
```

Listing 10.2 Zeilen in die Tabelle »ausbildungsberuf« einfügen

10.1.3 Zeilen in die Tabelle »lehrfach« einfügen

Bei der Tabelle *lehrfach* handelt es sich ebenfalls um eine nicht abhängige Tabelle. Wir können also neue Zeilen in die Tabelle *lehrfach* einfügen, ohne auf irgendwelche Abhängigkeiten (*Foreign Key Constraints*) achten zu müssen. Es ist nur wichtig, dass eindeutige Primärschlüsselwerte eingetragen werden. Listing 10.3 enthält zehn **INSERT**-Anweisungen, mit denen wir zehn neue Lehrfächer in die Tabelle *lehrfach* eintragen:

```
INSERT INTO lehrfach (lehrfachid,lehrfach)
VALUES (1,'Mathematik');
```

```
INSERT INTO lehrfach (lehrfachid, lehrfach)
VALUES (2, 'Buchhaltung 1');
INSERT INTO lehrfach (lehrfachid, lehrfach)
VALUES (3, 'Buchhaltung 2');
INSERT INTO lehrfach (lehrfachid, lehrfach)
VALUES (4, 'Mechanik Grundlagen 1');
INSERT INTO lehrfach (lehrfachid, lehrfach)
VALUES (5, 'Mechanik Grundlagen 2');
INSERT INTO lehrfach (lehrfachid, lehrfach)
VALUES (6, 'Englisch');
INSERT INTO lehrfach (lehrfachid, lehrfach)
VALUES (7, 'Elektronik Grundlagen 1');
INSERT INTO lehrfach (lehrfachid, lehrfach)
VALUES (8, 'Elektronik Grundlagen 2');
INSERT INTO lehrfach (lehrfachid, lehrfach)
VALUES (9, 'Rechnungsbearbeitung 1');
INSERT INTO lehrfach (lehrfachid, lehrfach)
VALUES (10, 'Rechnungsbearbeitung 2');
```

Listing 10.3 Zeilen in die Tabelle »lehrfach« einfügen

10.1.4 Zeilen in die Tabelle »adresse« (inklusive der Beziehungen) einfügen

Die Tabelle *adresse* befindet sich in einer 1:n-Abhängigkeit zu der Tabelle *auszubildender*. Neue Zeilen, die wir hier mit einer **INSERT**-Anweisung eintragen, müssen also auf gültige Primärschlüsselwerte in der Tabelle *auszubildender* verweisen. Die sechs **INSERT**-Anweisungen in Listing 10.4 sorgen dafür, dass sechs Adressen in die Tabelle *adresse* eingetragen werden. Die ersten drei **INSERT**-Anweisungen werden mit den Fremdschlüsselwerten 1, 3 und 5 ausgestattet und verweisen somit in der Tabelle *auszubildender* auf die Zeilen mit den Primärschlüsselwerten 1, 3 und 5. Die letzten drei **INSERT**-Anweisungen enthalten keine Fremdschlüsselwerte. Hier wurde anstelle eines Fremdschlüsselwerts der nicht definierte Wert **NULL** eingetragen. Die letzten drei Zeilen verweisen also auf keine Auszubildenden. Es handelt sich stattdessen um Adressen, die nicht zugeordnet sind.

```
/* Mit Zuordnung zu Auszubildenden */
INSERT INTO adresse (aid, strasse, nr, plz, ort, fk_ausid)
VALUES (1, 'Mondstraße', '8', 50827, 'Köln', 1);
INSERT INTO adresse (aid, strasse, nr, plz, ort, fk_ausid)
VALUES (2, 'Sternstraße', '10', 50127, 'Bonn', 3);
INSERT INTO adresse (aid, strasse, nr, plz, ort, fk_ausid)
VALUES (3, 'Sonnenstraße', '1', 50129, 'Bonn', 5);
```

```
/*Ohne Zuordnung zu Auszubildenden*/  
INSERT INTO adresse (aid,strasse,nr,plz,ort,fk_ausid)  
VALUES (4,'Jupiterstraße','11',50827,'Köln',NULL);  
INSERT INTO adresse (aid,strasse,nr,plz,ort,fk_ausid)  
VALUES (5,'Uranusstraße','9',50127,'Bonn',NULL);  
INSERT INTO adresse (aid,strasse,nr,plz,ort,fk_ausid)  
VALUES (6,'Marsstraße','9',50129,'Bonn',NULL);
```

Listing 10.4 Neue Zeilen in die Tabelle »adresse« einfügen

10.1.5 Zeilen in die Tabelle »ausbildungsvertrag« (inklusive der Beziehungen) einfügen

Die Tabelle *ausbildungsvertrag* steht jeweils in einer 1:1-Beziehung zu den Tabellen *auszubildender* und *ausbildungsberuf*. Neue Zeilen, die wir mit einer **INSERT**-Anweisung eintragen, müssen also auf gültige Primärschlüsselwerte der Tabellen *auszubildender* und *ausbildungsberuf* verweisen. Listing 10.5 enthält sechs **INSERT**-Anweisungen. Die letzten beiden Werte, die in der **VALUES**-Klausel enthalten sind, verweisen mit ihren Fremdschlüsselwerten der Spalten *fk_ausid* und *fk_berufsid* auf die Primärschlüsselwerte der Tabellen *auszubildender* (Primärschlüsselspalte: *ausid*) und *ausbildungsberuf* (Primärschlüsselspalte: *berufsid*). Jede Zeile, die wir hier einfügen, enthält also eine Beziehung zu den Zeilen in den Tabellen *auszubildender* und *ausbildungsberuf*.

```
INSERT INTO ausbildungsvertrag (vid,vertragsdatum,fk_ausid,fk_berufsid)  
VALUES (1,'2015-06-01',5,2);  
INSERT INTO ausbildungsvertrag (vid,vertragsdatum,fk_ausid,fk_berufsid)  
VALUES (2,'2015-06-01',4,4);  
INSERT INTO ausbildungsvertrag (vid,vertragsdatum,fk_ausid,fk_berufsid)  
VALUES (3,'2015-06-01',1,3);  
INSERT INTO ausbildungsvertrag (vid,vertragsdatum,fk_ausid,fk_berufsid)  
VALUES (4,'2015-06-01',3,1);  
INSERT INTO ausbildungsvertrag (vid,vertragsdatum,fk_ausid,fk_berufsid)  
VALUES (5,'2015-06-01',6,1);  
INSERT INTO ausbildungsvertrag (vid,vertragsdatum,fk_ausid,fk_berufsid)  
VALUES (6,'2015-06-01',2,5);
```

Listing 10.5 Zeilen in die Tabelle »ausbildungsvertrag« einfügen

10.1.6 Zeilen in die Tabelle »beruflehrfach« (inklusive der Beziehungen) einfügen

Die Tabelle *beruflehrfach* bildet eine m:n-Beziehung zwischen den Tabellen *ausbildungsberuf* und *lehrfach* ab. In der Tabelle *beruflehrfach* werden Kombinationen von

Fremdschlüsselwerten gespeichert, die jeweils auf die Primärschlüsselwerte der Tabellen *ausbildungsberuf* und *lehrfach* verweisen.

Die Fremdschlüsselspalte *fk_berufsid* der Tabelle *beruflehrfach* verweist auf die Primärschlüsselspalte *berufsid* der Tabelle *ausbildungsberuf*. Die Fremdschlüsselspalte *fk_lehrfachid* wiederum verweist auf die Primärschlüsselspalte *lehrfachid* der Tabelle *lehrfach*. In Listing 10.6 sehen Sie zwölf **INSERT**-Anweisungen, mit denen Kombinationen von Fremdschlüsselwerten gespeichert werden, um entsprechende Beziehungen zu den Zeilen der Tabellen *ausbildungsberuf* und *lehrfach* herzustellen:

```
INSERT INTO beruflehrfach (fk_berufsid,fk_lehrfachid)
VALUES (1,1);
INSERT INTO beruflehrfach (fk_berufsid,fk_lehrfachid)
VALUES (1,6);
INSERT INTO beruflehrfach (fk_berufsid,fk_lehrfachid)
VALUES (1,7);
INSERT INTO beruflehrfach (fk_berufsid,fk_lehrfachid)
VALUES (1,8);
INSERT INTO beruflehrfach (fk_berufsid,fk_lehrfachid)
VALUES (4,2);
INSERT INTO beruflehrfach (fk_berufsid,fk_lehrfachid)
VALUES (4,3);
INSERT INTO beruflehrfach (fk_berufsid,fk_lehrfachid)
VALUES (4,6);
INSERT INTO beruflehrfach (fk_berufsid,fk_lehrfachid)
VALUES (4,9);
INSERT INTO beruflehrfach (fk_berufsid,fk_lehrfachid)
VALUES (6,1);
INSERT INTO beruflehrfach (fk_berufsid,fk_lehrfachid)
VALUES (6,4);
INSERT INTO beruflehrfach (fk_berufsid,fk_lehrfachid)
VALUES (6,7);
INSERT INTO beruflehrfach (fk_berufsid,fk_lehrfachid)
VALUES (6,8);
```

Listing 10.6 Zeilen in die Tabelle »beruflehrfach« einfügen

10.1.7 Zeilen in die Tabelle »mitarbeiterausbildungsbetrieb« (inklusive der Beziehungen) einfügen

Die Tabelle *mitarbeiterausbildungsbetrieb* bildet eine 1:n-Beziehung mit sich selbst. Die Tabelle enthält eine Fremdschlüsselspalte *fk_mitarbeiterid*, die auf die Primärschlüsselspalte *mitarbeiterid* referenziert.

Listing 10.7 enthält zehn **INSERT**-Anweisungen, in denen jeweils Werte der Spalte *fk_mitarbeiterid* auf Werte der Spalte *mitarbeiterid* verweisen:

```
INSERT INTO mitarbeiterausbildungsbetrieb (mitarbeiterid,name,vorname,
fk_mitarbeiterid)
VALUES (1,'Müller','Alfred',NULL);
INSERT INTO mitarbeiterausbildungsbetrieb (mitarbeiterid,name,vorname,
fk_mitarbeiterid)
VALUES (2,'Ungern','Peter',1);
INSERT INTO mitarbeiterausbildungsbetrieb (mitarbeiterid,name,vorname,
fk_mitarbeiterid)
VALUES (3,'Erdenschein','Claudia',1);
INSERT INTO mitarbeiterausbildungsbetrieb (mitarbeiterid,name,vorname,
fk_mitarbeiterid)
VALUES (4,'Sternenschein','Ute',1);
INSERT INTO mitarbeiterausbildungsbetrieb (mitarbeiterid,name,vorname,
fk_mitarbeiterid)
VALUES (5,'Augustus','Frank',1);
INSERT INTO mitarbeiterausbildungsbetrieb (mitarbeiterid,name,vorname,
fk_mitarbeiterid)
VALUES (6,'Erdenfels','Christine',NULL);
INSERT INTO mitarbeiterausbildungsbetrieb (mitarbeiterid,name,vorname,
fk_mitarbeiterid)
VALUES (7,'Hoffnung','Ralf',6);
INSERT INTO mitarbeiterausbildungsbetrieb (mitarbeiterid,name,vorname,
fk_mitarbeiterid)
VALUES (8,'Freud','Erika',6);
INSERT INTO mitarbeiterausbildungsbetrieb (mitarbeiterid,name,vorname,
fk_mitarbeiterid)
VALUES (9,'Bergfels','Diether',6);
INSERT INTO mitarbeiterausbildungsbetrieb (mitarbeiterid,name,vorname,
fk_mitarbeiterid)
VALUES (10,'Lemon','Reinhold',6);
```

Listing 10.7 Zeilen in die Tabelle »mitarbeiterausbildungsbetrieb« einfügen



Zusammenfassung: Zeilen, die in Beziehung stehen, in Tabellen einfügen

Wenn Zeilen in *Kindtabellen*, die in Beziehung zu Zeilen aus *Elterntabellen* stehen, eingefügt werden sollen, gilt es, eine Reihenfolge einzuhalten.

Zuerst werden die Zeilen in eine Elterntabelle eingefügt. Dann werden die Zeilen mit den entsprechenden *Fremdschlüsselwerten* in Kindtabellen eingefügt.

10.1.8 Übungen

Übung 1

In Kapitel 9, »Datenmodelle in Tabellen überführen«, haben Sie Tabellen erstellt, die Informationen für eine Bibliothek verwalten sollen. In dieser Übung sollen für die Tabellen *fachbereich* und *verlag*, die starke Entitäten darstellen, **INSERT**-Anweisungen formuliert werden, mit denen Sie Zeilen in die Tabellen einfügen. Die Tabellen sind also autark. Sie müssen hier keine Beziehungen zu Zeilen in anderen Tabellen beachten. In Tabelle 10.1 sehen Sie Zeilen, die für die Tabelle *verlag* vorgesehen sind. Tabelle 10.2 enthält wiederum Zeilen, die für die Tabelle *fachbereich* bestimmt sind.

verlagid	verlag
1	Rheinwerk Verlag
2	Elektro Verlag
3	Mechanik Verlag
4	Kaufmann Verlag
5	Medien Verlag

Tabelle 10.1 Zeilen für die Tabelle »verlag«

fachbereichid	fachbereich
1	Elektrotechnik
2	Kaufmann
3	Mechanik
4	Pneumatik

Tabelle 10.2 Zeilen für die Tabelle »fachbereich«

Übung 2

Fügen Sie Zeilen in die Tabelle *fachbuch* ein, die mit einem Fremdschlüsselwert auf Zeilen in der Tabelle *verlag* referenzieren. In Übung 1 haben Sie fünf Verlage in die Tabelle *verlag* eingefügt. Uns stehen also fünf Primärschlüsselwerte aus der Tabelle *verlag* zur Verfügung, die wir als Fremdschlüsselwerte in der Tabelle *fachbuch* verwenden können, je nachdem, von welchem Verlag ein Fachbuch verlegt wird. Beachten Sie, dass die Fremdschlüsselwerte in die Spalte *fk_verlagid* eingefügt werden. Diese

Fremdschlüsselwerte referenzieren auf Zeilen, die Sie in Tabelle 10.3 für **INSERT**-Anweisungen für die Tabelle *verlag* genutzt haben. Die letzten drei Zeilen mit den Primärschlüsselwerten 7, 8 und 9 der Spalte *fachbuchid* verfügen über keine Referenz, die über Fremdschlüsselwerte festgelegt wurde. Tragen Sie hier jeweils **NULL** in die **VALUES**-Klausel der **INSERT**-Anweisung ein.

fachbuchid	isbn	titel	fk_verlagid
1	1235	Mechanik	3
2	9878	Elektrotechnik	2
3	2323	Elektronik	2
4	2254	Pneumatik	3
5	4455	Mathematik Grundlagen 1	2
6	4456	Mathematik Grundlagen 2	2
7	5566	Mengenlehre	NULL
8	7766	Kommunikation 1	NULL
9	7767	Kommunikation 2	NULL

Tabelle 10.3 Zeilen für die Tabelle »fachbuch«

Übung 3

Jetzt fehlt uns noch die Zuordnung eines Fachbuchs zu einem Fachbereich. Diese Zuordnung können Sie mit der Schlüsseltabelle *fachbereichfachbuch* realisieren. Hier müssen jeweils gültige Primärschlüsselwerte oder Kombinationen aus den Tabellen *fachbuch* und *fachbereich* eingetragen werden, um die Beziehung eines Schlüsselpaars zu den Zeilen der Tabellen *fachbuch* bzw. *fachbereich* herzustellen.

In Tabelle 10.4 sehen Sie die Zuordnungen über die Fremdschlüsselwerte eines Fachbuchs zu einem Fachbereich. Formulieren Sie **INSERT**-Anweisungen, mit denen Sie die jeweiligen Zuordnungen in die Tabelle *fachbereichfachbuch* eintragen.

fk_fachbereichid	fk_fachbuchid
3	1
1	3
4	4

Tabelle 10.4 Zeilen für die Tabelle »fachbereichfachbuch«

fk_fachbereichid	fk_fachbuchid
1	5
1	6
1	7
2	8
2	9

Tabelle 10.4 Zeilen für die Tabelle »fachbereichfachbuch« (Forts.)

Lösung zu Übung 1

```

/* Zeilen in die Tabelle verlag einfügen */
INSERT INTO verlag (verlagid,verlag)
VALUES (1, 'Rheinwerk Verlag');
INSERT INTO verlag (verlagid,verlag)
VALUES (2, 'Elektro Verlag');
INSERT INTO verlag (verlagid,verlag)
VALUES (3, 'Mechanik Verlag');
INSERT INTO verlag (verlagid,verlag)
VALUES (4, 'Kaufmann Verlag');
INSERT INTO verlag (verlagid,verlag)
VALUES (5, 'Medien Verlag');

/* Zeilen in die Tabelle fachbereich einfügen */
INSERT INTO fachbereich (fachbereichid,fachbereich)
VALUES (1, 'Elektrotechnik');
INSERT INTO fachbereich (fachbereichid,fachbereich)
VALUES (2, 'Kaufmann');
INSERT INTO fachbereich (fachbereichid,fachbereich)
VALUES (3, 'Mechanik');
INSERT INTO fachbereich (fachbereichid,fachbereich)
VALUES (4, 'Pneumatik');

```

Listing 10.8 INSERT-Anweisungen zum Einfügen von Zeilen in die Tabellen »verlag« und »fachbereich«

Lösung zu Übung 2

```

INSERT INTO fachbuch (fachbuchid,isbn,titel,fk_verlagid)
VALUES (1,'1235','Mechanik',3);

```

```
INSERT INTO fachbuch (fachbuchid,isbn,titel,fk_verlagid)
VALUES (2,'9878','Elektrotechnik',2);
INSERT INTO fachbuch (fachbuchid,isbn,titel,fk_verlagid)
VALUES (3,'2323','Elektronik',2);
INSERT INTO fachbuch (fachbuchid,isbn,titel,fk_verlagid)
VALUES (4,'2254','Pneumatik',3);
INSERT INTO fachbuch (fachbuchid,isbn,titel,fk_verlagid)
VALUES (5,'4455','Mathematik Grundlagen 1',2);
INSERT INTO fachbuch (fachbuchid,isbn,titel,fk_verlagid)
VALUES (6,'4456','Mathematik Grundlagen 2',2);
INSERT INTO fachbuch (fachbuchid,isbn,titel,fk_verlagid)
VALUES (7,'5566','Mengenlehre',NULL);
INSERT INTO fachbuch (fachbuchid,isbn,titel,fk_verlagid)
VALUES (8,'7766','Kommunikation 1',NULL);
INSERT INTO fachbuch (fachbuchid,isbn,titel,fk_verlagid)
VALUES (9,'7767','Kommunikation 2',NULL);
```

Listing 10.9 INSERT-Anweisungen zum Einfügen von Zeilen in die Tabelle »fachbuch«

Lösung zu Übung 3

