webmasters press

Marco Emrich

Mit zahlreichen Übungen Schritt für Schritt zum Profi

Datenbanken & SQL

für Einsteiger

Datenbankdesign und MySQL in der Praxis

Auf Wunsch mit Online-Support! → Infos im Buch

- → Datenbanken fürs Web entwickeln
- → Das relationale Modell verstehen
- → Datenbanken mit UML modellieren
- → SQL-Befehle richtig einsetzen

Autorisiertes Curriculum für das **Webmasters Europe** Ausbildungs- und Zertifizierungsprogramm

we


webmasters
press

Datenbanken & SQL für Einsteiger

Datenbankdesign und MySQL in der Praxis

von Marco Emrich

Art.-Nr. 011720120

Version 3.5.1 vom 9.10.2013

Autorisiertes Curriculum für das Webmasters Europe Ausbildungs- und Zertifizierungsprogramm

www.webmasters-press.de
Nordostpark 7, 90411 Nürnberg, Germany

Informationen zu dieser Buchreihe

Dieses Buch ist Teil unserer Buchreihe zum Zertifizierungsprogramm des Europäischen Webmasterverbandes, Webmasters Europe e.V. (WE).

Das Webmasters Europe Ausbildungs- und Zertifizierungsprogramm ist modular aufgebaut und bietet Abschlüsse und Zertifizierungen auf verschiedenen Ebenen an – von der Experten-Zertifizierung für einzelne Themen bis hin zu Diploma-Abschlüssen für Webdesigner, Web-Entwickler, Webmaster und Online Marketing Manager. Nähere Informationen dazu finden Sie unter http://de.webmasters-europe.org/bildungsprogramm

Unsere Buchreihe bietet Ihnen nicht nur eine fundierte und praxisnahe Einführung in das jeweilige Thema, sondern ist von Webmasters Europe e.V. offiziell autorisiert zur Vorbereitung auf die WE-Zertifikatsprüfungen. Damit haben Sie die Garantie, dass alle prüfungsrelevanten Themen behandelt werden.


webmasters
press

Jetzt Buch registrieren und Online-Vorteile sichern!

Registrieren Sie Ihr Buch auf webmasters-press.de und sichern Sie sich folgende Online-Vorteile:

- Laden Sie Begleitmaterial und Lösungen zu den Übungsaufgaben herunter
- Erhalten Sie kostenfrei Patches und Updates zu Ihrem Buch
- Aktualisieren Sie kostengünstig auf neue Auflagen des Buchs

www.webmasters-press.de/register

Registrierungscode: 5183437152

Erweitern Sie Ihr Buch zum tutoriell betreuten Online-Seminar!

- Support und persönliche Betreuung durch qualifizierten Tutor
- Einsendeaufgaben mit Korrektur und Feedback
- Qualifiziertes Zeugnis
- Online-Campus mit vielen Funktionen

www.webmasters-fernakademie.de/upgrade

Registrierungscode: 5183437152

Inhaltsverzeichnis

Vorwort

Kaum noch eine Website kann es sich heutzutage erlauben, mit rein statischen Inhalten aufzuwarten. So ist im *Web 2.0* die Mitarbeit der Benutzer gefragt. Onlineshops aktualisieren ständig ihre Inhalte. Selbst scheinbar statische Webseiten verwenden meist ein ausgeklügeltes ***CMS*** (Content Management System), um die Pflege zu erleichtern.

Das moderne *World Wide Web* verarbeitet täglich eine gigantische Datenflut, die außerdem stetig wächst. Wohin also mit all den Daten?

Ein Konzept, das sich seit fast 40 Jahren als **das** Konzept der Datenhaltung schlechthin erwiesen hat, hat auch die Webentwicklung im Sturm erobert. Die Rede ist vom ***relationalen Datenbankmodell***. Ein Modell, das allen Versuchen es abzulösen trotzte und bis heute als stabil, fundiert und ausgereift gilt. Wenn der führende Datenbankexperte *C. J. Date* recht behält, wird das relationale Datenbankmodell auch in 100 Jahren noch vorherrschend sein (Date 2005).