```
INSERT INTO fachbereichfachbuch (fk_fachbereichid,fk_fachbuchid)
VALUES (3,1);
INSERT INTO fachbereichfachbuch (fk_fachbereichid,fk_fachbuchid)
VALUES (1,3);
INSERT INTO fachbereichfachbuch (fk_fachbereichid,fk_fachbuchid)
VALUES (4,4);
INSERT INTO fachbereichfachbuch (fk_fachbereichid,fk_fachbuchid)
VALUES (1,5);
INSERT INTO fachbereichfachbuch (fk_fachbereichid,fk_fachbuchid)
VALUES (1,6);
INSERT INTO fachbereichfachbuch (fk_fachbereichid,fk_fachbuchid)
VALUES (1,7);
INSERT INTO fachbereichfachbuch (fk_fachbereichid,fk_fachbuchid)
VALUES (2,8);
INSERT INTO fachbereichfachbuch (fk_fachbereichid,fk_fachbuchid)
VALUES (2,9);
```

Listing 10.10 INSERT-Anweisungen zum Einfügen von Zeilen in die Tabelle »fachbereichfachbuch«

10.2 Zeilen aus Tabellen, die in Beziehung stehen, mit JOIN verbunden abfragen

Wir haben Tabellen in Beziehung zueinander gesetzt und Datensätze in Tabellen eingefügt, die berücksichtigen, dass sie in Beziehung zueinander stehen. Jetzt schauen wir uns an, wie Sie Datensätze aus solchen Tabellen abfragen. Daher nutzen wir die Beziehungen zwischen den Tabellen, um **SELECT**-Abfragen zu formulieren, mit denen wir die Datensätze, deren Schlüssel gleich sind, verbunden abfragen. Das heißt, dass Sie in der Spaltenauswahlliste die Wahl haben zwischen den Spalten, die in den verbundenen Tabellen vorkommen.

Es gibt vier unterschiedliche **JOIN**-Arten, mit denen Sie Tabellen abfragen können, deren Zeilen durch Fremdschlüsselbeziehungen verbunden sind: der **INNER JOIN**, der **LEFT OUTER JOIN**, der **RIGHT OUTER JOIN** und der **FULL OUTER JOIN**. Wenn Sie einen **INNER JOIN** verwenden, werden die Zeilen aus Tabellen verbunden abgefragt, wenn Schlüsselwerte übereinstimmen. Ein **LEFT OUTER JOIN** verbindet hingegen alle Zeilen der linken Tabelle mit allen Zeilen der rechten Tabelle. Außerdem bleiben Ihnen beim **LEFT OUTER JOIN** alle Zeilen erhalten, die in der linken Tabelle existent sind, aber in der rechten Tabelle nicht mit Schlüsselwerten verbunden werden können. Für den **RIGHT OUTER JOIN** gilt das Ganze umgekehrt. Ein **FULL OUTER JOIN** verbindet Zeilen mit übereinstimmenden Schlüsselwerten. In die Ergebnistabelle werden auch Zeilen der linken und rechten Tabelle aufgenommen, die keine übereinstimmenden Schlüsselwerte haben. Ein **CROSS JOIN** schließlich verbindet alle Zeilen der beiden Tabellen unabhängig von allen Schlüsselbeziehungen miteinander.

Die Diagramme in Abbildung 10.1 bis Abbildung 10.5 veranschaulichen die Operationen:

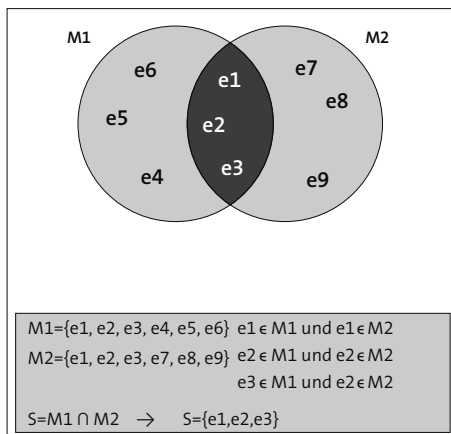


Abbildung 10.1 Ein INNER JOIN

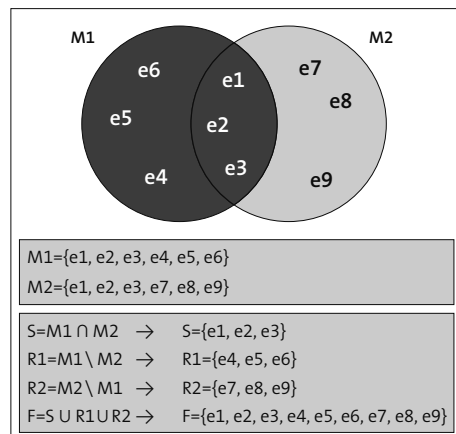


Abbildung 10.2 Ein LEFT OUTER JOIN

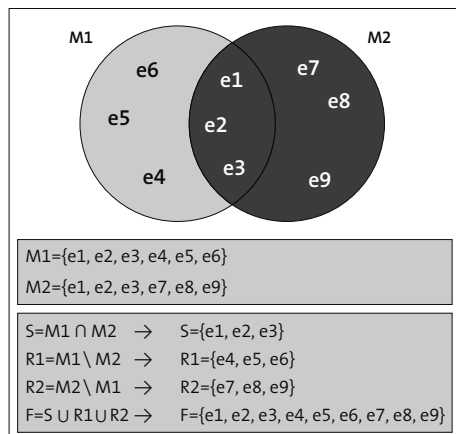


Abbildung 10.3 Ein RIGHT OUTER JOIN

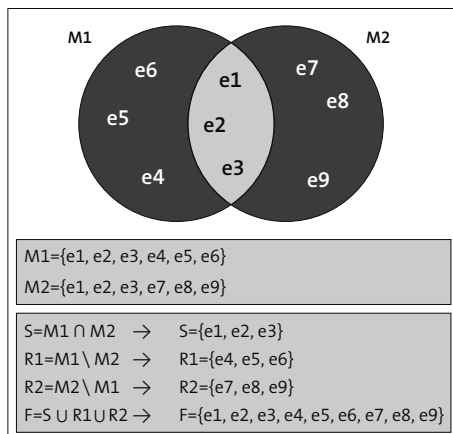


Abbildung 10.4 Ein FULL OUTER JOIN

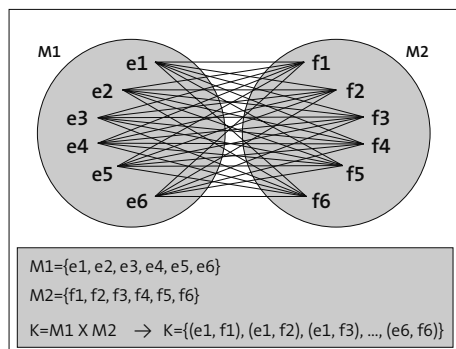


Abbildung 10.5 Ein CROSS JOIN

10.2.1 Zeilen mit einem INNER JOIN verbinden

Ein **INNER JOIN** verbindet Zeilen von Tabellen, indem auf Gleichheit von Primärschlüssel- und Fremdschlüsselwerten geprüft wird. Wie bereits erläutert, stehen die Tabellen *auszubildender* und *adresse* über eine Fremdschlüsselbeziehung in der Tabelle *adresse* zueinander in Beziehung.

In Abbildung 10.6 sehen Sie, dass jeweils die Fremdschlüsselwerte der Spalte *fk_ausid* der Tabelle *adresse* auf gleiche Primärschlüsselwerte der Spalte *ausid* in der Tabelle *auszubildender* verweisen.

Diese Tabellen, die in Beziehung zueinander stehen, wollen wir mit einer **SELECT**-Anweisung abfragen. Wenn wir zwei Tabellen abfragen, stehen uns natürlich auch die Spalten aus zwei Tabellen zur Verfügung. Sie realisieren die Verbindung von Zeilen über identische Schlüsselwerte aus Tabellen mit einem **INNER JOIN**. Ein **INNER JOIN** ver-

bindet Zeilen von Tabellen nur dann, wenn eine Schlüsselbeziehung zwischen einer Eltern- und einer Kindtabelle existiert. Wenn Zeilen in den Tabellen vorhanden sind, die über keine Beziehung verfügen, werden sie also nicht von einem **INNER JOIN** berücksichtigt.

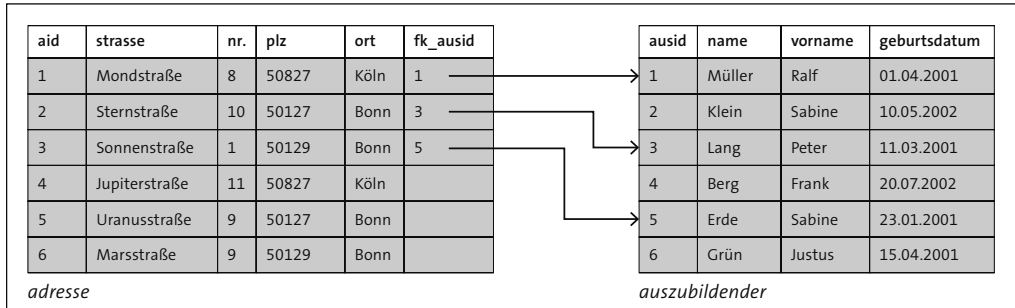


Abbildung 10.6 Zeilen von Tabellen mit Schlüsselwerten verbinden

In Listing 10.11 sehen Sie eine **SELECT**-Abfrage mit einem **INNER JOIN**:

```
SELECT name,vorname,plz,ort
FROM auszubildender INNER JOIN adresse
ON ausid=fk_ausid;
```

Listing 10.11 Die Tabellen »auszubildender« und »adresse« mit einem **INNER JOIN** verbunden abfragen

Hinter dem Schlüsselwort **SELECT** sehen Sie wie gewohnt eine Spaltenauswahlliste. Hier gibt es nur eine Neuerung: Die Spalten *name* und *vorname* fragen wir aus der Tabelle *auszubildender* ab. Die Spalten *plz* und *ort* stammen hingegen aus der Tabelle *adresse*. Es folgen das Schlüsselwort **FROM** und die Tabellenbezeichnung *auszubildender*. Bis hierhin haben Sie fast nichts Neues kennengelernt.

Hinter der Tabellenbezeichnung *auszubildender* folgen jetzt die neuen Schlüsselwörter **INNER JOIN** (innerer Verbund). Hinter der **INNER JOIN**-Klausel notieren Sie die Tabelle, die über einen **INNER JOIN** verbunden abgefragt werden soll. Somit haben Sie schon einmal bekannt gemacht, welche Tabellen verbunden abgefragt werden sollen.

Jetzt müssen wir nur noch angeben, wie die Zeilen der Tabellen miteinander verbunden werden sollen. Die Bedingung, die auf Gleichheit von Schlüsselwerten prüft (Primärschlüssel- und Fremdschlüsselwerte), legen Sie in einer **ON**-Klausel fest.

Das Schlüsselwort **INNER** der **INNER JOIN**-Klausel ist optional anzugeben. In Listing 10.12 sehen Sie eine **SELECT**-Abfrage mit einem **INNER JOIN**, in dem auf das Schlüsselwort **INNER** verzichtet wurde. Diese Abfrage liefert Ihnen exakt das gleiche Ergebnis wie die Abfrage aus Listing 10.11.

```
SELECT name,vorname,plz,ort
FROM auszubildender JOIN adresse
ON ausid=fk_ausid;
```

Listing 10.12 Einen INNER JOIN ohne das optionale Schlüsselwort INNER verwenden

Tabelle 10.5 zeigt Ihnen das Ergebnis der Abfrage:

name	vorname	plz	ort
Müller	Ralf	50827	Köln
Lang	Peter	50127	Bonn
Erde	Sabine	50129	Bonn

Tabelle 10.5 Ergebnistabelle für eine Abfrage mit einem INNER JOIN

Die Zeilen, die jeweils durch eine Referenz der Kindtabelle *adresse* auf Zeilen der Elterntabelle *auszubildender* verweisen, konnten erfolgreich verbunden werden. Sicher fällt Ihnen auf, dass die Datensätze, die über keine Referenz (Fremdschlüsselwert gleich NULL) aus der Tabelle *adresse* verfügen, nicht verbunden werden konnten. Wie Sie Zeilen, die über keine Schlüsselbeziehungen verfügen, mit in die Ergebnisliste aufnehmen, werden Sie in den nächsten Abschnitten erfahren.

10.2.2 Zeilen mit einem LEFT OUTER JOIN verbinden

Ein **LEFT OUTER JOIN** verknüpft ebenfalls Zeilen von Tabellen über eine Fremdschlüsselbeziehung. Zunächst einmal verbindet ein **LEFT OUTER JOIN** alle Zeilen von zwei Tabellen über eine Schlüsselbeziehung. Immer dann, wenn ein Fremdschlüssel aus einer Kindtabelle auf einen Primärschlüssel aus einer Elterntabelle referenziert, werden Zeilen entsprechend verbunden. Das kennen Sie bereits vom **INNER JOIN**.

Ein **LEFT OUTER JOIN** gibt zusätzlich die Zeilen der Tabelle *auszubildender* (die links von der **LEFT OUTER JOIN**-Klausel notiert ist) aus, die nicht über Schlüsselwerte mit der Tabelle *adresse* (die rechts von der **LEFT OUTER JOIN**-Klausel notiert ist) verbunden werden können. Diese Zeilen bleiben uns also erhalten. In diesem Fall bedeutet das, dass die Spaltenwerte der Zeilen der Elterntabelle *auszubildender* ausgegeben werden, auf die keine Referenzierung aus den Zeilen der Kindtabelle *adresse* erfolgt.



LEFT OUTER JOIN und Schlüsselbeziehungen

Beachten Sie Folgendes: Einem **LEFT OUTER JOIN** ist es grundsätzlich egal, ob Sie Fremdschlüsselwerte mit Primärschlüsselwerten verglichen. In der **ON**-Klausel wird

eine Bedingung ausgewertet. Wenn diese wahr ist, werden Zeilen verbunden ausgegeben. Wenn sie nicht wahr ist, wird keine Zeile verbunden abgefragt. Das war es auch schon.

Der **LEFT OUTER JOIN** fragt auch die Zeilen aus der Tabelle links von der **LEFT OUTER JOIN**-Klausel ab, die nicht über eine passende Schlüsselbeziehung mit der rechten Tabelle verbunden werden können. Die Zeilen der Tabelle links von der **LEFT OUTER JOIN**-Klausel sind daher ebenso in der Ergebnistabelle enthalten.

In Abbildung 10.6 sehen Sie das Ergebnis der Abfrage aus Listing 10.13. Darin finden Sie jetzt insgesamt sechs Zeilen. Aus der Tabelle *adresse* wird nicht auf die Zeilen mit den Primärschlüsselwerten 2, 4 und 6 in der Spalte *ausid* referenziert. Diesen drei Auszubildenden kann derzeit noch kein Wohnort zugeordnet werden. In der Tabelle *adresse* sehen Sie also auch keine Zeilen mit Fremdschlüsselwerten, die auf die Zeilen der Tabelle *auszubildender* mit Primärschlüsselwerten verweisen.

Die **SELECT**-Abfrage in Listing 10.13 verbindet Zeilen über einen **LEFT OUTER JOIN**:

```
SELECT name,vorname,plz,ort
FROM auszubildender LEFT OUTER JOIN adresse
ON ausid=fk_ausid;
```

Listing 10.13 Die Tabellen »auszubildender« und »adresse« mit einem **LEFT OUTER JOIN** verbunden abfragen

Die Zeilen aus der linken Elterntabelle *auszubildender*, die nicht mit der Kindtabelle *adresse* verbunden werden können, werden ebenso abgefragt. Anstelle des **INNER JOIN**-Schlüsselworts verwenden Sie jetzt das Schlüsselwort **LEFT OUTER JOIN**, um alle Zeilen der linken Tabelle abzufragen, die über Schlüsselwerte mit der rechten Tabelle verbunden oder eben nicht verbunden werden können.

Das Schlüsselwort **OUTER** der **LEFT OUTER JOIN**-Klausel ist dabei optional anzugeben. In Listing 10.14 sehen Sie eine **SELECT**-Abfrage mit einem **LEFT OUTER JOIN**, in dem auf das Schlüsselwort **OUTER** verzichtet wurde. Diese Abfrage liefert Ihnen exakt das gleiche Ergebnis wie die Abfrage aus Listing 10.13.

```
SELECT name,vorname,plz,ort
FROM auszubildender LEFT JOIN adresse
ON ausid=fk_ausid;
```

Listing 10.14 Einen **LEFT OUTER JOIN** ohne das optionale Schlüsselwort **OUTER** verwenden

Tabelle 10.6 zeigt Ihnen das Ergebnis der Abfrage aus Listing 10.14:

name	vorname	plz	ort
Müller	Ralf	50827	Köln
Klein	Sabine	NULL	NULL
Lang	Peter	50127	Bonn
Berg	Frank	NULL	NULL
Erde	Sabine	50129	Bonn
Grün	Justus	NULL	NULL

Tabelle 10.6 Ergebnistabelle für eine Abfrage mit einem LEFT OUTER JOIN

Sie sehen, dass die Zeilen der Tabelle *auszubildender*, denen über die Fremdschlüsselbeziehungen der Tabelle *adresse* kein Wohnort zugeordnet werden konnte, ebenfalls angezeigt werden. Für diese Zeilen existieren natürlich keine Angaben bzw. Spaltenwerte zum Wohnort.

Wenn ein **LEFT OUTER JOIN** verwendet wird und Zeilen in der linken Tabelle existieren, die nicht über eine Schlüsselbeziehung mit Zeilen der rechten Tabelle verbunden werden können, füllt die Datenbank die Spaltenwerte der rechten Tabelle mit **NULL**-Werten auf. Wie können wir nun die Auszubildenden ermitteln, denen keine Adresse zugeordnet ist? Sie haben eben erfahren, dass die Datenbank unter Verwendung eines **LEFT OUTER JOINs** alle Spaltenwerte von Zeilen der rechten Tabelle, die nicht mit Zeilen aus der linken Tabelle verbunden werden können, mit **NULL**-Werten auffüllt. Mit einer einfachen **WHERE**-Klausel, die auf **NULL**-Werte prüft, können Sie jetzt die Auszubildenden ermitteln, denen keine Adresse zugeordnet ist. Die **WHERE**-Klausel notieren Sie wie in Listing 10.15 angegeben hinter der **ON**-Klausel, um die Spalte *plz* der Tabelle *adresse* auf **NULL** zu prüfen. Welche Spalte der Tabelle *adresse* Sie auf **NULL**-Werte prüfen, ist nicht bedeutend, da ja sämtliche Spaltenwerte der Tabelle *wohntort* mit **NULL**-Werten in der Ergebnistabelle aufgefüllt werden, wenn keine Referenzierung zu einem Auszubildenden bzw. einer Zeile der Tabelle *auszubildender* möglich ist.

```
SELECT name,vorname,plz,ort
FROM auszubildender LEFT OUTER JOIN adresse
ON ausid=fk_ausid
WHERE plz IS NULL;
```

Listing 10.15 Die Tabellen »auszubildender« und »adresse« mit einem **LEFT OUTER JOIN** verbunden abfragen und nur die Zeilen filtern, denen keine Adresse zugeordnet werden konnte

Tabelle 10.7 zeigt Ihnen das Ergebnis der Abfrage aus Listing 10.15. Gemäß der Bedingung der **WHERE**-Klausel sind jetzt nur noch diejenigen Zeilen der Tabelle *auszubildender* in der Ergebnistabelle enthalten, die nicht über Schlüsselwerte mit einer Adresse verbunden werden konnten.

name	vorname	plz	ort
Klein	Sabine	NULL	NULL
Berg	Frank	NULL	NULL
Grün	Justus	NULL	NULL

Tabelle 10.7 Ergebnistabelle für eine Abfrage mit einem **LEFT OUTER JOIN** und einer Bedingung, die eine Spalte der rechten Tabelle auf **NULL**-Werte prüft

10.2.3 Zeilen mit einem **RIGHT OUTER JOIN** verbinden

Ein **RIGHT OUTER JOIN** verbindet zunächst einmal wie ein **INNER JOIN** oder ein **LEFT OUTER JOIN** alle Zeilen von zwei Tabellen, die über Schlüsselwerte verbunden abgefragt werden können.

Wie sieht es nun aus, wenn wir eine Kindtabelle abfragen wollen, die Zeilen ohne Referenzierung auf die Elterntabelle enthält? In Abschnitt 10.1.4 haben Sie bereits Zeilen in die Tabelle *adresse* eingefügt, die nicht auf Fremdschlüsselwerte, sondern auf Primärschlüsselwerte aus Zeilen der Tabelle *auszubildender* verweisen. Im Beispiel enthalten diese Zeilen **NULL**-Werte. Diese Zeilen der Tabelle *adresse* stehen also schlicht und ergreifend nicht in einer Beziehung zu Zeilen der Tabelle *auszubildender*.

Ein **RIGHT OUTER JOIN** gibt auch die Zeilen der Tabelle *adresse* (die rechts von der **RIGHT OUTER JOIN**-Klausel notiert ist) aus, die nicht über Schlüsselwerte mit der Tabelle *auszubildender* (die links von der **RIGHT OUTER JOIN**-Klausel notiert ist) verbunden werden können. Diese Zeilen bleiben uns also erhalten.

In unserem Fall bedeutet das, dass die Spaltenwerte derjenigen Zeilen der Tabelle *adresse* ausgegeben werden, die über keine Referenzierung zu den Zeilen der Tabelle *auszubildender* verfügen.

RIGHT OUTER JOIN und Schlüsselbeziehungen

Beachten Sie auch bei dieser Beziehungsart Folgendes: Einem **RIGHT OUTER JOIN** ist es ebenfalls grundsätzlich egal, ob Sie Fremdschlüsselwerte mit Primärschlüsselwerten vergleichen. In der **ON**-Klausel wird eine Bedingung ausgewertet; wenn sie wahr ist, werden Zeilen verbunden ausgegeben. Wenn sie nicht wahr ist, wird keine Zeile verbunden zurückgegeben.



Der **RIGHT OUTER JOIN** fragt also auch die Zeilen aus der Tabelle rechts von der **RIGHT OUTER JOIN**-Klausel ab, die nicht über eine passende Schlüsselbeziehung verbunden werden können. Die Zeilen der Tabelle rechts von der **RIGHT OUTER JOIN**-Klausel sind daher ebenso in der Ergebnistabelle enthalten.

Betrachten Sie auch hier Abbildung 10.6. In der Tabelle *adresse* sind Zeilen mit den Primärschlüsselwerten 4, 5 und 6 der Spalte *aid* enthalten, für die keine Fremdschlüsselwerte eingefügt wurden. Hier gibt es also keine Referenzierung auf Primärschlüsselwerte der Tabelle *auszubildender* und somit keine gültigen Beziehungen.

Für den **RIGHT OUTER JOIN** gilt ebenfalls, dass das Schlüsselwort **OUTER** optional anzugeben ist. In Listing 10.17 sehen Sie eine **SELECT**-Abfrage mit einem **RIGHT OUTER JOIN**, in dem auf das Schlüsselwort **OUTER** verzichtet wurde. Diese Abfrage liefert Ihnen exakt das gleiche Ergebnis wie die Abfrage aus Listing 10.16.

```
SELECT name,vorname,plz,ort
FROM auszubildender RIGHT OUTER JOIN adresse
ON ausid=fk_ausid;
```

Listing 10.16 Die Tabellen »auszubildender« und »adresse« mit einem **RIGHT OUTER JOIN** verbunden abfragen

```
SELECT name,vorname,plz,ort
FROM auszubildender RIGHT JOIN adresse
ON ausid=fk_ausid;
```

Listing 10.17 Einen **RIGHT OUTER JOIN** ohne das optionale Schlüsselwort **OUTER** verwenden

Zunächst einmal werden auch hier die Zeilen mit übereinstimmenden Schlüsselwerten verbunden abgefragt. Außerdem werden Zeilen der rechten Tabelle ausgegeben, die keine Einträge (nicht definierte Werte, also **NULL**-Werte) in der Fremdschlüsselspalte enthalten. Die Spalten der linken Tabelle *auszubildender* werden hier wieder mit **NULL**-Werten aufgefüllt.

Das Ergebnis der Abfrage sehen Sie in Tabelle 10.8:

name	vorname	plz	ort
Müller	Ralf	50827	Köln
Lang	Peter	50127	Bonn
Erde	Sabine	50129	Bonn

Tabelle 10.8 Ergebnis für eine Abfrage mit einem **RIGHT OUTER JOIN**

name	vorname	plz	ort
NULL	NULL	50827	Köln
NULL	NULL	50127	Bonn
NULL	NULL	50129	Bonn

Tabelle 10.8 Ergebnis für eine Abfrage mit einem RIGHT OUTER JOIN (Forts.)

Die Zeilen der Tabelle *adresse*, die keine Fremdschlüsselwerte enthalten, werden ebenfalls angezeigt. Wenn ein **RIGHT OUTER JOIN** verwendet wird und die Zeilen der rechten Tabelle nicht über eine Schlüsselbeziehung mit Zeilen der linken Tabelle verbunden werden können, füllt die Datenbank die Spaltenwerte der linken Tabelle wie beim **LEFT OUTER JOIN** mit NULL-Werten auf.

Vielleicht fragen Sie sich jetzt auch, wie Sie ausschließlich die Adressen ermitteln können, die keinem Auszubildenden zugeordnet sind. Wie beim **LEFT OUTER JOIN** prüfen wir beim **RIGHT OUTER JOIN** einfach wieder auf Spaltenwerte aus der linken Tabelle *auszubildender*, deren Zeilen mit keinem Wohnort verbunden werden konnten und somit mit NULL-Werten gefüllt wurden. In der **WHERE**-Klausel in Listing 10.18 werden die Spaltenwerte der Spalte *name* auf NULL geprüft, um nur die Zeilen in der Ergebnistabelle anzuzeigen, die nicht über eine Schlüsselreferenzierung mit Zeilen von Auszubildenden verbunden werden konnten. Sie können hier auch wieder jede beliebige Spalte der Tabelle *auszubildender* auf NULL-Werte prüfen.

```
SELECT name,vorname,plz,ort
FROM auszubildender RIGHT OUTER JOIN adresse
ON ausid=fk_ausid
WHERE name IS NULL;
```

Listing 10.18 Die Tabellen »auszubildender« und »adresse« mit einem RIGHT OUTER JOIN verbunden abfragen und nur die Zeilen ausgeben, die in der rechten Tabelle über keine Schlüsselwerte verfügen

Tabelle 10.9 zeigt Ihnen das Ergebnis der Abfrage aus Listing 10.18.

name	vorname	plz	ort
NULL	NULL	50827	Köln
NULL	NULL	50127	Bonn
NULL	NULL	50129	Bonn

Tabelle 10.9 Ergebnistabelle für eine Abfrage mit einem RIGHT OUTER JOIN und einer Bedingung, die eine Spalte der rechten Tabelle auf NULL-Werte prüft

Gemäß der Bedingung der **WHERE**-Klausel sind jetzt nur noch diejenigen Zeilen der Tabelle *adresse* in der Ergebnistabelle enthalten, die nicht über Schlüsselwerte mit Zeilen der Tabelle *auszubildender* verbunden werden konnten.

10.2.4 Zeilen mit einem FULL OUTER JOIN verbinden

Ein **FULL OUTER JOIN** verbindet die Zeilen aus zwei Tabellen, die über übereinstimmende Schlüsselwerte verfügen.

Mit einem **FULL OUTER JOIN** werden auch die Zeilen ermittelt, die über keine übereinstimmenden Schlüsselwerte der Tabellen verbunden werden können. Ein **FULL OUTER JOIN** ist also eine Mischung aus **INNER JOIN**, **RIGHT OUTER JOIN** und **LEFT OUTER JOIN**.



MySQL und MariaDB unterstützen keinen FULL OUTER JOIN

Die MySQL- und MariaDB-Datenbanken unterstützen keinen **FULL OUTER JOIN** gemäß dem SQL-Standard. Mit den Kenntnissen, die Sie bis jetzt erworben haben, ist es möglich, einen **FULL OUTER JOIN** zu simulieren, um das gleiche Ergebnis zu erhalten. Zum Simulieren eines **FULL OUTER JOINs** werden Sie im Anschluss an diesen Abschnitt die **UNION**-Klausel verwenden.

Falls Sie eine MySQL- oder MariaDB-Datenbank verwenden, sollten Sie diesen Abschnitt dennoch aufmerksam lesen, da ein **FULL OUTER JOIN** nun einmal zum Standard gehört und oft genutzt wird.

Um den **FULL OUTER JOIN** besser zu verstehen, betrachten wir wieder Abbildung 10.6. In der Tabelle *auszubildender* sind Zeilen enthalten, auf die nicht mit einem Fremdschlüsselwert aus der Tabelle *adresse* referenziert wird. In der Tabelle *adresse* wiederum sind Zeilen abgelegt, die keine Fremdschlüsselwerte enthalten. Somit kann hier kein Bezug aus der Tabelle *adresse* zur Tabelle *auszubildender* hergestellt werden. Das kennen Sie bereits aus den Beispielen zum **LEFT OUTER JOIN** und **RIGHT OUTER JOIN**. Ein **FULL OUTER JOIN** stellt eine Kombination aus diesen beiden dar. Wenn Sie also Tabellen mit einem **FULL OUTER JOIN** verbinden, erhalten Sie alle Zeilen mit übereinstimmenden Schlüsselwerten sowie die Zeilen der linken und der rechten Tabelle, in denen Schlüsselwerte nicht referenziert werden können.

In Listing 10.19 sehen Sie eine **SELECT**-Abfrage, die die Zeilen der Tabellen *auszubildender* und *adresse* über einen **FULL OUTER JOIN** verbindet:

```
SELECT name,vorname,plz,ort
FROM auszubildender FULL OUTER JOIN adresse
ON ausid=fk_ausid;
```

Listing 10.19 Mit einem **FULL OUTER JOIN** Zeilen aus Tabellen verbunden abfragen

In Tabelle 10.10 sehen Sie das Ergebnis der **SELECT**-Abfrage aus Listing 10.19. Dort finden Sie Zeilen, die über Schlüsselwerte verbunden wurden. Außerdem sehen Sie Zeilen mit Auszubildenden ohne passende Adresse und gleichzeitig Adressen, denen keine Auszubildenden zugeordnet werden konnten. Für die jeweils fehlenden Werte erhalten wir wieder nicht definierte Werte (**NULL**).

name	vorname	plz	ort
Müller	Ralf	50827	Köln
Klein	Sabine	NULL	NULL
Lang	Peter	50127	Bonn
Berg	Frank	NULL	NULL
Erde	Sabine	50129	Bonn
Grün	Justus	NULL	NULL
NULL	NULL	50129	Bonn
NULL	NULL	50127	Bonn
NULL	NULL	50827	Köln

Tabelle 10.10 Ergebnistabelle für eine Abfrage mit einem FULL OUTER JOIN

In den beiden letzten Abschnitten haben wir uns jeweils die Frage gestellt, wie wir die Zeilen eines Auszubildenden ermitteln, die keinem Wohnort zugeordnet sind, oder wie wir die Wohnorte ermitteln, die keinem Auszubildenden zuzuordnen sind. Wir haben eine Antwort auf diese Fragen gefunden, indem wir die betreffenden Zeilen mit einer **WHERE**-Klausel in einer **SELECT**-Abfrage gefiltert haben. Prüfen Sie einfach die Zeilen, in denen keine passende Eltern- oder Kindzeile mit einem Schlüssel verbunden werden kann, auf Gleichheit mit einem **NULL**-Wert.

In Listing 10.20 sehen Sie eine Abfrage, in der die **WHERE**-Klausel jeweils eine Spalte aus den Tabellen *auszubildender* und *adresse* auf Gleichheit mit **NULL** prüft. Wir haben es hier mit zwei Bedingungen zu tun, die mit einem logischen **OR**-Operator verknüpft werden. Das heißt, dass die **WHERE**-Klausel erfüllt ist, wenn eine der Bedingungen erfüllt ist. Oder anders gesagt: Die Bedingung ist wahr, wenn einer der Spaltenwerte der Spalten *name* oder *plz* gleich **NULL** ist.

```
SELECT name,vorname,plz,ort
FROM auszubildender FULL OUTER JOIN adresse
```

```
ON ausid=fk_ausid
where name IS NULL OR plz IS NULL;
```

Listing 10.20 In einem FULL OUTER JOIN eine Spalte der linken und eine Spalte der rechten Tabelle in einer WHERE-Klausel auf NULL prüfen

Tabelle 10.11 zeigt Ihnen das Ergebnis der Abfrage aus Listing 10.20. Die ersten drei Zeilen enthalten ausschließlich die Auszubildenden, denen nicht über passende Schlüsselwerte aus der Tabelle *adresse* Adressen zugeordnet werden können. Die Zeilen 4 bis 6 hingegen enthalten die Adressen, die aufgrund fehlender Schlüsselbeziehungen keinem Auszubildenden in der Tabelle *auszubildender* zugeordnet werden können.

Mit der Ergänzung einer **WHERE**-Klausel, in der jeweils Spaltenwerte der an einer **JOIN**-Verbindung beteiligten Tabellen auf **NULL** geprüft werden, können Sie prüfen, wo keine Beziehungen zwischen den Zeilen der beteiligten Tabellen existieren.

name	vorname	plz	ort
Klein	Sabine	NULL	NULL
Berg	Frank	NULL	NULL
Grün	Justus	NULL	NULL
NULL	NULL	50129	Bonn
NULL	NULL	50127	Bonn
NULL	NULL	50827	Köln

Tabelle 10.11 Ergebnistabelle für eine FULL OUTER JOIN-Abfrage, die mit einer WHERE-Klausel versehen wurde und prüft, ob eine Spalte je Tabelle gleich NULL ist

10.2.5 Einen FULL OUTER JOIN unter MySQL oder MariaDB nachbilden

Den Lesern, die eine MySQL- oder MariaDB-Datenbank verwenden, ist dieser Abschnitt gewidmet, damit sie einen **FULL OUTER JOIN** mithilfe einer **UNION**-Klausel nachbilden können. In Kapitel 5, »Mengenoperationen anwenden«, haben Sie gelernt, wie Sie Schnittmengen, Vereinigungsmengen und Differenzmengen von Zeilen bilden. Dieses Wissen werden wir uns jetzt zunutze machen, um einen **FULL OUTER JOIN** auf einer MySQL- oder MariaDB-Datenbank nachzubilden. Die in Abschnitt 10.2.4 dargestellten Beispiele werden wir in der gleichen Reihenfolge in unserer alternativen Lösung mit der **UNION**-Klausel nutzen, um einen **FULL OUTER JOIN** zu realisieren.

Betrachten Sie hierzu Listing 10.21. Es entspricht Listing 10.19. In dieser **SELECT**-Abfrage haben wir die Zeilen der Tabellen *auszubildender* und *adresse* über einen **FULL OUTER JOIN** verbunden.

```
SELECT name,vorname,plz,ort
FROM auszubildender FULL OUTER JOIN adresse
ON ausid=fk_ausid;
```

Listing 10.21 Mit einem **FULL OUTER JOIN** Zeilen aus Tabellen verbunden abfragen

Um einen **FULL OUTER JOIN** nachzubilden, betrachten Sie die zwei **SELECT**-Abfragen in Listing 10.22, die jeweils einen **LEFT OUTER JOIN** und einen **RIGHT OUTER JOIN** realisieren:

```
SELECT name,vorname,plz,ort
FROM auszubildender LEFT OUTER JOIN adresse
ON ausid=fk_ausid;
```

```
SELECT name,vorname,plz,ort
FROM auszubildender RIGHT OUTER JOIN adresse
ON ausid=fk_ausid;
```

Listing 10.22 **INNER JOIN**, **LEFT OUTER JOIN** und **RIGHT OUTER JOIN**

Die erste Abfrage aus Listing 10.22 verbindet die Zeilen der Tabellen *auszubildender* und *adresse* miteinander, und zwar über einen **LEFT OUTER JOIN**. Das bedeutet, dass alle Zeilen verbunden werden, die über gleiche Schlüsselwerte verfügen, und es werden die Zeilen aus der linken Tabelle mit aufgenommen, für die in der rechten Tabelle keine passenden Schlüsselwerte vorhanden sind.

Die zweite Abfrage aus Listing 10.22 verbindet die Zeilen der Tabellen *auszubildender* und *adresse* über einen **RIGHT OUTER JOIN**. Hier werden also alle Zeilen der rechten Tabelle mit allen Zeilen der linken Tabelle verbunden, die über gleiche Schlüsselwerte verfügen. Außerdem werden die Zeilen der rechten Tabelle mit ausgegeben, in denen keine Fremdschlüsselwerte festgelegt wurden.

Die zwei **SELECT**-Abfragen haben eines gemeinsam: Sie fragen die Spalten *name*, *vorname*, *plz* und *ort* ab. Erinnern Sie sich noch an die **UNION**-Klausel aus Abschnitt 5.2? Dort haben Sie gelernt, dass für eine Vereinigung von Zeilen von mehreren Tabellen die Spaltenanzahl der einzelnen Abfragen identisch sein muss. Außerdem müssen die Datentypen in der Sequenz der Spaltenangabe gleich sein. Diese Voraussetzung ist hier gegeben.

Sehen wir uns dennoch noch einmal die Ergebnistabellen der zwei **SELECT**-Abfragen an, damit Sie der Lösung auch vertrauen können.

Die erste Abfrage liefert, wie in Tabelle 10.12 gezeigt, die Ergebniszeilen, die aus einem **LEFT OUTER JOIN** für die Zeilen der Tabellen *auszubildender* und *adresse* resultieren:

name	vorname	plz	ort
Müller	Ralf	50827	Köln
Klein	Sabine	NULL	NULL
Lang	Peter	50127	Bonn
Berg	Frank	NULL	NULL
Erde	Sabine	50129	Bonn
Grün	Justus	NULL	NULL

Tabelle 10.12 Ergebnistabelle für eine Abfrage mit einem LEFT OUTER JOIN

Die zweite Abfrage liefert Ihnen, wie in Tabelle 10.13 gezeigt, die Ergebniszeilen, die aus einem **RIGHT OUTER JOIN** für die Zeilen der Tabellen *auszubildender* und *adresse* resultieren:

name	vorname	plz	ort
Müller	Ralf	50827	Köln
Lang	Peter	50127	Bonn
Erde	Sabine	50129	Bonn
NULL	NULL	50827	Köln
NULL	NULL	50127	Bonn
NULL	NULL	50129	Bonn

Tabelle 10.13 Ergebnistabelle für eine Abfrage mit einem RIGHT OUTER JOIN

Die Übersicht der zwei Ergebnistabellen zeigt Ihnen, dass wir mit einer Vereinigung der Ergebniszeilen aus den zwei Ergebnistabellen einen **FULL OUTER JOIN** in seiner Funktionalität nachbilden können.

In Listing 10.23 sehen Sie die Vereinigung der Ergebniszeilen der zwei Abfragen, die mit einer **UNION**-Klausel realisiert wird. Es werden einfach die Zeilen der Ergebnistabellen der zwei **SELECT**-Abfragen vereinigt, die jeweils mit einer **LEFT OUTER JOIN**-Klausel und einer **RIGHT OUTER JOIN**-Klausel ausgestattet sind.

```
SELECT name,vorname,plz,ort
FROM auszubildender LEFT OUTER JOIN adresse
```

```

ON ausid=fk_ausid
UNION
SELECT name,vorname,plz,ort
FROM auszubildender RIGHT OUTER JOIN adresse
ON ausid=fk_ausid;

```

Listing 10.23 Einen FULL OUTER JOIN mit UNION nachbilden

Wie bei einem **FULL OUTER JOIN** erhalten Sie als Ergebnis eine Tabelle, die die Zeilen enthält, die über Schlüsselwerte (**LEFT** und **RIGHT OUTER JOIN**) verbunden werden konnten. Die Zeilen der linken Tabelle, die mit keinem Schlüsselwert der rechten Tabelle übereinstimmen, und die Zeilen der rechten Tabelle, für die kein Fremdschlüsselwert festgelegt wurde, sind ebenfalls Bestandteil des Ergebnisses.

Die Zeilen, die ausschließlich in der linken oder rechten Tabelle enthalten sind, werden von der Datenbank automatisch mit **NULL**-Werten aufgefüllt. Das Auffüllen mit **NULL**-Werten ist dem **LEFT OUTER JOIN** und dem **RIGHT OUTER JOIN** geschuldet. Der **LEFT** und der **RIGHT OUTER JOIN** liefern uns auch zweimal die Ergebniszeilen, die einem **INNER JOIN** entsprächen. Hier sorgt der **UNION**-Vereinigungsoperator für Eindeutigkeit.

Das Ergebnis sehen Sie in Tabelle 10.14:

name	vorname	plz	ort
Müller	Ralf	50827	Köln
Lang	Peter	50127	Bonn
Erde	Sabine	50129	Bonn
Klein	Sabine	NULL	NULL
Berg	Frank	NULL	NULL
Grün	Justus	NULL	NULL
NULL	NULL	50827	Köln
NULL	NULL	50127	Bonn
NULL	NULL	50129	Bonn

Tabelle 10.14 Einen FULL OUTER JOIN mit UNION nachbilden

Als Nächstes werden wir, wie in den anderen Beispielen, mit einer **WHERE**-Klausel filtern, welche Zeilen der Tabellen *auszubildender* und *adresse* nicht über Schlüsselbeziehungen verbunden werden können.

In Listing 10.24 sehen Sie, dass hinter den einzelnen **SELECT**-Anweisungen mit einer **WHERE**-Klausel jeweils eine Spalte der rechten bzw. linken Tabelle auf **NULL** geprüft wird, um nur die Zeilen zu ermitteln, die nicht über Schlüsselwerte verbunden werden können:

```
SELECT name,vorname,plz,ort
FROM auszubildender LEFT OUTER JOIN adresse
ON ausid=fk_ausid
WHERE plz IS NULL
UNION
SELECT name,vorname,plz,ort
FROM auszubildender RIGHT OUTER JOIN adresse
ON ausid=fk_ausid
WHERE name IS NULL;
```

Listing 10.24 Die Zeilen ermitteln, die nicht über einen JOIN verbunden werden konnten

Tabelle 10.15 zeigt Ihnen das Ergebnis der Abfrage aus Listing 10.24. Die ersten drei Zeilen enthalten – wie im Beispiel aus Abschnitt 10.2.4 – wieder ausschließlich die Auszubildenden, denen nicht über passende Schlüsselwerte aus der Tabelle *adresse* Adressen zugeordnet werden können. Die Zeilen 4 bis 6 hingegen enthalten wieder die Adressen, die aufgrund fehlender Schlüsselbeziehungen keinem Auszubildenden in der Tabelle *auszubildender* zugeordnet werden können.

name	vorname	plz	ort
Klein	Sabine	NULL	NULL
Berg	Frank	NULL	NULL
Grün	Justus	NULL	NULL
NULL	NULL	50827	Köln
NULL	NULL	50127	Bonn
NULL	NULL	50129	Bonn

Tabelle 10.15 Nur die Zeilen ermitteln, die nicht verbunden abgefragt werden konnten

10.2.6 Zeilen mit einem **CROSS JOIN** verbinden

Ein **CROSS JOIN** verbindet völlig unabhängig von Schlüsselbeziehungen zwischen Eltern- und Kindtabellen die Zeilen einer Tabelle mit allen Zeilen einer anderen Tabelle. Betrachten Sie eine Zeile der linken Tabelle, so wird diese Zeile mit jeder Zeile der rechten Tabelle verbunden.

Mathematisch betrachtet handelt es sich hierbei um ein kartesisches Produkt. Sie können **CROSS JOINS** verwenden, um Testdaten zu generieren oder um alle Zeilen von zwei Tabellen miteinander verbunden auszugeben. Wenn Sie Testdaten generieren wollen, so reicht eine Tabelle mit 20.000 Zeilen aus, die Sie zum Beispiel mit einer Tabelle, die 100 Zeilen enthält, verbunden abfragen. So generieren Sie eine Ergebnistabelle mit Testdaten, die 2.000.000 Zeilen enthält.

Um Ihnen die Funktion eines **CROSS JOINS** möglichst einfach zu erklären, bleiben wir aber bei den Auszubildenden, denen Adressen zugeordnet sind.

Abbildung 10.7 sehen Sie, dass die erste Zeile der Tabelle *adresse* mit allen Zeilen der Tabelle *auszubildender* verbunden wird. Somit gilt auch, dass jede Zeile der rechten Tabelle *auszubildender* mit allen Zeilen der Tabelle *adresse* verbunden ist.

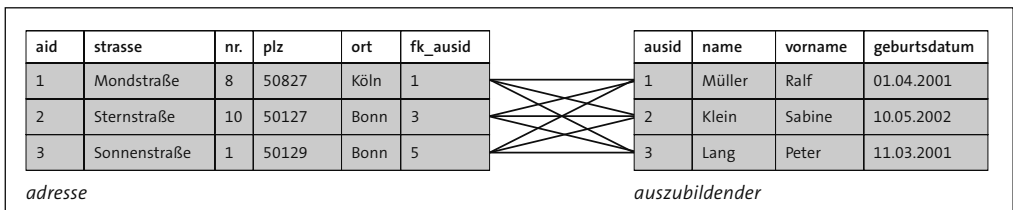


Abbildung 10.7 Zeilen von Tabellen, die über einen CROSS JOIN verbunden abgefragt werden

Abbildung 10.7 zeigte Beispiel gut, um Sie mit dem **CROSS JOIN** vertraut zu machen.

Um eine **SELECT**-Abfrage über zwei Tabellen mit einem **CROSS JOIN** auszustatten, notieren Sie zwischen den Tabellen *auszubildender* und *adresse* die Schlüsselwörter **CROSS JOIN**. Das war es schon, da ein **CROSS JOIN** ja lediglich alle Zeilen einer Tabelle mit allen Zeilen einer anderen Tabelle verbindet, und das auch umgekehrt. Es wird also keine Bedingung (**ON**-Klausel mit Bedingung) benötigt, die Schlüsselwerte auf Gleichheit prüft. In dieser **SELECT**-Abfrage wurde noch die Spalte *strasse* mit in die Spaltenauswahlliste aufgenommen. So können Sie leichter sehen, dass tatsächlich alle Zeilen einer Tabelle mit allen Zeilen einer anderen Tabelle verbunden werden und umgekehrt.