> *Hundert Jahre in die Zukunft blickend, erwarte ich, dass Datenbanksysteme immer noch auf Codds relationalem Modell basieren. Warum? Die Grundlagen des Models – Mengenlehre und Prädikatenlogik – sind ihrerseits grundsolide. Elemente der Prädikatenlogik im Besonderen reichen gut 2000 Jahre zurück – wenigstens bis zu Aristoteles (384-322 v.Chr.).*

Keine Sorge, Sie lernen hier nicht nur 40 Jahre alte Konzepte, sondern auch moderne Erkenntnisse. Erkenntnisse, die erst kürzlich den Schritt in den Mainstream der Softwareentwicklung geschafft haben und die den Unternehmen, die sie bereits anwenden, enorme Wettbewerbsvorteile bescheren. Dazu zählen beispielsweise Datenbank-Migrationen oder objekt-relationales Mapping. Auch die Datenbanktheorie lernen Sie in der aktuellsten Version kennen. Sie bekommen die Mittel an die Hand, die Sie benötigen, um erfolgreich moderne ***datenbankgestützte Webanwendungen*** zu entwickeln.

Ich werde Ihnen sowohl ein tiefes Verständnis des relationalen Datenbankmodells als auch alle wichtigen Handgriffe für die Praxis vermitteln. Sie werden von meiner langjährigen Berufserfahrung in der Webentwicklung profitieren und lernen, wie Sie typische Fallen gekonnt umschiffen.

Für Feedback, Fragen und auch Kritik habe ich immer ein offenes Ohr unter *m.emrich@webmasters.de*.

Ich freue mich, Sie nun in die spannende Welt hinter den Kulissen der datenbankgestützten Webanwendungen mitzunehmen.

Viel Spaß beim Durcharbeiten dieses Buches und viel Erfolg beim Bau Ihrer Webanwendungen!

Ihr Marco Emrich

1 Einführung

In dieser Lektion lernen Sie:

- wie Sie dieses Buch verwenden können.
- wichtige Persönlichkeiten aus dem Datenbankumfeld kennen.

Wenn ich weitergesehen habe, dann stehend auf den Schultern von Riesen.

Newton (1676)
gefunden bei C.J. Date

1.1 Allgemeines

Vorab möchte Ihnen noch kurz erläutern, wie Sie dieses Buch verwenden können.

Aufbau der Lerneinheiten

Das vorliegende Buch ist in kleine Lerneinheiten – genannt Lektionen – aufgeteilt. Jede Lektion schließt mit Zusammenfassung, Wiederholungsfragen (*Testen Sie Ihr Wissen*) und einem Übungsblock ab. An die »normalen« Übungen schließt sich dann ein Block mit *Zusatzübungen* an. Ich empfehle Ihnen, die Übungen immer vollständig zu bearbeiten. Sie helfen Ihnen, das Erlernte zu festigen – getreu dem Grundsatz *learning by doing*. Die Zusatzübungen sind nicht zwingend erforderlich. Sie sind manchmal etwas schwieriger und geben Ihnen Gelegenheit, über das Geforderte hinaus zu gehen. Falls Sie der Meinung sind, dass Ihnen einfach noch etwas mehr Übung nicht schaden könnte, kann ich Ihnen die Zusatzübungen natürlich auch empfehlen.

1.2 Auf den Schultern von Riesen…

In der Welt der relationalen Datenbanken gibt es viele verschiedene Meinungen. Diese weichen oft nur in Details, manchmal aber auch sehr stark voneinander ab. Unter anderem betrifft das

- das Verständnis des relationalen Datenbankmodells.
- Konzepte des Datenbankentwurfs.
- die allgemeine Vorgehensweise bei der Softwareentwicklung in Bezug auf Datenbanken.

Meine Sichtweise ist aus jahrelanger Praxiserfahrung in der Webentwicklung gewachsen. Außerdem schließe ich mich der Meinung bekannter Forscher und praktizierender Entwickler an. Insbesondere sind das diese vier:

- Edgar Frank »Ted« Codd
- Christopher J. Date
- Scott W. Ambler
- Martin Fowler

Hier finden Sie einen kurzen Abriss zu diesen wichtigen Personen der Datenbankgeschichte. Es sind die Lehren dieser vier, auf denen die Inhalte dieses Buches im wesentlichen basieren.

Ted

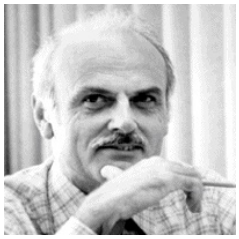


Abb. 1.1 Edgar F. Codd (Quelle: Wikipedia)

Edgar Frank Codd – »Ted«, wie ihn seine Freunde nannten, wurde 1923 in England geboren und ist der Erfinder des relationalen Datenbankmodells. 1969 stellte er seinem Arbeitgeber IBM das erste Werk über das relationale Datenbankmodell vor (Codd, 1969). Zusammen mit Christopher J. Date gründete er 1984 ein Consulting-Unternehmen[1] und arbeitete mit ihm an der Weiterentwicklung des Modells. Bis zu seinem Tod 2003 veröffentlichte er noch neue Erkenntnisse. Mehr zu seinem Leben erfahren Sie bei Spicer (2003) und bei Wikipedia[2].