```
SELECT name,vorname,plz,ort,strasse
FROM auszubildender CROSS JOIN adresse;
```

Listing 10.25 Zeilen aus zwei Tabellen mit einem CROSS JOIN verbunden abfragen

Tabelle 10.16 zeigt Ihnen das Ergebnis der **SELECT**-Abfrage aus Listing 10.25.

name	vorname	plz	ort	strasse
Müller	Ralf	50827	Köln	Mondstraße
Klein	Sabine	50827	Köln	Mondstraße
Lang	Peter	50827	Köln	Mondstraße
Berg	Frank	50827	Köln	Mondstraße
Erde	Sabine	50827	Köln	Mondstraße
Grün	Justus	50827	Köln	Mondstraße
Müller	Ralf	50127	Bonn	Sternstraße
Klein	Sabine	50127	Bonn	Sternstraße
Lang	Peter	50127	Bonn	Sternstraße
Berg	Frank	50127	Bonn	Sternstraße

Tabelle 10.16 Auszug aus der Ergebnistabelle für eine SELECT-Abfrage mit einem CROSS JOIN

Sie sehen, dass jede Zeile der Tabelle *auszubildender* mit jeder Zeile der Tabelle *adresse* ausgegeben wird. Wenn Sie die Abfrage nachvollziehen, werden Sie sehen, dass aus den sechs Zeilen der Tabelle *auszubildender*, in denen die Auszubildenden hinterlegt werden, und den sechs Zeilen der Tabelle *adresse*, in denen Wohnorte gespeichert sind, dank einem **CROSS JOIN** eine Ergebnistabelle mit 36 Zeilen wird.

In Tabelle 10.16 ist lediglich ein Auszug aus der Ergebnistabelle zu sehen. Um zu prüfen, ob es sich tatsächlich um 36 Zeilen handelt, sollten Sie das Beispiel praktisch nachvollziehen.

10.2.7 Zeilen von drei Tabellen mit einem INNER JOIN verbinden

Sie erinnern sich bestimmt, dass ich Ihnen in Abschnitt 7.4 die Ausbildungsdatenbank als Modell in der UML-Notation vorgestellt habe. Hier stehen die Tabellen *ausbildungsberuf*, *beruflehrfach* und *lehrfach* in Beziehung zueinander.

Die Werte der Fremdschlüsselspalten *fk_berufsid* und *fk_lehrfachid* aus der Tabelle *beruflehrfach* zeigen jeweils auf Werte der Primärschlüsselspalten *berufsid* und *lehrfachid* der Tabellen *ausbildungsberuf* und *lehrfach*. Somit sind die optimalen Voraussetzungen gegeben, um mit einem **INNER JOIN** Zeilen aus drei Tabellen zu verbinden und abzufragen.

In Listing 10.26 sehen Sie eine **SELECT**-Abfrage, in der ein **INNER JOIN** über die drei Tabellen *ausbildungsberuf*, *beruflehrfach* und *lehrfach* realisiert wird.

```
SELECT berufsbezeichnung, lehrfach
FROM ausbildungsberuf
INNER JOIN beruflehrfach
ON berufsids=fk_berufsids
INNER JOIN lehrfach
ON fk_lehrfachids=lehrfachids;
```

Listing 10.26 Zeilen von drei Tabellen mit einem **INNER JOIN** verbunden abfragen

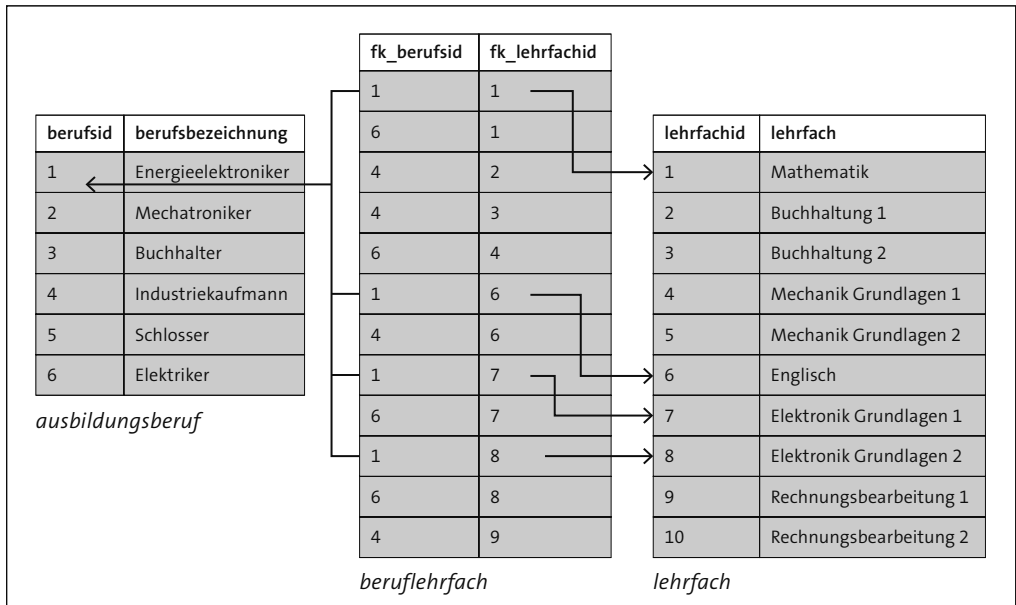


Abbildung 10.8 Drei Tabellen, die zueinander in Beziehung stehen

Vom Grundsatz her habe ich den **INNER JOIN** bereits in Abschnitt 10.2.1 behandelt. Der Unterschied in Listing 10.26 besteht zunächst darin, dass hinter der ersten **ON**-Klausel ein weiterer **INNER JOIN** angegeben wird, der die Tabelle *lehrfach* verbindet. In der Bedingung der **ON**-Klausel wird auf Gleichheit zwischen den Spaltenwerten der Fremdschlüsselspalte *fk_lehrfach* und der Primärschlüsselspalte *lehrfachids* geprüft.

Zunächst wird also mit der ersten **ON**-Klausel auf Gleichheit zwischen den Spaltenwerten der Spalten *berufsids* und *fk_berufsids* geprüft und somit eine Verbindung zwischen den Tabellen *ausbildungsberuf* und *beruflehrfach* geschaffen.

Dann wird in der zweiten **ON**-Klausel auf Gleichheit zwischen den Spaltenwerten der Spalten *fk_lehrfach* und *lehrfachid* geprüft und somit eine weitere Verbindung zwischen den Tabellen *beruflehrfach* und *lehrfach* geschaffen.



Mehrere Tabellen verbinden

Sie können natürlich auch mehr als drei Tabellen mit einem **INNER JOIN** verbunden abfragen. Sie müssen lediglich hinter der letzten **ON**-Klausel eine weitere **INNER JOIN**-Klausel mit der dazugehörigen **ON**-Klausel anfügen.

In Tabelle 10.17 sehen Sie das Ergebnis der **SELECT**-Abfrage aus Listing 10.26. Die Abfrage liefert uns verbunden über die Schlüsseltabelle *beruflehrfach* die Berufsbezeichnungen und die jeweiligen Lehrfächer als Ergebnistabelle zurück.

berufsbezeichnung	lehrfach
Energieelektroniker	Mathematik
Energieelektroniker	Englisch
Energieelektroniker	Elektronik Grundlagen 1
Energieelektroniker	Elektronik Grundlagen 2
Industriekaufmann	Buchhaltung 1
Industriekaufmann	Buchhaltung 2
Industriekaufmann	Englisch
Industriekaufmann	Rechnungsbearbeitung 1

Tabelle 10.17 Ergebnistabelle für eine verbundene Abfrage aus drei Tabellen

10.2.8 Spalten in einem JOIN über Tabellennamen referenzieren

Bisher haben Sie Zeilen von Tabellen verbunden, deren Spaltenbezeichnungen unterschiedlich sind. Stellen Sie sich vor, Sie formulieren einen **INNER JOIN** über zwei Tabellen, in denen Spalten mit gleichen Spaltennamen vorkommen. Wenn gleichnamige Spaltennamen in der Spaltenauswahlliste einer **SELECT**-Anweisung enthalten sind, kann nicht zwischen den Spalten unterschieden werden. Wenn gleichnamige Spalten in zwei oder mehreren Tabellen vorkommen, kann die Datenbank also nicht unterscheiden, welche Spalte aus welcher Tabelle ausgewählt werden soll.

Das Problem ist allerdings leicht zu lösen: Schreiben Sie die jeweiligen Spaltennamen in der Spaltenauswahlliste in der Notation *tabellenname.spaltenname*. Zuerst notieren Sie also die Tabelle, gefolgt von einem Punkt, und schließlich dem Spaltennamen.

Sehen wir uns als Nächstes ein Beispiel an. In Abbildung 10.9 sehen Sie einen Ausschnitt unseres Datenmodells für eine Ausbildungsdatenbank in der *UML-Notation*.

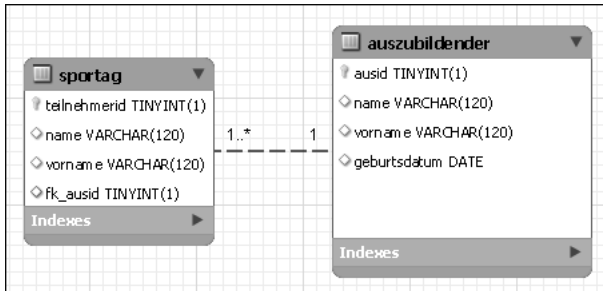


Abbildung 10.9 Tabellen, die identische Spaltennamen enthalten

Das Datenmodell wurde um die Tabelle *sportag* (Sport-Arbeitsgruppe) erweitert. In der Tabelle sind die Spalten *teilnehmerid*, *name*, *vorname* und *fk_ausid* enthalten. Die Spalte *fk_ausid* repräsentiert hier wieder die Fremdschlüsselspalte. Die Spaltennamen *name* und *vorname* kommen in den Tabellen *auszubildender* und *sportag* vor.

Um das Beispiel nachvollziehen zu können, verwenden wir die **CREATE TABLE**-Anweisung aus Listing 10.27, die die Tabelle *sportag* in der *ausbildungsdatenbank* anlegt:

```

CREATE TABLE sportag (
    teilnehmerid SMALLINT PRIMARY KEY NOT NULL,
    name VARCHAR(120),
    vorname VARCHAR(120),
    fk_ausid SMALLINT,
    CONSTRAINT fk_ausid_auszubildender
        FOREIGN KEY (fk_ausid)
            REFERENCES auszubildender (ausid)
);
  
```

Listing 10.27 CREATE TABLE-Anweisung für die Tabelle »sportag«

Listing 10.28 enthält zwei **INSERT**-Anweisungen, die benötigt werden, um zwei Zeilen in die Tabelle *sportag* einzufügen:

```

INSERT INTO sportag (teilnehmerid,name,vorname,fk_ausid)
VALUES (1,'Müller','Ralf',1);

INSERT INTO sportag (teilnehmerid,name,vorname,fk_ausid)
VALUES (2,'Klein','Sabine',2);
  
```

Listing 10.28 Zeilen in die Tabelle »sportag« einfügen

In Abbildung 10.10 sehen Sie die Tabellen *sportag* und *auszubildender*. Sie enthalten beide die Spaltennamen *name* und *vorname*.

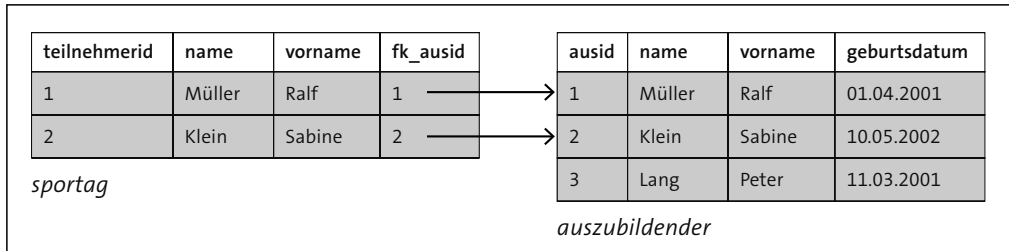


Abbildung 10.10 Tabellen mit gleichen Spaltennamen

Als Nächstes betrachten wir eine **SELECT**-Abfrage mit einem **INNER JOIN**, in dem die Spalten *name* und *vorname* abgefragt werden sollen. Außerdem soll aus der Tabelle *auszubildender* die Spalte *geburtsdatum* abgefragt werden.

```
SELECT name,vorname,geburtsdatum
FROM auszubildender INNER JOIN sportag
ON ausid=fk_ausid;
```

Listing 10.29 Mit einem **INNER JOIN** die Tabellen »auszubildender« und »sportag« verbunden abfragen

Die **SELECT**-Abfrage aus Listing 10.29 führt zu einem Fehler, weil das Datenbanksystem die Spalten *name* und *vorname* nicht eindeutig einer Tabelle zuordnen kann. Die MySQL- und die MariaDB-Datenbank reagieren auf mehrfach gleich vorkommende Spaltennamen mit der Fehlermeldung:

Error Code: 1052. Column 'name' in field list is ambiguous

Die Lösung für dieses Problem sehen Sie in Listing 10.30. Dort werden sämtliche Spalten jeweils über einen Tabellennamen referenziert. Jetzt kann die Datenbank die jeweiligen Spalten den Tabellen zuordnen, und die Mehrdeutigkeit ist aufgehoben.

```
SELECT
    auszubildender.name,
    auszubildender.vorname,
    auszubildender.geburtsdatum
FROM auszubildender INNER JOIN sportag
ON auszubildender.ausid=sportag.fk_ausid;
```

Listing 10.30 Spalten über einen Tabellennamen referenzieren

Tabelle 10.18 zeigt das Ergebnis der Abfrage. Die **SELECT**-Abfrage konnte jetzt problemlos ausgeführt werden.

name	vorname	geburtsdatum
Müller	Ralf	2001-04-01
Klein	Sabine	2002-05-10

Tabelle 10.18 Ergebnistabelle für eine INNER JOIN-Abfrage mit mehrdeutigen Spaltennamen

In diesem Abschnitt haben Sie erfahren, wie Sie Spaltennamen aus zwei oder mehreren Tabellen über Tabellennamen referenzieren. Fällt Ihnen vielleicht auf, dass es noch einen anderen Weg gibt, das Problem der übereinstimmenden Bezeichnungen zu lösen?

Es wäre auch möglich gewesen, ohne die Spalten *name* und *vorname* in der Tabelle *sportag* auszukommen. Über die Fremdschlüsselbeziehung ist ein Zugriff auf die Spalten *name* und *vorname* in der Tabelle *auszubildender* ja bereits gegeben. Durch aufmerksames Nachdenken, welche Spalten überhaupt notwendig sind, hätte sich das Hindernis also auch umgehen lassen. Hier wollte ich Ihnen aber zeigen, wie Sie mit gleichen Spaltennamen umgehen, und deswegen habe ich dieses Beispiel gewählt.

10.2.9 Spalten in einem JOIN über Tabellenaliasse referenzieren

Wenn Sie Tabellen verbunden abfragen, können Sie hinter den Tabellennamen in der **SELECT**-Anweisung einen Tabellenalias notieren. Mit dem vergebenen Tabellenalias können Sie wiederum auf die Spalten der jeweiligen Tabelle referenzieren. Listing 10.31 ist funktionell identisch mit der **SELECT**-Abfrage aus Listing 10.30:

```
SELECT a.name,a.vorname,a.geburtsdatum
FROM auszubildender a INNER JOIN sportag s
ON a.ausid=s.fk_ausid;
```

Listing 10.31 Spalten über Tabellenaliasse referenzieren

In Listing 10.31 werden Tabellenaliasse verwendet, die direkt hinter dem Tabellennamen notiert werden. Der Tabelle *auszubildender* ist der Tabellenalias *a* zugeordnet, der Tabelle *sportag* wiederum der Tabellenalias *s*.

Die festgelegten Tabellenaliasse *a* und *s* werden in der Spaltenauswahlliste der **SELECT**-Anweisung und in der **ON**-Klausel verwendet. Wenn Sie Spalten über einen Tabellenalias referenzieren, ersparen Sie sich eine Menge Schreibarbeit, und die Abfrage wird übersichtlicher. In der Regel werden Spalten aus Tabellen, die mit einem **JOIN** verbunden sind, über Tabellenaliasse referenziert.

Das Ergebnis ist exakt identisch mit dem Ergebnis, das aus Listing 10.31 resultiert. Aus dem Grund verzichte ich hier auf eine Ergebnistabelle.



Tabellenaliasse in der ON-Klausel

Ein Tabellenalias kann in einer Spaltenauswahlliste verwendet werden. Ein weiterer Anwendungsfall für Tabellenaliasse ist die **ON**-Klausel, da es dort mehrfach gleich vergebene Spaltennamen geben kann.

In Kapitel 2 haben Sie erfahren, wie Sie Spalten einen Spaltenalias zuordnen. In den Tabellen *auszubildender* und *sportag* sind jeweils die Spalten *name* und *vorname* enthalten. Zwei gleichnamige Spalten, die aus einem **JOIN** hervorgehen, sollten mit Spaltenaliasen abgefragt werden, um die Spalten auseinanderzuhalten, die ja aus unterschiedlichen Tabellen kommen. Sehen wir uns als Nächstes ein Beispiel an, in dem keine Spaltenaliasse verwendet und die gleichnamigen Spalten aus den Tabellen *auszubildender* und *sportag* abgefragt werden:

```
SELECT a.name,a.vorname,a.geburtsdatum,s.name,s.vorname
FROM auszubildender a INNER JOIN sportag s
ON a.ausid=s.fk_ausid;
```

Listing 10.32 Tabellen mit gleichen Spaltennamen mit einem **INNER JOIN** abfragen

Tabelle 10.19 zeigt das Ergebnis der Abfrage aus Listing 10.32.

name	vorname	geburtsdatum	name	vorname
Müller	Ralf	2001-04-01	Müller	Ralf
Klein	Sabine	2002-05-10	Klein	Sabine

Tabelle 10.19 Ergebnistabelle mit gleichen Spaltennamen

Sie sehen, dass Sie die jeweils doppelt vorkommenden Spalten *name* und *vorname* anhand des Namens keiner Tabelle zuordnen können. Hier helfen uns die Spaltenaliasse, die Sie in Abschnitt 2.7 kennengelernt haben.

In Listing 10.33 sehen Sie eine **SELECT**-Abfrage, in der die Spalten der Spaltenauswahlliste jeweils mit einem Spaltenalias versehen wurden:

```
SELECT
a.name AS aus_name,
a.vorname AS aus_vorname,
a.geburtsdatum AS aus_gebdatum,
s.name AS spag_name,
s.vorname AS spag_vorname
```

```
FROM auszubildender a INNER JOIN sporttag s
ON a.ausid=s.fk_ausid;
```

Listing 10.33 Doppelt vorkommende Spalten in einem JOIN mit Spaltenaliassen versehen

Als Resultat erhalten wir eine Ergebnistabelle, in der wir die Spalten eindeutig den Tabellen zuordnen können, aus denen sie stammen.

aus_name	aus_vorname	aus_gebdatum	spag_name	spag_vorname
Müller	Ralf	2001-04-01	Müller	Ralf
Klein	Sabine	2002-05-10	Klein	Sabine

Tabelle 10.20 Ergebnistabelle mit Spaltenaliassen

10.2.10 Zeilen mit einem Self Join verbinden

Bevor wir uns mit dem *Self Join* im Detail befassen, müssen wir klären, wozu er verwendet wird. Mit einem Self Join können Sie hierarchische Strukturen einer Tabelle abbilden, die über die entsprechenden Voraussetzungen verfügt. Um hierarchische Strukturen in einer Tabelle abzubilden, verbinden Sie die Zeilen einer Tabelle mit den Zeilen ebendieser einen Tabelle. Um dies zu realisieren, nutzen Sie eine Fremdschlüsselbeziehung. Der Unterschied ist der, dass der Fremdschlüssel nicht auf einen Primärschlüssel in einer anderen Tabelle verweist, sondern auf den Primärschlüssel, der in der gleichen Tabelle definiert wurde wie der Fremdschlüssel. Sie nutzen also ausschließlich eine Tabelle, um ihre Zeilen wieder miteinander zu verbinden. Eine Tabelle stellt also die Eltern- und die Kindtabelle gleichzeitig dar.

Um einen Self Join besser zu verstehen, ist es gut, die abzufragende Tabelle, deren Zeilen mit einem **JOIN** verbunden werden sollen, wie zwei eigenständige Tabellen zu betrachten. Wir stellen uns also vor, dass es sich um zwei verschiedene Tabellen handelt. In der ersten Tabelle finden wir eine Primärschlüsselspalte vor. In der zweiten Tabelle befindet sich eine Fremdschlüsselspalte, die wiederum auf die Primärschlüsselspalte der ersten Tabelle verweist.

Der Self Join ist keine SQL-Klausel

Bitte beachten Sie, dass der Self Join keine SQL-Klausel im klassischen Sinne ist. Es handelt sich lediglich um eine Bezeichnung für einen **JOIN**, der ausschließlich auf eine Tabelle angewendet wird. Einen Self Join können Sie mit einem **INNER JOIN**, einem **LEFT OUTER JOIN**, einem **RIGHT OUTER JOIN** und mit einem **FULL OUTER JOIN** realisieren.



In Abbildung 10.11 sehen Sie so eine Beziehung, die wir uns vorgestellt haben.

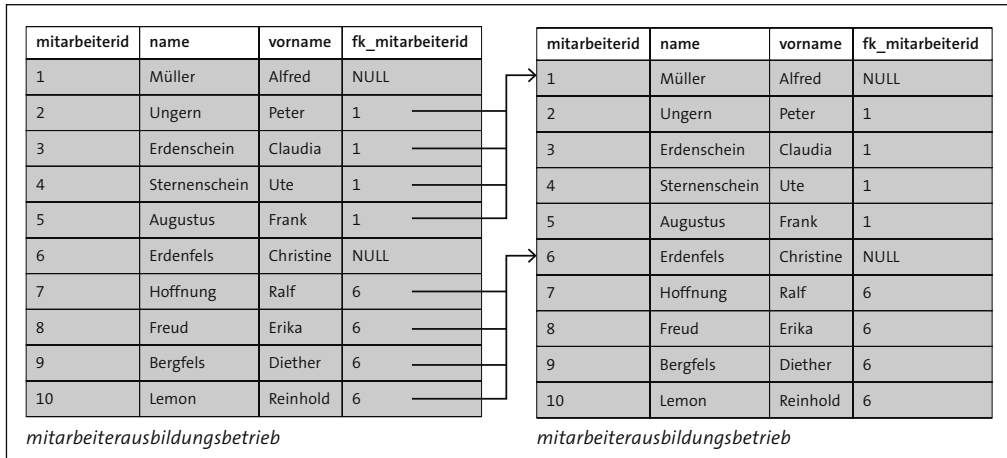


Abbildung 10.11 Eine Tabelle mit Zeilen, die auf Zeilen in der gleichen Tabelle referenzieren

In der linken Tabelle sehen Sie die Primärschlüsselspalte *mitarbeiterid*; in der rechten Tabelle sehen Sie die Fremdschlüsselspalte *fk_mitarbeiterid*. Die Fremdschlüsselwerte der Spalte *fk_mitarbeiterid* der rechten Tabelle verweisen wiederum auf die Primärschlüsselwerte der Spalte *mitarbeiterid*.

Mit diesen Erkenntnissen können wir einen **INNER JOIN** verwenden, der ermittelt, welche Mitarbeiter welchen Vorgesetzten untergeordnet sind. Den Mitarbeitern Alfred Müller und Christine Erdenfels sind in diesem Beispiel gleichzeitig die Rollen des Vorgesetzten und Inhabers zugeordnet, da keine Fremdschlüsselwerte (NULL) für diese Mitarbeiter existieren.

Als Nächstes sehen wir uns Listing 10.34 an:

```
SELECT
    ma1.mitarbeiterid AS vorgesetzterid,
    ma1.name AS vorgesetztername,
    ma2.name AS mitarbeitername,
    ma2.mitarbeiterid AS mitarbeiterid
FROM mitarbeiterausbildungsbetrieb ma1 INNER JOIN
    mitarbeiterausbildungsbetrieb ma2
ON ma1.mitarbeiterid=ma2.fk_mitarbeiterid;
```

Listing 10.34 Eine Tabelle mit sich selbst verbunden abfragen

Hier werden mit einem **INNER JOIN** die Zeilen einer Tabelle mit sich selbst verbunden abgefragt. Daraus ergibt sich die Notwendigkeit, Tabellenaliasse zu verwenden. Ohne die Tabellenaliasse könnte die Datenbank, wie im letzten Abschnitt beschrieben, nicht unterscheiden, welche Spalten zu welcher Tabelle gehören.

Sonst entspricht der **INNER JOIN** den Regeln, die Sie bereits kennengelernt haben.

In Tabelle 10.21 sehen Sie das Ergebnis der Abfrage.

vorgesetzterid	vorgesetztername	mitarbeitername	mitarbeiterid
1	Müller	Ungern	2
1	Müller	Erdenschein	3
1	Müller	Sternenschein	4
1	Müller	Augustus	5
6	Erdenfels	Hoffnung	7
6	Erdenfels	Freud	8
6	Erdenfels	Bergfels	9
6	Erdenfels	Lemon	10

Tabelle 10.21 Ergebnistabelle für einen INNER JOIN, der Zeilen einer Tabelle miteinander verbindet

Die Zeilen der Vorgesetzten (Tabellenalias *ma1*) wurden immer dann mit den Zeilen der Mitarbeiter (Tabellenalias *ma2*) verbunden, wenn die Fremdschlüsselwerte, die auf die Mitarbeiter zeigen, gleich den Primärschlüsselwerten der Vorgesetzten sind.

10.2.11 Zeilen mit einem INNER JOIN ohne Schlüsselvergleich verbinden

Bisher haben Sie **JOIN**-Typen in einer **SELECT**-Abfrage verwendet, um die Zeilen von zwei oder mehreren Tabellen miteinander zu verbinden. Die **JOINS**, die wir bisher formuliert haben, wurden stets über gültige Schlüsselbeziehungen ausgewertet, die bereits in der **CREATE TABLE**-Anweisung festgelegt waren. Es war also immer eine Bedingung vorhanden, die in der **ON**-Klausel genutzt werden konnte, um Spaltenwerte zu vergleichen.

In diesem Abschnitt zeige ich Ihnen, dass Sie einen beliebigen **JOIN**-Typ unabhängig von der Existenz von Fremdschlüsselbeziehungen verwenden können. In der **ON**-Klausel können beliebige Spalten aus zwei Tabellen verglichen werden. Das funktioniert allerdings nur, wenn der Datentyp der zu vergleichenden Spaltenwerte gleich ist oder implizit von der Datenbank konvertiert werden kann. In Abschnitt 4.3 habe ich Ihnen den Unterschied zwischen einer expliziten und einer impliziten Typkonvertierung erklärt. Jetzt ist für Sie wichtig zu wissen, dass in der Bedingung der **ON**-Klausel Spaltenwerte des gleichen Typs ausgewertet werden.

Sehen wir uns als Erstes das Beispiel in Abbildung 10.12 an.

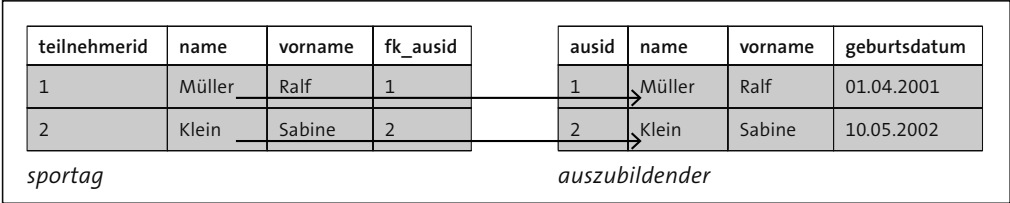


Abbildung 10.12 Verbindungen zwischen Zeilen mit Nicht-Schlüsselspalten realisieren

Hier sehen Sie die bekannten Tabellen *auszubildender* und *sporttag*, in denen jeweils die Spalte *name* enthalten ist. Die Datentypen sind in beiden Tabellen gleich, nämlich **VARCHAR(120)**. Somit hindert uns nichts daran, diese beiden Spalten wie Schlüsselspalten in einer Bedingung einer **ON**-Klausel zu verwenden. Wir könnten also einen **INNER JOIN** formulieren, um die Spalte *name*, die hier in beiden Tabellen vorhanden ist, in einer Bedingung auszuwerten. Auch hier werden die Zeilen der Tabellen nur dann verbunden ausgegeben, wenn die Bedingung in der **ON**-Klausel erfüllt ist.

Kommen wir zu einem konkreten Beispiel:

```
SELECT a.name, a.vorname, a.geburtsdatum, s.teilnehmerid
FROM auszubildender a INNER JOIN sporttag s
ON a.name=s.name;
```

Listing 10.35 Ein **INNER JOIN** mit einer Bedingung, die Spalten in der **ON**-Klausel auswertet, die keine Schlüsselspalten sind

In Listing 10.35 ist eine **SELECT**-Abfrage mit einem **INNER JOIN** zu sehen, die in der **ON**-Klausel die Spalte *name* der beiden Tabellen *auszubildender* und *sporttag* auf Gleichheit prüft.

In Tabelle 10.22 sehen Sie das Ergebnis der **SELECT**-Abfrage aus Listing 10.35. Die Zeilen der Tabellen *auszubildender* und *sporttag* werden immer dann miteinander verbunden und in die Ergebnistabelle mit aufgenommen, wenn die Werte der Spalte *name* der Bedingung in der **ON**-Klausel entsprechen.

name	vorname	geburtsdatum	teilnehmerid
Müller	Ralf	2001-04-01	1
Klein	Sabine	2002-05-10	2

Tabelle 10.22 Ergebnis eines **INNER JOINs**, der ohne Schlüsselspalten gebildet wurde

[*]

Zusammenfassung: Zeilen, die in Beziehung stehen, abfragen

Mit folgenden **JOIN**-Typen können Tabellen verbunden abgefragt werden:

- ▶ **INNER JOIN** – verbindet ausschließlich Zeilen mit gültigen Schlüsselwerten.
- ▶ **LEFT OUTER JOIN** – verbindet Zeilen mit gültigen Schlüsselwerten und gibt Zeilen der linken Tabelle aus, die nicht verbunden abgefragt werden können.
- ▶ **RIGHT OUTER JOIN** – verbindet Zeilen mit gültigen Schlüsselwerten und gibt Zeilen der rechten Tabelle aus, die nicht verbunden abgefragt werden können.
- ▶ **FULL OUTER JOIN** – verbindet Zeilen mit gültigen Schlüsselwerten und gibt die Zeilen der linken und rechten Tabelle aus, die nicht über Schlüsselwerte verbunden werden können.
- ▶ **CROSS JOIN** – verbindet jede Zeile einer Tabelle mit jeder Zeile einer anderen Tabelle und umgekehrt.
- ▶ **Self Join** – verbindet eine Zeile einer Tabelle mit Zeilen der gleichen Tabelle.

Die Verbindung zwischen den Zeilen wird (bis auf den **CROSS JOIN** mit einer Bedingung) stets über Primärschlüsselwerte und darauf referenzierende Fremdschlüsselwerte hergestellt.

Die hier aufgeführten **JOIN**-Typen (bis auf den **CROSS JOIN**) können nicht nur über Schlüsselbeziehungen realisiert werden. Solange die Datentypen der Spalten, die an einem **JOIN** beteiligt sind, identisch sind, kann mit einer Bedingung in einer **ON**-Klausel eine Verbindung hergestellt werden.

10.2.12 Übungen

In den folgenden Übungen werden Sie Zeilen aus zwei oder mehreren Tabellen verbunden abfragen. Um Ihnen die Formulierung der **SELECT**-Abfragen zu erleichtern, sehen Sie in Abbildung 10.13 noch einmal das Datenbankmodell der Bibliotheksdatenbank, das Sie bereits in Kapitel 9 kennengelernt haben.

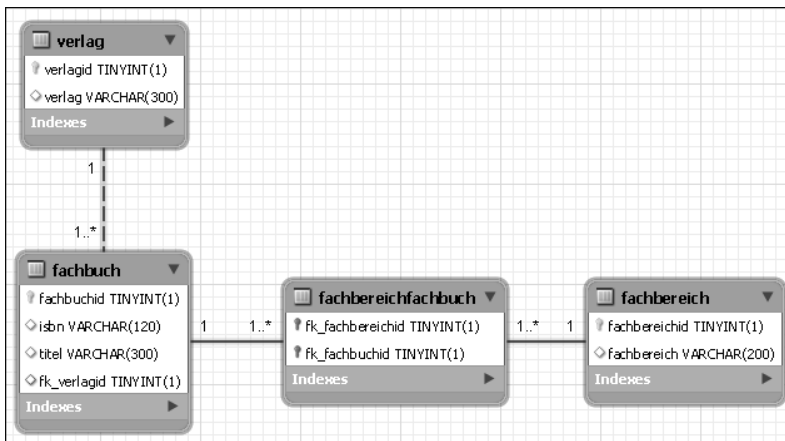


Abbildung 10.13 UML-Modell der Bibliotheksdatenbank

Bevor Sie eine **SELECT**-Anweisung formulieren, in der Sie mit einem **JOIN** die Zeilen von Tabellen verbinden, betrachten Sie zunächst das Modell, um die beteiligten Primärschlüsselspalten und Fremdschlüsselspalten zu identifizieren.

Übung 1

Die Tabellen *fachbuch* und *verlag* stehen in Beziehung zueinander. Die Tabelle *fachbuch* enthält eine Fremdschlüsselspalte *fk_verlagid*, deren Fremdschlüsselwerte auf Primärschlüsselwerte der Spalte *verlagid* in der Tabelle *verlag* referenzieren.

Formulieren Sie eine **SELECT**-Anweisung, in der mit einem **INNER JOIN** die Zeilen der Tabellen *fachbuch* und *verlag* verbunden ausgegeben werden. Hierbei sollen aus der Tabelle *fachbuch* die Spalten *titel*, *isbn* und *fk_verlagid* zurückgeliefert werden. Aus der Tabelle *verlag* sollen die Spalten *verlag* und *verlagid* ausgegeben werden.

Übung 2

In dieser Übung sollen wieder die Zeilen der Tabellen *fachbuch* und *verlag* verbunden ausgegeben werden. In der Ergebnistabelle sollen alle Zeilen enthalten sein, die über Schlüsselwerte verbunden werden können. Außerdem sollen die Zeilen der Tabelle *fachbuch* ausgegeben werden, die über keinen Fremdschlüsselwert zu einem Primärschlüsselwert der Spalte *verlagid* verfügen.

Übung 3

In der letzten Übung haben Sie die Zeilen in einer Ergebnistabelle ausgegeben, die sich über Schlüsselwerte verbinden ließen, sowie die Zeilen der linken Tabelle, die keine Fremdschlüsselwerte enthalten und somit keiner Zeile der rechten Tabelle zugeordnet werden konnten.

In dieser Übung fragen Sie die Zeilen der Tabellen *fachbuch* und *verlag* verbunden ab. In der Ergebnistabelle sollen neben den Zeilen, die sich über Schlüsselwerte verbinden lassen, auch die Zeilen der Tabelle *verlag* enthalten sein, auf die keine Fremdschlüsselwerte aus der Tabelle *fachbuch* verweisen.

Übung 4

In dieser Übung sollen alle Zeilen aus den Tabellen *fachbuch* und *verlag* in einer Ergebnistabelle ausgegeben werden, die sich verbinden lassen oder eben nicht verbinden lassen.

Das heißt also, dass auch die Zeilen der linken Tabellen *fachbuch*, die auf keine Zeilen der rechten Tabelle *verlag* referenzieren, enthalten sein sollen. Oder anders gesagt: Fachbücher, die keinem Verlag zugeordnet sind.

Fehlen dürfen auch nicht die Zeilen aus der rechten Tabelle *verlag*, auf die keine Referenz über Fremdschlüsselwerte aus der linken Tabelle existiert. Das heißt, sämtliche Zeilen der Verlage, denen kein Fachbuch zugeordnet ist, sollen auch enthalten sein.

Falls Sie eine MySQL- oder MariaDB-Datenbank verwenden, müssen Sie beachten, dass der **JOIN**-Typ, der in dieser Übung als Lösung gefordert wird, nicht von einer MySQL- oder MariaDB-Datenbank unterstützt wird. In Abschnitt 10.2.5 habe ich dargestellt, wie Sie alternativ die Funktionalität des geforderten **JOIN**-Typs nachbilden können.

Übung 5

Formulieren Sie eine **SELECT**-Abfrage, in der jede Zeile der Tabelle *fachbuch* mit jeder Zeile der Tabelle *verlag* verbunden ausgegeben wird. Beachten Sie bei dieser Übung, dass hier keine Bedingung erforderlich ist, die auf die Gleichheit von Schlüsselwerten prüft.

Übung 6

Die Zeilen der Tabellen *fachbuch* und *fachbereich* stehen über die Schlüsseltabelle *fachbereichfachbuch* in Beziehung zueinander. In der Tabelle *fachbuch* wird über einen Primärschlüsselwert der Primärschlüsselspalte *fachbuchid* jede Zeile eindeutig identifiziert. Die Tabelle *fachbereich* enthält ebenfalls eine Primärschlüsselspalte *fachbereichid*, mit deren Primärschlüsselwerten jede Zeile eindeutig identifiziert werden kann.

In der Tabelle *fachbereichfachbuch* werden die Primärschlüsselwerte in Beziehung zueinander gesetzt.

Formulieren Sie eine **SELECT**-Abfrage, in der Sie die Zeilen der Tabellen *fachbuch*, *fachbereichfachbuch* und *fachbereich* mit einem **INNER JOIN** verbunden abfragen, um eine Ergebnistabelle zu erhalten. In der Spaltenauswahlliste fordern Sie aus der Tabelle *fachbuch* die Spalte *titel* und aus der Tabelle *fachbereich* die gleichnamige Spalte *fachbereich* an.

Übung 7

Eine Bibliothek für Fachbücher ist natürlich nur sinnvoll, wenn die Fachbücher auch ausgeliehen werden können. In Abbildung 10.14 sehen Sie, dass dem Datenmodell eine weitere Tabelle, *ausleihe*, hinzugefügt wurde.

Der Primärschlüssel der Tabelle setzt sich aus den Spalten *ausleihid* und *exemplarid* zusammen. Der Verleihzeitraum wird durch die Spalten *von* und *bis* abgebildet.

Um für diese Übung ein Beispiel zu erhalten, ist die Modellierung der Tabelle bewusst schlicht gehalten. Die Tabelle *ausleihe* enthält die Spalten *isbn* und *titel*, die auch in

der Tabelle *fachbuch* enthalten sind und somit redundant vorkommen. Die Spalte *fachbuchid* stellt eine Fremdschlüsselspalte dar, die wiederum auf die Primärschlüsselspalte *fachbuchid* der Tabelle *fachbuch* referenziert.

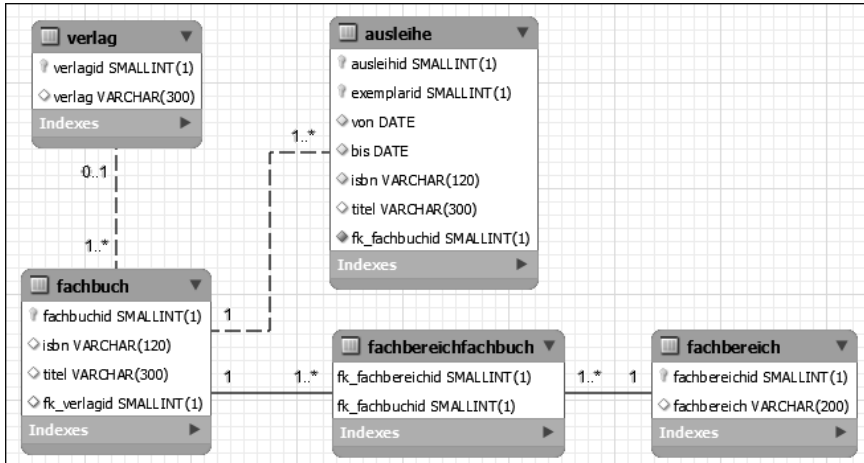


Abbildung 10.14 Das Datenmodell wurde um die Tabelle »ausleihe« erweitert.

Kommen wir jetzt zu der eigentlichen Übung: Die Spaltennamen *isbn*, *titel* und *fachbuchid* sind in beiden Tabellen identisch. Wenn Sie eine Abfrage mit einem **INNER JOIN** formulieren, wird die Datenbank die Spaltennamen nicht eindeutig einer Tabelle zuzuordnen können. Aus diesem Grund soll in dieser Übung ein **INNER JOIN** zwischen den Tabellen *ausleihe* und *fachbuch* formuliert werden, wobei Sie die Spalten der Tabellen jeweils über die Tabellennamen ansprechen.

Nehmen Sie aus der Tabelle *fachbuch* die Spalten *isbn* und *titel* mit in die Spaltenauswahlliste auf. Fragen Sie des Weiteren die gleichnamigen Spalten *isbn* und *titel* aus der Tabelle *ausleihe* über die Spaltenauswahlliste ab. Außerdem sollen die Spalten *von* und *bis* der Tabelle *ausleihe* enthalten sein.

In Listing 10.36 ist die **CREATE TABLE**-Anweisung aufgeführt, die Sie benötigen, um die Tabelle *ausleihe* zu erstellen.

```
CREATE TABLE ausleihe (
  ausleihid SMALLINT, exemplarid SMALLINT,
  von DATE NOT NULL,
  bis DATE NOT NULL,
  isbn VARCHAR(120) NOT NULL,
  titel VARCHAR(300),
  fk_fachbuchid SMALLINT NOT NULL,
  PRIMARY KEY (ausleihid, exemplarid),
  CONSTRAINT fk_fachbuchid_fachbuch
```

```

FOREIGN KEY (fk_fachbuchid)
REFERENCES fachbuch (fachbuchid)
);

```

Listing 10.36 Die Tabelle »ausleihe« mit einer CREATE TABLE-Anweisung erstellen

Um eine Ergebnistabelle zu erhalten, fügen wir zunächst zwei Zeilen in die Tabelle *ausleihe* ein. Listing 10.37 enthält zwei **INSERT**-Anweisungen, mit denen Sie Zeilen in die Tabelle *ausleihe* einfügen können:

```

INSERT INTO ausleihe (ausleihid,exemplarid,von,bis,isbn,titel,fk_fachbuchid)
VALUES (1,1,'2015-04-01','2015-05-01','9878','Elektrotechnik',2);

```

```

INSERT INTO ausleihe (ausleihid,exemplarid,von,bis,isbn,titel,fk_fachbuchid)
VALUES (2,3,'2015-05-10','2015-05-20','9878','Elektrotechnik',2);

```

Listing 10.37 Zeilen in die Tabelle »ausleihe« einfügen

Übung 8

In Übung 7 haben Sie einen **INNER JOIN** über die Tabellen *fachbuch* und *ausleihe* formuliert und die Spalten jeweils über die Tabellennamen referenziert. Verwenden Sie in dieser Übung Tabellenaliasse, um die jeweiligen Spalten der Tabellen *fachbuch* und *ausleihe* anzusprechen.

Übung 9

In Abschnitt 10.2.10 haben Sie erfahren, dass Sie Zeilen aus einer Tabelle mit sich selbst verbunden abfragen können. Betrachten Sie hierzu Tabelle 10.23, in der Fachbereiche in einer hierarchischen Struktur dargestellt werden.

fachbereichid	fachbereich	fk_fachbereich
1	Elektrotechnik	NULL
2	Kommunikationselektronik	1
3	Energieelektronik	1
4	Kaufmann	NULL
5	Industriekaufmann	4
6	Einzelhandelskaufmann	4
7	Mechanik	NULL

Tabelle 10.23 Zeilen, die auf Zeilen in der gleichen Tabelle referenzieren

fachbereichid	fachbereich	fk_fachbereich
8	Industriemechaniker	7
9	Feinmechaniker	7

Tabelle 10.23 Zeilen, die auf Zeilen in der gleichen Tabelle referenzieren (Forts.)

In Listing 10.38 sehen Sie eine **CREATE TABLE**-Anweisung, mit der Sie die Tabelle *fachbereichhierarchie* erstellen:

```
CREATE TABLE fachbereichhierarchie (  
    fachbereichid SMALLINT PRIMARY KEY,  
    fachbereich VARCHAR(200),  
    fk_fachbereich SMALLINT,  
    CONSTRAINT fk_fachbereich_hierarchie  
        FOREIGN KEY (fk_fachbereich)  
        REFERENCES fachbereichhierarchie (fachbereichid)  
);
```

Listing 10.38 Eine Tabelle erstellen, die mit einer Fremdschlüsselspalte auf die eigene Primärschlüsselspalte referenziert

Auch hier müssen Sie zuerst Zeilen einfügen, um sie anschließend mit einem **INNER JOIN** verbunden abfragen zu können. In Listing 10.39 sind die **INSERT**-Anweisungen aufgeführt, mit denen Sie die benötigten Zeilen in die Tabelle *fachbereichhierarchie* einfügen:

```
INSERT INTO fachbereichhierarchie (fachbereichid,fachbereich,fk_fachbereich)  
VALUES (1,'Elektrotechnik',NULL);  
INSERT INTO fachbereichhierarchie (fachbereichid,fachbereich,fk_fachbereich)  
VALUES (2,'Kommunikationselektronik',1);  
INSERT INTO fachbereichhierarchie (fachbereichid,fachbereich,fk_fachbereich)  
VALUES (3,'Energieelektronik',1);  
INSERT INTO fachbereichhierarchie (fachbereichid,fachbereich,fk_fachbereich)  
VALUES (4,'Kaufmann',NULL);  
INSERT INTO fachbereichhierarchie (fachbereichid,fachbereich,fk_fachbereich)  
VALUES (5,'Industrie Kaufmann',4);  
INSERT INTO fachbereichhierarchie (fachbereichid,fachbereich,fk_fachbereich)  
VALUES (6,'Einzelhandelskaufmann',4);  
INSERT INTO fachbereichhierarchie (fachbereichid,fachbereich,fk_fachbereich)  
VALUES (7,'Mechanik',NULL);  
INSERT INTO fachbereichhierarchie (fachbereichid,fachbereich,fk_fachbereich)  
VALUES (8,'Industriemechaniker',7);
```

```
INSERT INTO fachbereichhierarchie (fachbereichid,fachbereich,fk_fachbereich)
VALUES (9,'Feinmechaniker',7);
```

Listing 10.39 Zeilen in die Tabelle »fachbereichhierarchie« einfügen

Wenn Sie die Zeilen in die Tabelle *fachbereichhierarchie* eingefügt haben, formulieren Sie eine **SELECT**-Anweisung, in der Sie mit einem **INNER JOIN** die Zeilen der Tabelle *fachbereichhierarchie* wiederum mit Zeilen der Tabelle *fachbereichhierarchie* verbunden abfragen. Vergessen Sie nicht, die Tabellenaliasse zu verwenden. In die Spaltenauswahlliste nehmen Sie entsprechend pro Tabellenalias jeweils die Spalte *fachbereich* mit auf, um die Hierarchie der Fachbereiche abzubilden. Der Übersicht halber ordnen Sie der Spalte *fachbereich* jeweils einen sprechenden Spaltenalias zu.

Übung 10

In Abschnitt 10.2.11 haben Sie einen **INNER JOIN** kennengelernt, der keine Fremdschlüsselbeziehungen nutzt, um diese in einer **ON**-Klausel auszuwerten. In der Bedingung der **ON**-Klausel haben Sie Spalten aus zwei Tabellen verwendet, denen der gleiche Wertebereich (Datentyp) zugeordnet ist. Nutzen Sie in dieser Übung die Tabelle *ausleihe*, die wir in der Lösung zu Übung 7 erstellt haben, um diese Tabelle mit Zeilen der Tabelle *fachbuch* zu verbinden. Setzen Sie diesmal allerdings nicht die Fremdschlüsselbeziehungen ein. Nutzen Sie in der **ON**-Klausel die Spalte *isbn*, die in beiden Tabellen vorkommt, um eine Bedingung zu formulieren, die auf Gleichheit der Spaltenwerte prüft. Verwenden Sie auch hier Tabellenaliasse, um auf die jeweiligen Spalten der Tabellen zu referenzieren. In der Spaltenauswahlliste wählen Sie die Spalten *isbn* und *titel* der Tabelle *fachbuch* und die Spalten *von* und *bis* der Tabelle *ausleihe*.

Lösung zu Übung 1

```
SELECT titel,isbn,fk_verlagid,verlag,verlagid FROM fachbuch
INNER JOIN verlag
ON fk_verlagid=verlagid;
```

Listing 10.40 Mit einem **INNER JOIN** Zeilen von zwei Tabellen verbunden abfragen**Lösung zu Übung 2**

```
SELECT titel,isbn,fk_verlagid,verlag,verlagid FROM fachbuch
LEFT OUTER JOIN verlag
ON fk_verlagid=verlagid;
```

Listing 10.41 Mit einem **LEFT OUTER JOIN** auch diejenigen Spaltenwerte der linken Tabelle ausgeben, die nicht mit Zeilen der rechten Tabelle verbunden werden konnten

Lösung zu Übung 3

```
SELECT titel,isbn,fk_verlagid,verlag,verlagid FROM fachbuch  
RIGHT OUTER JOIN verlag  
ON fk_verlagid=verlagid;
```

Listing 10.42 Mit einem RIGHT OUTER JOIN auch diejenigen Spaltenwerte der rechten Tabelle ausgeben, auf die keine Zeilen der linken Tabelle referenzieren

Lösung zu Übung 4

```
SELECT titel,isbn,fk_verlagid,verlag,verlagid FROM fachbuch  
FULL OUTER JOIN verlag  
ON fk_verlagid=verlagid;
```

Listing 10.43 Mit einem FULL OUTER JOIN alle Zeilen verbinden, die sich über Schlüsselwerte verbinden lassen, und diejenigen Zeilen der beteiligten Tabellen ausgeben, die sich nicht über Schlüsselwerte verbinden lassen

```
SELECT titel,isbn,fk_verlagid,verlag,verlagid FROM fachbuch  
LEFT OUTER JOIN verlag  
ON fk_verlagid=verlagid  
UNION  
SELECT titel,isbn,fk_verlagid,verlag,verlagid FROM fachbuch  
RIGHT OUTER JOIN verlag  
ON fk_verlagid=verlagid;
```

Listing 10.44 Einen FULL OUTER JOIN unter MySQL und MariaDB nachbilden

Lösung zu Übung 5

```
SELECT titel,isbn,fk_verlagid,verlag,verlagid FROM fachbuch  
CROSS JOIN verlag;
```

Listing 10.45 Alle Zeilen einer Tabelle mit sämtlichen Zeilen einer anderen Tabelle verbinden und umgekehrt

Lösung zu Übung 6

```
SELECT titel,fachbereich FROM fachbuch  
INNER JOIN  
fachbereichfachbuch  
ON fachbuchid=fk_fachbuchid
```



```
INNER JOIN
fachbereich
ON fk_fachbereichid=fachbereichid;
```

Listing 10.46 Einen INNER JOIN über drei Tabellen anwenden

Lösung zu Übung 7

```
SELECT
    ausleihe.isbn,
    ausleihe.titel,
    fachbuch.isbn,
    fachbuch.titel,
    ausleihe.von,
    ausleihe.bis
FROM ausleihe INNER JOIN fachbuch
ON ausleihe.fk_fachbuchid=fachbuch.fachbuchid;
```

Listing 10.47 Die Spalten in einer SELECT-Abfrage mit dem Tabellennamen referenzieren

Lösung zu Übung 8

```
SELECT a.isbn,a.titel,f.isbn,f.titel,a.von,a.bis
FROM ausleihe a INNER JOIN fachbuch f
ON a.fk_fachbuchid=f.fachbuchid;
```

Listing 10.48 Die Spalten in einer SELECT-Abfrage mit Tabellenaliasen referenzieren

Lösung zu Übung 9

```
SELECT
f1.fachbereich AS bereich_uber,
f2.fachbereich AS bereich_unter
from fachbereichhierarchie f1
INNER JOIN fachbereichhierarchie f2
ON f1.fachbereichid=f2.fk_fachbereich;
```

Listing 10.49 Eine Tabelle mit einem INNER JOIN mit sich selbst verbunden abfragen

Lösung zu Übung 10