---

1 siehe Campbell-Kelly (2003)
2 en.wikipedia.org/wiki/Edgar_F._Codd

Chris

Abb. 1.2 C.J. Date

Christopher J. Date (geboren 1941) ist Autor zahlreicher Bücher und Artikel und gilt heute als **der** Experte für relationale Datenbanken. Codd lernte er 1970 kennen und wurde einer der ersten Befürworter des relationalen Modells. Er sagt von sich selbst, dass es seine größte Errungenschaft war, Codds Theorien für eine breite Öffentlichkeit verständlich darzustellen (Haigh, 2007).

> *Ted war der Typ, der den Kram erfunden hat, und ich derjenige, der es erklären musste.*
>
> aus Darrow B. (2004)

Heute wartet und verbessert er noch immer das relationale Modell[1]. Mehr zu seinem Leben erfahren Sie bei Wikipedia[2] und Haigh (2007).

---

1 aktuellere Fassungen können Sie in Date (2004, 2005 und 2007) nachlesen
2 en.wikipedia.org/wiki/Christopher_J._Date

Scott

Abb. 1.3 Scott Ambler (Ambler Homepage)

Bekannt geworden ist Scott Ambler durch seine agilen Vorgehensmodelle, seine Arbeit im Modellierungssektor und seine Errungenschaften beim objekt-relationalen Mapping. Er ist Autor und Co-Autor mehrerer Bücher zu den genannten Themen. Seine beiden Datenbank-Bücher (Ambler, 2003, 2006) berücksichtigen nicht nur die technischen, sondern auch die menschlichen Aspekte der Datenbankentwicklung.

Auf seiner Homepage[1] finden Sie viele interessante Artikel, und unter www.agiledata.org widmet er sich exklusiv der Datenbankentwicklung. Sie können Scott auch bei einer Präsentation im Video[2] zusehen.

---

1 www.ambysoft.com
2 www.infoq.com/presentations/ambler-database-refactoring

Martin

Abb. 1.4 Martin Fowler (Oshineye, 2007)

Martin Fowler ist der leitende Wissenschaftler bei Thought Works. Er hat sich in vielen Bereichen der Softwareentwicklung verdient gemacht. Lassen wir ihn sich kurz vorstellen (Fowler 2003):

> *Ich bin Autor, Konferenzsprecher, freier Berater und hab' generell ein großes Mundwerk im Bereich der Softwareentwicklung. Ich konzentriere mich auf das Design von Enterprise-Software, verfolge, was gutes Design ausmacht und welche Praktiken dazu nötig sind. Ich bin Pionier in der Objekt-Orientierung, im Refactoring, bei Mustern, Agilen Methoden, Domain Modellierung, Extreme Programming und der Unified Modeling Language (UML).*

Viele moderne Webframeworks wie *Rails*, *Django*, *Doctrine* oder *Symfony* greifen auf seine Muster[1] zurück. Wenn Sie Martin als Redner sehen möchten, werfen Sie einen Blick auf seine Keynote von der *Railsconf 2006*[2].

---

1 Meist *Active Record* oder *Data Mapper* aus (Fowler 2002)
2 www.scribemedia.org/2006/07/03/rails-martin-fowler

1.3 Voraussetzungen

Das Wissensumfeld relationaler Datenbanken ist recht umfangreich. Es gibt viele konkurrierende Ansätze zum Entwurf und zur Verwendung von Datenbanken. Dieses Buch will keine wändefüllende Enzyklopädie sein, sondern Ihnen einen möglichst schnellen praxisnahen Einstieg ermöglichen. Deswegen stelle ich Ihnen nicht alle Möglichkeiten und Strategien vor, sondern beschränke mich auf eine Vorgehensweise, die vor allem für Webanwendungen gut geeignet ist. Am besten lässt sich dieses Buch einsetzen, wenn folgende Voraussetzungen vorliegen:

- Sie entwickeln eine Webanwendung auf Kundenwunsch.
- Die Anwendung wird praktisch eingesetzt und ist kein Forschungsprojekt.
- Eine Datenbank sorgt für die persistente Datenspeicherung. Es ist nicht erforderlich, dass weitere Anwendungen, außer Ihrer Webanwendung, (direkt) auf die Daten zugreifen.
- Es ist auch nicht erforderlich, dass ein Benutzer direkt mit der Datenbank arbeitet.[1] Ein Benutzer arbeitet immer mit der Webanwendung.
- Es gibt keinen Grund, Daten in mehreren (verschiedenen) Datenbanken für die gleiche Anwendung zu speichern. Anders ausgedrückt: Zu genau einer Webanwendung gehört genau eine Datenbank.
- Es sind keine Altdaten (***legacy data***) zu übernehmen.
- Es sind kein Altanwendungen (***legacy applications***) anzubinden.

---

1 Früher war es üblich, dass Anwender direkt mit Hilfe einer Datenbankabfragesprache wie z. B. SQL gearbeitet haben. Heutzutage arbeiten Endanwender aber nur noch mit grafischen Oberflächen (GUI). Das SQL bleibt Anwendungsentwicklern vorbehalten.

Sollten Sie in der Praxis nicht diese Situation vorfinden, so können Sie die im Buch vorgestellten Konzepte natürlich dennoch anwenden. Sie müssen dann zusätzliche Punkte beachten (zum Beispiel den Import von Altdaten), auf die ich im Rahmen dieses Buches nicht eingehen kann.

2 Datenbanken

In dieser Lektion lernen Sie:

- wozu Datenbanken verwendet werden.
- welche Rolle relationale Datenbanken im World Wide Web spielen.
- was genau eine Datenbank eigentlich ist.
- welche Datenbank Management Systeme es gibt.

2006 wurden weltweit über 15.2 Milliarden Euro für relationale Datenbanken ausgegeben.

SQL Industry News (2007)

2.1 Warum Datenbanken?

Datenbanken werden überall dort benötigt, wo Daten strukturiert gespeichert werden müssen. Firmen z. B. benötigen Datenbanken, um ihre Kundenkontakte zu speichern oder ihre Aufträge zu verwalten.

2.2 Datenbanken im Web

Auch im World Wide Web spielen Datenbanken eine große Rolle. Viele Webanwendungen wären mit rein statischen HTML-Seiten nicht realisierbar. Webanwendungen nutzen serverseitige Programmierung (z. B. mit PHP, Ruby, Python oder Java), um auf Datenbanken zuzugreifen und die darin gespeicherten Daten in Webseiten zu integrieren. Mit Hilfe dieser Technologien können Sie ***dynamische Webseiten*** programmieren.

> Dynamische Webseiten sind Webseiten, die erst zum Zeitpunkt des Aufrufs durch einen Webseiten-Besucher ***generiert*** werden. Auf diesen Seiten können die aktuellsten Informationen aus einer Datenbank angezeigt werden.

Beispiel

In Abb. 2.2 z. B. sehen Sie die Mitarbeiteransicht einer Seminarverwaltung. Damit pflegt ein Mitarbeiter einer Bildungseinrichtung Seminare mit Name, Beschreibung, Kategorie usw. ein. Alle Änderungen sind in der Kundenansicht für Kunden und Besucher sofort sichtbar (siehe Abb. 2.1). Aktuelle Informationen entnimmt die Webanwendung bei jedem Aufruf automatisch aus der Datenbank.

Die Daten manuell in statische HTML-Seiten einzupflegen und aktuell zu halten, wäre dagegen sehr aufwändig und fehleranfällig. Eine Datenbank spart also Zeit und Kosten und hilft, Fehler zu vermeiden.

Abb. 2.1 Produktdatenbank für Seminare (Kundenansicht)

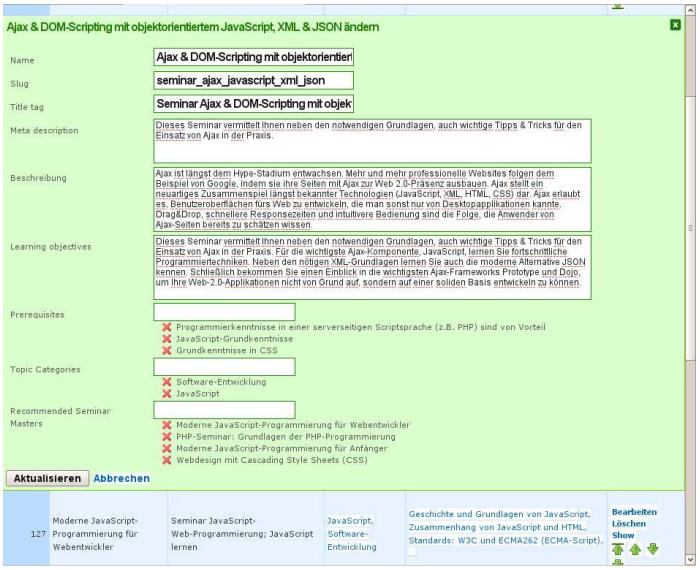


Abb. 2.2 Produktdatenbank für Seminare (Mitarbeiteransicht)