```
SELECT f.isbn,f.titel,a.von,a.bis
FROM fachbuch f INNER JOIN ausleihe a
ON f.isbn=a.isbn;
```

Listing 10.50 Zeilen von Tabellen über Nicht-Schlüsselwerte verbinden

10.3 Beziehungen (Schlüsselbeziehungen) ändern

Der Status einer Beziehung kann sich ändern. Das gilt auch für die Zeilen einer Kindtabelle, die mit einem Fremdschlüsselwert auf einen Primärschlüsselwert einer Zeile einer Elterntabelle verweisen. Hier gibt es nur eine klare Einschränkung, und zwar, dass nur gültige Beziehungen zwischen Zeilen existieren dürfen. In Abschnitt 9.4 haben Sie die *referenzielle Integrität* kennengelernt, über die eine Datenbank sicherstellt, dass sie keine ungültigen Beziehungen zulässt. Bei der Beziehungsauswahl zwischen den Zeilen einer Kindtabelle und den Zeilen einer Elterntabelle müssen Sie also stets darauf achten, dass sie auch gültig ist.

In Abschnitt 10.3.1 erfahren Sie, wie Sie Beziehungen zwischen Zeilen aus Eltern- und Kindtabellen ändern.

10.3.1 Beziehungen aus Zeilen aus einer Kindtabelle ändern

Beziehungen zu ändern, ist sehr einfach. Sie ändern einen Fremdschlüsselwert einer Kindtabelle in einen anderen Primärschlüsselwert einer Elterntabelle. Die Zeile in der Kindtabelle referenziert dann auf die andere Zeile der Elterntabelle.

Sehen wir uns hierzu ein Beispiel an. In Abbildung 10.15 sehen Sie die Tabellen *auszubildender* und *adresse*.

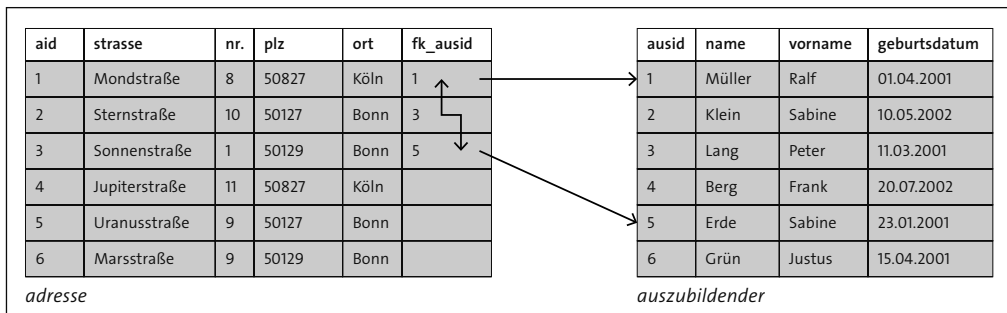


Abbildung 10.15 Beziehungen zwischen Zeilen in Kind- und Elterntabellen ändern

Die Zeilen der Tabellen stehen über eine Fremdschlüsselspalte in der Tabelle *adresse* in Beziehung zueinander. Auf die Auszubildende Sabine Erde mit dem Primärschlüsselwert 5 (der Spalte *ausid*) referenziert der Fremdschlüsselwert 5 der Spalte *fk_ausid* der Tabelle *adresse*. Dem Auszubildenden Ralf Müller mit dem Primärschlüsselwert 1 (der Spalte *ausid*) wird ebenfalls über einen Fremdschlüsselwert 1 aus der Tabelle *adresse* eine Adresse zugeordnet. Die beiden Auszubildenden tauschen nun die Wohnungen. Die Beziehungen der Adressen zu den Auszubildenden sind somit nicht mehr richtig. Die alten Beziehungen müssen also verändert werden, um die aktuellen Adressen den Auszubildenden zuordnen zu können.

Index

_ (Platzhalter)	71
% (Platzhalter)	71
.....	472
1:1-Beziehung	289
1:n-Beziehung	291

A

Abfrage	
<i>korrelierende</i>	554
<i>nicht korrelierende</i>	560
<i>Spalte</i>	47
Abhängigkeit	
<i>transitive</i>	317
<i>voll funktionale</i>	314
ACID-Prinzip	413
Administratorberechtigung	36
Aggregationsoperation	530
Algebra	
<i>boolesche</i>	91
Alias	127
ALL	232, 239, 566, 567
AND (Logik)	91
AND NOT (Logik)	98
Änderungsanomalie	310
Anforderungskatalog	279
ANSI-SQL-Standard	25
ANY	566, 567
AS	128
ASC	114
Asterisk	49
Atomar	312
Attribut	281
<i>atomares</i>	312
<i>optionales</i>	286
<i>Pflichtattribut</i>	286
<i>Wertebereich</i>	284
Ausdruck	452
<i>auswerten</i>	452
<i>evaluieren</i>	452
Ausfallsicherheit	411
Autoinkrement-Schlüssel	194
AVG	537

B

Basic Latin	173
Bedingungen verknüpfen	
<i>AND NOT-Operator</i>	99
<i>AND-Operator</i>	92
<i>Kombinationen</i>	101
<i>OR NOT-Operator</i>	100
<i>OR-Operator</i>	96
Bedingungslogik	487
BEGIN	416
Benutzer	
<i>anlegen</i>	260
<i>entfernen</i>	261
Berechtigungen	
<i>entziehen (REVOKE)</i>	270
<i>Rollen</i>	270, 271
<i>vergeben (GRANT)</i>	268
<i>verwalten</i>	268
BETWEEN	67
Beziehung	388
<i>1:1</i>	326
<i>1:n</i>	328
<i>aufheben</i>	399
<i>identifizierende</i>	301
<i>m:n</i>	330
<i>nicht identifizierende</i>	302
<i>Verhältnismäßigkeit</i>	289
<i>zwischen Objekten</i>	280
Binärsystem	166
Boolesche Algebra	91
Boyce-Codd-Normalform → BCNF	
B-Tree-Index	600

C

Cache	605
CASE	487
case-insensitive	49
CAST	179
CHAR	175
CHAR_LENGTH	477
CHARINDEX	478
CHECK	207, 221
Codd, Edgar F.	23
Collation	112
COMMIT	416

CONSTRAINT	199, 215, 327
COUNT	530, 531
<i>DISTINCT</i>	533
CREATE DATABASE	323
CREATE INDEX	606
CREATE TABLE	165, 187, 188
CREATE VIEW	576
CROSS JOIN	366
CRUD	22
CURRENT_DATE	496
CURRENT_TIME	496
CURRENT_TIMESTAMP	496

D

DATE_ADD	522
DATE_FORMAT	500
DATE_SUB	522
DATEADD (MS SQL Server)	524
Datenbank	
<i>anlegen</i>	323
<i>anlegen, UTF-8</i>	324
<i>modellieren</i>	279
Datenbankabfragesprache	21
Datenbankbenutzer	
<i>Administrator</i>	36
Datenbankmodell	279
Datenbanksystem	24
Datenmodell	323
Datensätze	22
Datentyp	
<i>CHAR</i>	175
<i>DATE</i>	496
<i>Datum und Zeit</i>	172
<i>DECIMAL</i>	171
<i>Fixkomma</i>	171
<i>Fließkomma</i>	171
<i>floating point</i>	171
<i>ganze Zahlen</i>	169
<i>INTEGER</i>	170
<i>konvertieren</i>	179
<i>NUMERIC</i>	171
<i>rationale Zahlen</i>	171
<i>SIGNED</i>	170
<i>TINYINT</i>	165
<i>Überblick</i>	165
<i>umwandeln</i>	179
<i>UNSIGNED</i>	170
<i>VARCHAR</i>	175
<i>Zeichenketten</i>	173
DATEPART	517

Datum	172
<i>Berichtsdatum</i>	498
<i>einfügen</i>	497
<i>formatieren</i>	499
Datumsangabe	
<i>berechnen</i>	520
<i>berechnen (MS SQL Server)</i>	524
<i>berechnen (MySQL und MariaDB)</i>	521
<i>berechnen (PostgreSQL)</i>	523
<i>extrahieren</i>	515
<i>formatieren (MS SQL Server)</i>	510
<i>formatieren (MySQL und MariaDB)</i>	500
<i>formatieren (PostgreSQL)</i>	505
Datumsformat UTC	495
Datumswert	60
DBCC DROPCLEANBUFFERS	605
DECIMAL	171
DEFAULT	205
DEFAULT CHARACTER	324
DELETE	156, 268
<i>View</i>	590
Denormalisierung	318
DESC	114
DETERMINISTIC	619
Differenzmenge	229, 235, 250
Disjunktion	94
Disjunktionsfalle	104
DISTINCT	131, 132, 237, 533
Domain	284
Drei-Werte-Logik	81
DROP INDEX	609
DROP TABLE	191
DROP VIEW	596
Dualsystem → Binärsystem	

E

Einfügeanomalie	310
Einschränkungen	
<i>Spalten</i>	199
<i>Tabelle</i>	214
ELSE	488, 492
END	489
Entität	
<i>1:1-Beziehung</i>	289
<i>1:n-Beziehung</i>	291
<i>Attribute</i>	281
<i>Beziehungen</i>	288
<i>Beziehungen zwischen Entitäten</i>	287
<i>Definition</i>	280
<i>Fremdschlüssel</i>	290

Entität (Forts.)	
<i>identifizieren</i>	280
<i>in Beziehung</i>	288
<i>Kardinalität</i>	289
<i>m:n-Beziehung</i>	293
<i>Pflichtattribut</i>	286
<i>Schlüsselattribut</i>	281
<i>Schlüsselattribute</i>	282
<i>schwache</i>	298
<i>Selbstverweis</i>	296
<i>starke</i>	298
Entity-Relationship-Diagrammeditor	302
Entity-Relationship-Modell	280
<i>Krähenfußnotation</i>	291
Entity-Relationship-Notation	280
Ergebnis	
<i>filtern</i>	457
<i>sortieren</i>	458
Ergebniszeilen einschränken	109
EXCEPT	235, 250
Existenzprüfung	571
EXISTS	571
Explizite Typkonvertierung	180
Expression	451
EXTRACT	515

F

FETCH	109, 110
FOR	469
FOREIGN KEY	327
Foreign Key	290
FORMAT	510
Fremdschlüsseldefinition	341
Fremdschlüsselspalte	327
FROM	48, 469
FULL OUTER JOIN	360, 362
Funktion	463
<i>CONCAT</i>	464
<i>LOWER</i>	464
<i>SUBSTRING</i>	464
<i>Textwerte</i>	464
<i>TRIM</i>	464
<i>UPPER</i>	464
<i>verschachtelte</i>	482

G

GRANT	268, 271
GROUP BY	529, 531
<i>AVG</i>	537

COUNT	531
HAVING	545
MAX	539
MIN	540
<i>nach zwei oder mehr Spalten</i>	
<i>gruppieren</i>	548
NULL-Werte	541
SUM	535

H

HAVING	545
HeidiSQL	263
Hochkommata	55

I

Identifying	301
Identifying Relationship	330
Implizite Typkonvertierung	181
IN	76
Index	599
<i>B-Tree-Index</i>	600
<i>CREATE INDEX</i>	606
<i>DBCC DROPCLEANBUFFERS</i>	605
<i>DROP INDEX</i>	609
<i>eindeutiger</i>	606
<i>erstellen</i>	602
<i>Fremdschlüsselspalten indexieren</i>	609
<i>löschen</i>	609
<i>MS SQL Server</i>	605
<i>MySQL</i>	605
<i>Spalten</i>	606
<i>SQL_NO_CACHE</i>	605
<i>Tabelle</i>	604
<i>UNIQUE</i>	606
information_schema	140
INNER JOIN	351, 352, 558
<i>mehrere Tabellen</i>	368
<i>Schlüsselvergleich</i>	377
INSERT	135, 268
<i>View</i>	584
Installation	28
International Organization for Standardization → ISO	
INTERSECT	233, 245
INTERVAL	523
INTO	136
IS NOT NULL	83
IS NULL	83
ISO	24, 25

ISO 8601	57, 60, 172, 496
ISO/IEC 9075-1	25

J

JOIN

<i>FULL OUTER JOIN</i>	351
<i>gleichnamige Spalten</i>	370
<i>INNER JOIN</i>	351
<i>LEFT OUTER JOIN</i>	351
<i>ohne Schlüsselvergleich verbinden</i>	377
<i>ON</i>	353
<i>RIGHT OUTER JOIN</i>	351
<i>SELF JOIN</i>	375
<i>Tabellenaliasse</i>	373
Joker	71

K

Kardinalität	289, 301
Kommentar	42
Konjunktion	91
Konsistenz	412
Konstante	
<i>abfragen</i>	125
<i>Übersicht</i>	124
Konvertierungsfunktionen	179
Korrelation	554
Krähenfußnotation	291
Künstlicher Schlüssel → Schlüsselattribut	

L

Latin 1 Supplement	173
Leerzeichen filtern	466
LEFT OUTER JOIN	351, 354
LIKE	71
LIMIT	109
Logische Verknüpfung	91
Löschanomalie	310
LOWER	465

M

m:n-Beziehung	293
MariaDB	25
Mathematischer Operator	451
<i>Ergebnis filtern</i>	457
<i>Ergebnis sortieren</i>	458
<i>Potenz</i>	479
<i>Quadratwurzel</i>	480

MAX	539
Mehrwertigkeit	313
Menge	

<i>Differenzmenge</i>	229
<i>Schnittmenge</i>	229
<i>Vereinigungsmenge</i>	229
Mengenlehre	23
Mengenoperation	229, 236
<i>ORDER BY</i>	255
<i>WHERE</i>	253

Microsoft SQL Server Management

Studio	302
--------------	-----

Microsoft SQL Server → MS SQL Server

MIN	540
Modulo-Operator	451, 456
MS SQL Server	24

MySQL

<i>Dokumentation</i>	25
<i>FULL OUTER JOIN</i>	362

MySQL-Datenbank

<i>installieren</i>	28
<i>unterstützte Betriebssysteme</i>	25

N

Non-identifying	302
Normalform	309
<i>dritte</i>	316
<i>erste</i>	312
<i>zweite</i>	314
Normalisierung	309, 311
NOT	64
NOT BETWEEN	67
NOT EXISTS	573
NOT IN	77
NOT LIKE	76
NOT NULL	199, 200
NULL	79, 200
NULL-Wert	79, 82
<i>Gruppierung</i>	541
NUMERIC	171

O

ON	353
ON DELETE CASCADE	392
ON UPDATE CASCADE	337, 390
Operator	165
-	451
*	451
/	451

Operator (Forts.)	
%	451
+	451
Optimierung (Datenmodelle)	309
OR (Logik)	91, 94
OR NOT (Logik)	99
ORDER BY	113, 255, 490
View	581

P

Pattern	500
Performance	599
Pflichtattribut	286
Pflichtfeld	199
pgAdmin IV Client	302
Pipe-Operator	473
Plausibilitätsprüfung	207
POSITION	478
Potenz	479
POWER	479
Primärschlüssel	136, 192
hochzählen	194
Kombination	216
Tabellenebene	214
Projektion	47
Prozedur	
anlegen	632
ausführen	631

Q

Quadratwurzel	480
---------------	-----

R

Redundanz	309
Referenzielle Integrität	301, 334, 412
Relationale Datenbank	413
Relationales Datenbankmodell	24
Reservierte Schlüsselwörter	197
Restwert	456
Restwertoperator	451
REVOKE	272
RIGHT OUTER JOIN	351, 357
ROLLBACK	416
Rollen	270
anlegen (CREATE ROLE)	270
Berechtigung	271
Berechtigung entziehen	272
entfernen	272

zuordnen	271
root (Benutzer)	33, 36
Rückgabewert	618

S

Schema	140
Schlüsselattribut	281–283
Schlüsselbeziehung	
aufheben	399
erstellen	388
Schlüsselkandidat	283
Schlüsselvergleich	377
Schlüsselwert	152
Schnittmenge	229, 233, 245
SELECT	47, 48, 268, 488
einschränken	109
SELECT-Abfrage	22
Selektion	52
SELF JOIN	375
SET	146, 324
Sicht → View	
Skalarabfrage	562
Skalarfunktion	463
CHAR_LENGTH	477
für Zahlenwerte	476
LOWER	465
POSITION	477
POWER	477
Spaltenwerte verketteten	472
SQRT	477
SUBSTRING	469
TRIM	466
UPPER	465
Skalarfunktionen	
entfernen	627
entwickeln	617
Parameter	623
Rückgabewert	618
SOME	566, 567
Sonderzeichen	112
Sortieren	
ASC	114
DESC	114
mehrere Spalten	115
nach Datumswerten	117
nach numerischen Spaltenwerten	116
nicht definierte Werte	118
Richtung	114
Sortierreihenfolge	490

Spalte	47	AND NOT	98
alle Spalten abfragen	49	ANY	567
alle Werte ändern	150	AS	128
auf NULL prüfen	82	ASC	114
CHECK	207	BETWEEN	67
CONSTRAINTS	199	CASE	487–489
einem Alias zuordnen	128	CAST	179
entfernen	438	CREATE DATABASE	323, 324
fehlende Spaltenwerte	79	CREATE INDEX	606
Gleichheit	55	CREATE TABLE	165, 187
hinzufügen	438	CREATE VIEW	576
Intervall	66	CROSS JOIN	366
LIKE	71	DEFAULT CHARACTER	324
mehrere sortieren	115	DELETE	156
mehrere Spalten gruppieren	548	DESC	114
mehrere Werte einer Zeile ändern	147	DISTINCT	131, 132, 237
mehrere Zeilen gleichzeitig ändern	148	DROP INDEX	609
Mengenzugehörigkeit prüfen	76	DROP TABLE	191
mit Alias versehen	128	DROP VIEW	596
mit expliziter Angabe einfügen	136	ELSE	488
nach Datumswerten sortieren	117	END	489
nach numerischen Werten sortieren	116	EXCEPT	250
NOT NULL	199	EXISTS	571
NOT-Operator	64	FETCH	110
NULL-Zuweisung	151	FROM	48
ohne Angabe einfügen	140	FULL OUTER JOIN	360
Schlüsselwert ändern	152	GROUP BY	531
Spaltenauswahl abfragen	47	IN	76, 554, 561
Standardwert	205	INNER JOIN	352
umbenennen	425	INSERT	135
Ungleichheit	58	INTERSECT	233, 245
UNIQUE	203	INTO	136
Unterschiede	60, 62	IS NOT NULL	83
Werte ändern	146	IS NULL	83
Werte vergleichen	84	kommentieren	42
Spaltenwert		LEFT OUTER JOIN	354
addieren	452	LIKE	71
dividieren	455	LIMIT	109
ermitteln	23	NOT	64
gruppieren	529	NOT BETWEEN	67
multiplizieren	455	NOT EXISTS	573
Restwert berechnen	456	NOT IN	77
subtrahieren	454	NOT LIKE	76
SQL	21	ON DELETE CASCADE	392
Geschichte	23	ON UPDATE CASCADE	390
Grundlagen	21	OR	91
SQL_NO_CACHE	605	OR NOT	99
SQL-Anweisung		ORDER BY	113
ALL	232, 239, 567	RENAME TABLE	424
ALTER TABLE	423	RIGHT OUTER JOIN	357
AND	91	SELECT	47, 48

SQL-Anweisung (Forts.)		Elterntabellen	341
SELECT ... FROM	47	erstellen	187
SELF JOIN	375	filtern	51
SET	146, 324	Kindtabellen	341
SOME	567	löschen	191
THEN	488	m:n-Beziehung	330
TOP	110	Primärschlüssel erstellen	192
TRUNCATE	161	reservierte Schlüsselwörter	197
UNION	230	Selbstbezug	33
UNIQUE	203	Spalte	22
UPDATE	145	Spalten	47
VALUES	137	Spalten abfragen	48
WHEN	488	Struktur	22
WHERE	51, 52	Struktur verändern	423
SQL-Dialekt	26	umbenennen	423
SQL-Standard, ISO/IEC 9075-1	25	UNIQUE	218
SQL-Vergleichsoperator		virtuelle	575
BETWEEN	66	Zeile	22
Gleichheit	55	Zielstruktur	187
größer/gleich	62	Tabelle erstellen	187
kleiner/gleich	60	Autoinkrement-Schlüssel	194
LIKE	71	CHECK	207, 221
Mengenzugehörigkeit	76	DEFAULT	205
NOT	64	Einschränkung	199, 200, 203, 205, 207, 214, 218, 221
NULL-Value	79, 82	NOT NULL	199
Spaltenwerte vergleichen	84	NULL	200
Übersicht	52	PRIMARY KEY	189, 192
Ungleichheit	58	UNIQUE	203, 218
SQRT	480	Tabellenalias, JOIN	373
Standardwert	205	THEN	488, 492
String	173	TO_CHAR	505
Structured Query Language → SQL		TOP	109, 110
Subquery	553	Transaktion	411
korrelierende	553	ACID-Prinzip	413
korrelierende Abfragen	554	atomare	412
nicht korrelierende	560	Atomicity	413
SUBSTRING	469	BEGIN	416
SUM	535	COMMIT	416
Surrogatschlüssel → Schlüsselattribut		Consistency	413
		dauerhafte	413
		Durability	413
		Isolation	413
		isoliert ausführen	413, 419
		Konsistenz	412
		read committed	420
		read uncommitted	420
		repeatable read	420
		ROLLBACK	416
		serializable	420
		Snapshot	421

T	
<hr/>	
Tabelle	
1:1-Beziehung	326
1:n-Beziehung	328
Beziehung	341
CHECK	221
Definition	21, 23
Eigenschaften	23
Einschränkungen	214
Eltern-Kind-Beziehung	341

Auf einen Blick

1	Grundlagen kennenlernen und verstehen	21
2	Los geht's: Die Grundfunktionen der Tabellenabfrage (SELECT)	47
3	Zeilen einfügen (INSERT), ändern (UPDATE) und löschen (DELETE, TRUNCATE)	135
4	Tabellen anlegen (CREATE TABLE)	165
5	Mengenoperationen anwenden	229
6	Benutzer, Rollen und ihre Berechtigungen	259
7	Datenbanken modellieren	279
8	Datenmodelle optimieren (Normalisierung)	309
9	Datenmodelle in Tabellen überführen	323
10	Operationen auf Tabellen in Beziehungen anwenden	341
11	Transaktionen	411
12	Tabellenstrukturen verändern	423
13	Mit SQL rechnen	451
14	Skalarfunktionen anwenden	463
15	Bedingungslogik	487
16	Mit Zeit und Datum arbeiten	495
17	Spaltenwerte gruppieren (GROUP BY)	529
18	Mächtiges Werkzeug: Die Unterabfragen (Subqueries)	553
19	Views: Abfragen in virtuellen Tabellen speichern	575
20	Abfragen mit einem Index optimieren	599
21	Skalarfunktionen entwickeln	617
22	Prozeduren entwickeln	631

Inhalt

Materialien zum Buch	19
1 Grundlagen kennenlernen und verstehen	21
1.1 Die Tabelle als zentrales Element	21
1.2 Eine kleine Historie von SQL	23
1.3 Datenbanksysteme	24
1.4 SQL – ein Standard und seine Umsetzung	25
1.5 Zu diesem Buch	27
1.6 MySQL unter Windows installieren	28
1.7 Die MySQL-Übungsdatenbank anlegen	36
1.8 Eine erste Abfrage an die Datenbank senden	39
1.9 Kommentarfunktion	42
1.9.1 Übungen	44
2 Los geht's: Die Grundfunktionen der Tabellenabfrage (SELECT)	47
2.1 Mit einer SELECT-Anweisung Tabellen abfragen	47
2.1.1 Die Tabelle »mitarbeiter«	47
2.1.2 Wie frage ich eine Tabelle ab? (SELECT ... FROM)	47
2.1.3 Spalten einer Tabelle abfragen	48
2.1.4 Alle Spalten einer Tabelle abfragen	49
2.1.5 Übungen	50
2.2 Zeilen in einer Abfrage mit WHERE filtern	51
2.2.1 SQL-Vergleichsoperatoren	52
2.2.2 Spaltenwerte auf Gleichheit prüfen	55
2.2.3 Spaltenwerte auf Ungleichheit prüfen	58
2.2.4 Spaltenwerte auf kleiner als/gleich prüfen	60
2.2.5 Spaltenwerte auf größer als/gleich prüfen	62
2.2.6 Bedingungen mit dem NOT-Operator verneinen	64
2.2.7 Spaltenwerte auf ein Intervall prüfen (BETWEEN)	66

2.2.8	Spaltenwerte auf ein Muster prüfen (LIKE)	71
2.2.9	Spaltenwerte auf Mengenzugehörigkeit prüfen	76
2.2.10	Fehlende Spaltenwerte (NULL-Value)	79
2.2.11	Spaltenwerte auf NULL prüfen	82
2.2.12	Spaltenwerte auf »ist nicht NULL« prüfen	83
2.2.13	Spaltenwerte mit Spaltenwerten vergleichen	84
2.2.14	Übungen	86
2.3	Filterbedingungen mit AND (NOT) und OR (NOT) logisch verknüpfen	91
2.3.1	Der logische Verknüpfungsoperator AND	91
2.3.2	SQL-Bedingungen mit dem logischen AND-Operator verknüpfen	92
2.3.3	Der logische Verknüpfungsoperator OR	94
2.3.4	SQL-Bedingungen mit dem logischen OR-Operator verknüpfen	96
2.3.5	Der logische Verknüpfungsoperator AND NOT	98
2.3.6	SQL-Bedingungen mit dem AND NOT-Operator logisch verknüpfen	99
2.3.7	Der logische Verknüpfungsoperator OR NOT	99
2.3.8	SQL-Bedingungen mit dem logischen OR NOT-Operator verknüpfen	100
2.3.9	Logische Verknüpfungsoperatoren kombiniert anwenden	101
2.3.10	Übungen	106
2.4	Ergebniszeilen einer SELECT-Anweisung einschränken	109
2.4.1	Ergebniszeilen mit FETCH, LIMIT und TOP eingrenzen	109
2.4.2	Übungen	111
2.5	Datensätze sortiert abfragen	111
2.5.1	Aufsteigende Sortierung gemäß einer Spaltenangabe	113
2.5.2	Auf- und absteigende Sortierung mehrerer Spalten	115
2.5.3	Nach numerischen Spaltenwerten sortieren	116
2.5.4	Nach Datumswerten sortieren	117
2.5.5	Nicht definierte Werte in einer Sortierung beachten	118
2.5.6	ORDER BY mit einer WHERE-Klausel verwenden	119
2.5.7	Übungen	121
2.6	Konstanten in die Spaltenauswahlliste aufnehmen	124
2.6.1	Abfrage eines konstanten Textes	125
2.6.2	Konstanten und Spalten einer Tabelle gleichzeitig abfragen	125
2.6.3	Übungen	126
2.7	Spalten einen Alias zuordnen	127
2.7.1	Spalten in einer Abfrage mit einem Alias versehen	128
2.7.2	Ausgewählten Spalten einer Abfrage einen Alias zuordnen	128
2.7.3	Spalten und Konstanten einen Alias zuordnen	129
2.7.4	Übungen	130

2.8	Gleiche Ergebniszeilen ausschließen (DISTINCT)	131
2.8.1	Übungen	133

3 Zeilen einfügen (INSERT), ändern (UPDATE) und löschen (DELETE, TRUNCATE) 135

3.1	Zeilen mit einer INSERT-Anweisung einfügen	135
3.1.1	Spaltenwerte mit expliziter Spaltenangabe einfügen	136
3.1.2	Spaltenwerte ohne Spaltenangabe einfügen	140
3.1.3	Übungen	143
3.2	Zeilen mit einer UPDATE-Anweisung ändern	145
3.2.1	Einen Spaltenwert einer Zeile ändern	146
3.2.2	Mehrere Spaltenwerte einer Zeile gleichzeitig ändern	147
3.2.3	Spaltenwerte einer Spalte für mehrere Zeilen gleichzeitig ändern	148
3.2.4	Allen Spaltenwerten einer Spalte einen Wert zuordnen	150
3.2.5	Spaltenwerten mit einer UPDATE-Anweisung einen NULL-Wert zuweisen	151
3.2.6	Schlüsselwertspalten mit UPDATE einen neuen Wert zuweisen	152
3.2.7	Übungen	154
3.3	Zeilen mit einer DELETE-Anweisung löschen	156
3.3.1	Eine Zeile einer Tabelle löschen	157
3.3.2	Mehrere Zeilen einer Tabelle gleichzeitig löschen	158
3.3.3	Alle Zeilen einer Tabelle gleichzeitig löschen	158
3.3.4	Übungen	159
3.4	Alle Zeilen einer Tabelle mit einer TRUNCATE-Anweisung löschen	161
3.4.1	Die TRUNCATE-Anweisung anwenden	161
3.4.2	Übungen zum Thema »Alle Zeilen einer Tabelle mit einer TRUNCATE-Anweisung löschen«	162

4 Tabellen anlegen (CREATE TABLE) 165

4.1	Datentypen	165
4.1.1	Datentypen für ganze Zahlen	169
4.1.2	Datentypen für rationale Zahlen	171
4.1.3	Datentypen für Datum und Zeit	172
4.1.4	Datentypen für Zeichenketten	173
4.1.5	Übungen	176

4.2	Datentypen umwandeln	179
4.3	Explizite und implizite Typkonvertierung	180
4.3.1	Explizite Typkonvertierung	180
4.3.2	Implizite Typkonvertierung	181
4.3.3	Übungen	186
4.4	Einfache Tabellen mit CREATE TABLE erstellen	187
4.4.1	Zielstruktur der Tabelle	187
4.4.2	Tabellen mit der CREATE TABLE-Anweisung anlegen	188
4.4.3	Tabellen mit einer DROP-Anweisung löschen	191
4.4.4	Eine Tabelle mit einem Primärschlüssel ausstatten	192
4.4.5	Automatisch hochzählende numerische Primärschlüsselspalten festlegen	194
4.4.6	Reservierte Schlüsselwörter	197
4.4.7	Übungen	198
4.5	Spalten Einschränkungen (CONSTRAINTS) zuordnen	199
4.5.1	Spalten als Pflichtfelder (NOT NULL) definieren	199
4.5.2	Spalten mit einer UNIQUE-Einschränkung versehen	203
4.5.3	Standardwerte mit DEFAULT für Spalten festlegen	205
4.5.4	Bedingungen mit einer CHECK-Einschränkung für Spalten festlegen	207
4.5.5	Übungen	210
4.6	Spalten auf Tabellenebene Einschränkungen (CONSTRAINT) zuordnen	214
4.6.1	Einen Primärschlüssel auf Tabellenebene festlegen	214
4.6.2	Eine UNIQUE-Einschränkung auf Tabellenebene festlegen	218
4.6.3	Eine CHECK-Einschränkung auf Tabellenebene festlegen	221
4.6.4	Übungen	223

5 Mengenoperationen anwenden **229**

5.1	Mengenoperationen auf Ergebnistabellen anwenden	229
5.1.1	Eine Vereinigungsmenge aus zwei Mengen bilden	229
5.1.2	Eine Schnittmenge bilden	233
5.1.3	Eine Differenzmenge bilden	235
5.2	Eine Vereinigungsmenge bilden (UNION)	236
5.2.1	Vereinigungsmengen von Ergebnistabellen bilden	237
5.2.2	Übungen	243

5.3	Die Schnittmenge von Ergebnistabellen bilden (INTERSECT)	245
5.3.1	Schnittmengen von Ergebnistabellen	245
5.3.2	Übungen	247
5.4	Eine Differenzmenge aus Ergebnistabellen bilden (EXCEPT)	250
5.4.1	Differenzmenge von Ergebnismengen bilden	250
5.4.2	Übungen	251
5.5	Mengenoperationen in Kombination mit einer WHERE-Klausel verwenden	253
5.5.1	Vor einer Vereinigungsoperation mit UNION filtern	253
5.5.2	Übung	254
5.6	Vereinigungsmengen in Kombination mit einer ORDER BY-Klausel	255
5.6.1	Übungen	256
6	Benutzer, Rollen und ihre Berechtigungen	259
6.1	Benutzer anlegen (CREATE USER)	260
6.1.1	Nutzer in einer MySQL- oder MariaDB-Datenbank anlegen	260
6.1.2	Nutzer in einer PostgreSQL-Datenbank anlegen	260
6.1.3	Nutzer in einer MS SQL Server-Datenbank anlegen	261
6.2	Benutzer entfernen	261
6.3	Eine Verbindung für einen Datenbankbenutzer erstellen	262
6.3.1	Verbindung für eine MySQL-Datenbank einrichten	262
6.3.2	Verbindung für eine MariaDB-Datenbank herstellen	263
6.3.3	Verbindung für eine PostgreSQL-Datenbank herstellen	264
6.3.4	Verbindung für eine MS SQL Server-Datenbank herstellen	267
6.4	Berechtigungen verwalten	268
6.4.1	Berechtigungen vergeben (GRANT)	268
6.4.2	Berechtigungen entziehen (REVOKE)	270
6.5	Mit Rollen Berechtigungen zuordnen	270
6.5.1	Rollen anlegen (CREATE ROLE)	270
6.5.2	Rollen mit Berechtigungen ausstatten	271
6.5.3	Rollen Datenbanknutzern zuordnen	271
6.5.4	Rollen Berechtigungen entziehen	272
6.5.5	Rollen entfernen	272
6.6	Übungen	273

7 Datenbanken modellieren 279

7.1 Anforderungskatalog	279
7.2 Entitäten identifizieren und modellhaft abbilden	280
7.2.1 Entitäten identifizieren	280
7.2.2 Informationen zu den Entitäten ermitteln	281
7.2.3 Schlüsselattribute für Entitäten identifizieren	281
7.2.4 Die Wertebereiche von Attributen erkennen	284
7.2.5 Zwischen Pflichtattributen und optionalen Attributen unterscheiden	286
7.3 Beziehungen zwischen Entitäten festlegen	287
7.3.1 Beziehungen im Entity-Relationship-Modell definieren	288
7.3.2 Kardinalitäten von Beziehungen erkennen	289
7.3.3 Eine besondere 1:n-Beziehung – oder Entitäten, die auf sich selbst verweisen	296
7.3.4 Starke und schwache Entitäten unterscheiden	298
7.3.5 Die Beziehungstypen »identifying« und »non-identifying«	301
7.4 Datenmodelle in der UML-Notation darstellen	302
7.5 Übungen	305

8 Datenmodelle optimieren (Normalisierung) 309

8.1 Redundanzen erkennen	309
8.1.1 Was ist eine Redundanz?	309
8.1.2 Was bedeutet Normalisierung?	311
8.2 Die 1. Normalform anwenden	312
8.3 Die 2. Normalform anwenden	314
8.4 Die 3. Normalform anwenden	316
8.5 Denormalisierung	318
8.6 Übungen	320

9 Datenmodelle in Tabellen überführen 323

9.1 Die Ausbildungsdatenbank anlegen	323
9.1.1 Eine neue Datenbank mit UTF-8-Zeichensatz anlegen (MySQL, MariaDB)	324
9.1.2 Eine neue Datenbank mit UTF-8-Zeichensatz anlegen (PostgreSQL)	324
9.1.3 Eine neue Datenbank mit Unicode-Zeichensatz anlegen (MS SQL Server)	324
9.1.4 Übung	325
9.2 Tabellen mit Beziehungen zu anderen Tabellen erstellen	325
9.2.1 Die Ausbildungsdatenbank im Modell erfassen	325
9.2.2 Tabellen erstellen, die in einer 1:1-Beziehung stehen	326
9.2.3 Tabellen erstellen, die in einer 1:n-Beziehung stehen	328
9.2.4 Tabellen erstellen, die in einer m:n-Beziehung stehen	330
9.2.5 Tabellen erstellen, die zu sich selbst in Beziehung stehen	331
9.3 Übung	332
9.4 Die referenzielle Integrität verstehen	334

10 Operationen auf Tabellen in Beziehungen anwenden 341

10.1 Zeilen in Tabellen einfügen, die in Beziehung zueinander stehen	341
10.1.1 Zeilen in die Tabelle »auszubildender« einfügen	341
10.1.2 Zeilen in die Tabelle »ausbildungsberuf« einfügen	342
10.1.3 Zeilen in die Tabelle »lehrfach« einfügen	342
10.1.4 Zeilen in die Tabelle »adresse« (inklusive der Beziehungen) einfügen	343
10.1.5 Zeilen in die Tabelle »ausbildungsvertrag« (inklusive der Beziehungen) einfügen	344
10.1.6 Zeilen in die Tabelle »beruflehrfach« (inklusive der Beziehungen) einfügen	344
10.1.7 Zeilen in die Tabelle »mitarbeiterausbildungsbetrieb« (inklusive der Beziehungen) einfügen	345
10.1.8 Übungen	347

10.2	Zeilen aus Tabellen, die in Beziehung stehen, mit JOIN verbunden abfragen	351
10.2.1	Zeilen mit einem INNER JOIN verbinden	352
10.2.2	Zeilen mit einem LEFT OUTER JOIN verbinden	354
10.2.3	Zeilen mit einem RIGHT OUTER JOIN verbinden	357
10.2.4	Zeilen mit einem FULL OUTER JOIN verbinden	360
10.2.5	Einen FULL OUTER JOIN unter MySQL oder MariaDB nachbilden	362
10.2.6	Zeilen mit einem CROSS JOIN verbinden	366
10.2.7	Zeilen von drei Tabellen mit einem INNER JOIN verbinden	368
10.2.8	Spalten in einem JOIN über Tabellennamen referenzieren	370
10.2.9	Spalten in einem JOIN über Tabellenaliasse referenzieren	373
10.2.10	Zeilen mit einem Self Join verbinden	375
10.2.11	Zeilen mit einem INNER JOIN ohne Schlüsselvergleich verbinden	377
10.2.12	Übungen	379
10.3	Beziehungen (Schlüsselbeziehungen) ändern	388
10.3.1	Beziehungen aus Zeilen aus einer Kindtabelle ändern	388
10.3.2	Beziehungen aus Zeilen einer Elterntabelle ändern (ON UPDATE CASCADE)	390
10.3.3	Übungen	394
10.4	Beziehungen (Schlüsselbeziehungen) aufheben oder löschen	399
10.4.1	Zeilen aus Kindtabellen auf NULL setzen	399
10.4.2	Zeilen aus Kindtabellen löschen	402
10.4.3	Zeilen aus Elterntabellen löschen	403
10.4.4	Übungen	406

11 Transaktionen 411

11.1	Forderungen an relationale Datenbanksysteme	412
11.2	Transaktionen verstehen	414
11.2.1	Allgemeiner Aufbau einer Transaktion	414
11.2.2	Einen atomaren Datenzustand mit Transaktionen sicherstellen	415
11.2.3	Transaktionen mit ROLLBACK rückgängig machen	416
11.2.4	Operationen mit Transaktionen isoliert ausführen	419
11.3	Übungen	421

12 Tabellenstrukturen verändern 423

12.1 Eine Tabelle umbenennen	423
12.2 Spalten einer Tabelle ändern	425
12.2.1 Eine Spalte umbenennen	425
12.2.2 Den Datentyp einer Spalte ändern	427
12.2.3 Eine Spalte als Primärschlüsselspalte definieren	429
12.2.4 Einer Spalte eine NOT NULL-Einschränkung zuordnen	430
12.2.5 Einer Spalte eine NULL-Einschränkung zuordnen	431
12.2.6 Einer Spalte einen Standardwert (DEFAULT VALUE) zuordnen	432
12.2.7 Einer Spalte eine UNIQUE-Einschränkung zuordnen	435
12.2.8 Eine Spalte mit einer CHECK-Einschränkung versehen	436
12.3 Spalten hinzufügen und entfernen	438
12.3.1 Einer Tabelle eine Spalte hinzufügen	438
12.3.2 Eine Spalte aus einer Tabelle entfernen	439
12.4 Beziehungen zwischen Tabellen herstellen und entfernen	440
12.5 Übungen	443

13 Mit SQL rechnen 451

13.1 Spaltenwerte addieren	452
13.2 Spaltenwerte subtrahieren	454
13.3 Spaltenwerte multiplizieren	455
13.4 Spaltenwerte dividieren	455
13.5 Den Restwert einer Division von Spaltenwerten berechnen	456
13.6 Nach dem Ergebnis einer Berechnung filtern	457
13.7 Nach dem Ergebnis einer Berechnung sortieren lassen	458
13.8 Übungen	459

14 Skalarfunktionen anwenden 463

14.1 Funktionen für Textwerte	464
14.1.1 Zeichenkette in Kleinbuchstaben umwandeln (LOWER)	465
14.1.2 Spaltenwerte in Großbuchstaben umwandeln (UPPER)	465

14.1.3	Spaltenwerte von führenden und endenden Leerzeichen befreien (TRIM)	466
14.1.4	Text aus Spaltenwerten extrahieren (SUBSTRING)	469
14.1.5	Textspaltenwerte verkettet ausgeben	471
14.1.6	Übungen	474
14.2	Funktionen für Zahlenwerte	476
14.2.1	Die Länge einer Zeichenkette ermitteln (CHAR_LENGTH, LEN)	477
14.2.2	Die Startposition einer Zeichenkette in einem Textwert ermitteln (POSITION, CHARINDEX)	478
14.2.3	Potenzen berechnen (POWER)	479
14.2.4	Eine Quadratwurzel berechnen (SQRT)	480
14.2.5	Übungen	481
14.3	Verschachtelte Funktionsaufrufe	482
14.3.1	Übungen	486
15	Bedingungslogik	487
15.1	Die CASE-Klausel	487
15.2	Bedingungslogik in einer Spaltenauswahlliste einer SELECT-Anweisung anwenden	488
15.3	Bedingungslogik in einer ORDER BY-Klausel anwenden	490
15.4	Übungen	492
16	Mit Zeit und Datum arbeiten	495
16.1	Datumsformate	496
16.2	Skalarfunktionen für Zeit- und Datumsangaben in SQL nutzen	496
16.2.1	Datum, Zeit und Zeitstempel vom Datenbankserver ermitteln lassen	497
16.2.2	Ergebnislisten mit einem Berichtsdatum versehen	498
16.2.3	Übungen	499
16.3	Zeit- und Datumsangaben formatieren	499
16.3.1	Datumsformatierung unter MySQL und MariaDB (DATE_FORMAT)	500
16.3.2	Datumsformatierung unter PostgreSQL (TO_CHAR)	505

16.3.3	Datumsformatierung unter MS SQL Server (FORMAT)	510
16.3.4	Übungen	514
16.4	Datumsangaben extrahieren (EXTRACT)	515
16.4.1	Übungen	519
16.5	Mit Datumsangaben rechnen	520
16.5.1	Mit Datumswerten rechnen unter MySQL und MariaDB	521
16.5.2	Mit Datumswerten rechnen unter PostgreSQL	523
16.5.3	Mit Datumswerten rechnen unter MS SQL Server	524
16.5.4	Übungen	526

17 Spaltenwerte gruppieren (GROUP BY)

17.1	Die Aggregatfunktion COUNT anwenden	530
17.1.1	Übungen	535
17.2	Die Aggregatfunktion SUM anwenden	535
17.2.1	Übungen	536
17.3	Die Aggregatfunktion AVG anwenden	537
17.3.1	Übungen	538
17.4	Die Aggregatfunktion MAX anwenden	539
17.4.1	Übungen	540
17.5	NULL-Werte berücksichtigen	541
17.5.1	Übungen	545
17.6	Nach aggregierten Werten einer Gruppierung filtern (HAVING)	545
17.6.1	Übungen	547
17.7	Nach zwei oder mehr Spalten gruppieren	548
17.7.1	Übungen	550

18 Mächtiges Werkzeug: Die Unterabfragen (Subqueries)

18.1	Unterabfragen, die in Korrelation zueinander stehen	554
18.1.1	Übungen	558
18.2	Unterabfragen, die nicht in Korrelation zueinander stehen	560
18.2.1	Übungen	565

18.3 Vergleichsoperatoren auf Unterabfragen mit ANY, SOME und ALL anwenden	566
18.3.1 Übungen	569
18.4 Auf die Existenz von Ergebniszeilen aus Unterabfragen prüfen (EXISTS)	571
18.4.1 Übungen	574

19 Views: Abfragen in virtuellen Tabellen speichern 575

19.1 Einfache Views anlegen	576
19.1.1 Übungen	579
19.2 Views und ORDER BY	581
19.2.1 Übungen	583
19.3 INSERT, UPDATE und DELETE auf Views anwenden	584
19.3.1 Eine INSERT-Anweisung auf Views anwenden	584
19.3.2 Eine UPDATE-Anweisung auf Views anwenden	588
19.3.3 Eine DELETE-Anweisung auf Views anwenden	590
19.3.4 Views, auf die keine INSERT-, DELETE- oder UPDATE-Anweisung angewendet werden kann	592
19.3.5 Übungen	593
19.4 Views entfernen oder ersetzen	596
19.4.1 Übungen	597

20 Abfragen mit einem Index optimieren 599

20.1 Einführung	599
20.2 Syntax: Index erstellen	602
20.3 Eine Tabelle mit vielen Zeilen generieren	602
20.4 Einen Index für eine Tabelle anlegen	604
20.5 Einen Index über mehrere Spalten anlegen	606
20.6 Den Index einer Tabelle löschen	609
20.7 Fremdschlüsselspalten indexieren	609
20.8 Übungen	613

21 Skalarfunktionen entwickeln 617

21.1 Funktionen mit einem Rückgabewert erstellen	618
21.1.1 MySQL und MariaDB	619
21.1.2 PostgreSQL	621
21.1.3 MS SQL Server	622
21.2 Funktionen mit Parameter entwickeln	623
21.2.1 MySQL und MariaDB	624
21.2.2 PostgreSQL	625
21.2.3 MS SQL Server	626
21.3 Funktionen entfernen	627
21.4 Übungen	628

22 Prozeduren entwickeln 631

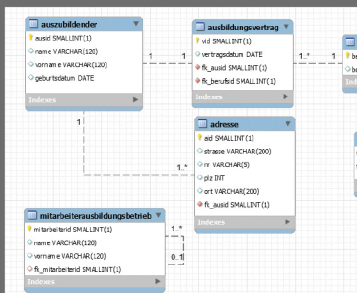
22.1 Prozeduren ausführen	631
22.2 Prozeduren anlegen	632
22.2.1 Prozeduren anlegen (MySQL und MariaDB)	632
22.2.2 Prozeduren anlegen (PostgreSQL)	634
22.2.3 Prozeduren anlegen (MS SQL Server)	635
22.3 Übung	636
 Index	 639

SQL verstehen und richtig einsetzen

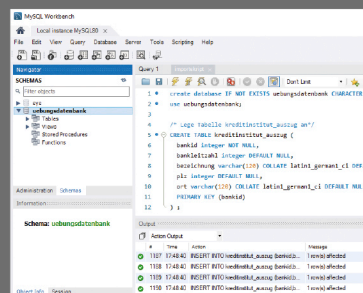
SQL ist das Fundament moderner Datenverarbeitung – und unverzichtbares IT-Fachwissen. Dieses Praxisbuch erklärt Ihnen Schritt für Schritt die Grundlagen und zeigt Ihnen, wie Sie Ihre Daten mit SQL richtig modellieren, verknüpfen und bearbeiten. Perfekt zum Einstiegen, Lernen und Nachschlagen.



Datenbanksystem einrichten



Konzepte verstehen



Tabellen erstellen und abfragen

Der richtige Einstieg für Sie

MySQL, MariaDB, PostgreSQL oder MS SQL Server? Mit welchem SQL-Dialekt Sie auch arbeiten, die detaillierten Erläuterungen in diesem Buch helfen Ihnen bei der Arbeit mit dem Datenbanksystem Ihrer Wahl.

SQL-Grundlagen und Praxiswissen

Ausführliche Erklärungen und Beispiele zeigen Ihnen, wie Sie Daten organisieren, abfragen und verändern. So lernen Sie, Redundanzen zu vermeiden und reibungslose Abfragen umzusetzen.

Von Übungsaufgaben profitieren

Anhand von Beispielen, Übungsaufgaben und Musterlösungen festigen Sie das Gelernte umgehend. Und sollten Sie dennoch etwas nachschlagen müssen, helfen Ihnen die übersichtlichen Zusammenfassungen.



Inkl. Übungsdatenbank zum Download



Michael Laube arbeitet seit vielen Jahren mit relationalen Datenbanken und SQL. Seine Erfahrung in der SQL-Entwicklung hat er in dieses Buch einfließen lassen. Er zeigt Ihnen, wie Sie SQL sicher nutzen und Probleme bei der Datenbankentwicklung elegant lösen.

Aus dem Inhalt

Grundlagen verstehen

- MySQL einrichten
- Tabellen und Daten
- Der SQL-Standard und seine Dialekte
- Tabellen anlegen, abfragen, ändern und löschen
- Berechtigungen

Für die Praxis

- Datenbanken modellieren und optimieren
- Tabellenbeziehungen
- Daten normalisieren und verknüpfen
- Skalarfunktionen und Prozeduren
- Spaltenwerte gruppieren
- Abfragen speichern
- Transaktionen
- Index anlegen