2.2.1 Webanwendungen

Websites, die aus vielen dynamischen Seiten bestehen, werden als ***Webanwendungen*** bezeichnet. In seiner Anfangszeit bestand das Web im Wesentlichen aus statischen Seiten und wurde eher als eine Art Bibliothek betrachtet. Heute sind viele Websites aber als Anwendungen zu verstehen, die mit klassischen Desktopanwendungen in Konkurrenz treten. Typische Webanwendungen sind z. B.

- **Produkt-Datenbanken**
  Durch Datenbankanbindung lassen sich z. B. Produkt- und Preislisten sehr effizient ins Web stellen.
- **Informationssysteme**
  Bestimmte Angebote, wie Suchmaschinen (z. B. www.google.de) oder Informationsportale (z. B. Die *International Movie Database* www.imdb.com, wo Sie zu nahezu allen Filmen Schauspieler, Regiesseur, Handlungsverlauf, Bewertungen, uvm. finden) wären ohne Datenbanken überhaupt nicht denkbar. Zur Realisierung von Suchfunktionen wird fast immer eine Datenbank eingesetzt.
- **Online-Shops**
  Online-Shops müssen nicht nur Produkte und Preise über das Internet zur Verfügung stellen, sondern auch Features wie Warenkorb, Bestellwesen, Bezahlsystem und eventuell Auftragsverfolgung. Datenbanken verwalten hier Kunden, Produkte, Bestellungen, Rechnungen u.v.m.
- **Online-Spiele**
  Auch der Entertainment-Sektor setzt heutzutage verstärkt auf Datenbanken. Viele moderne Onlinespiele speichern sowohl die Benutzerdaten als auch die Daten der Spielelemente in Datenbanken. Mein Hobby-Projekt fantasy-cards.net (ein Online-Kartenspiel) verwaltet z.B. eine Rangliste, die Ergebnisse einzelner Partien und die Errungenschaften der Spieler (Achievements) in einer Datenbank.
- **Web 2.0 Anwendungen**
  Gerade in der schönen neuen Welt des Web 2.0 sind Datenbanken der stille Superstar.

Web 2.0-Sites leben vom Mitmachen der Benutzer – und deren Daten wollen gespeichert werden. Es beginnt mit der Erfassung der Registrierungsdaten (Benutzername, Name, Passwort, Adresse usw.) und geht bis hin zu komplexen Funktionen, die für moderne Social Networks (z. B. www.xing.com, www.myspace.com oder www.linkedin.com) typisch sind.

2.2.2 Web-Komponenten

Große Webanwendungen integrieren oftmals verschiedene Komponenten, die für sich alleine schon kleine Anwendungen darstellen. Viele davon müssen ebenfalls ihre Daten permanent speichern. Typische Komponenten sind:

- Benutzerregistrierung
- Forum
- Wiki
- Blog
- Content Management System (CMS)
- Nachrichtensystem

2.3 Persistenz

Zu welchem Zweck setzen Webanwendungen nun Datenbanken ein?

Webanwendungen nutzen Datenbanken in erster Linie um ihre Daten zu persistieren. Persistenz bedeutet – vereinfacht ausgedrückt – dass die Daten nicht nur im RAM des Webservers gehalten werden, sondern auch einen Reboot überleben. Neben relationalen Datenbanken können dazu auch andere Technologien eingesetzt werden, z. B. einfache Textdateien, XML-Dateien oder NoSQL-Datenbanken. Am häufigsten greifen Webanwendungen jedoch auf relationale Datenbanken zurück.

Persistenz ist aber nur ein Aspekt. Relationale Datenbanken haben weitere Vorteile, die sie gerade für typische Webanwendungen attraktiv machen. Sie bieten z. B. mächtige Abfragemöglichkeiten, um schnell an die gewünschten Daten heranzukommen. Zudem gibt es die Möglichkeit, Teile der Applikationslogik in die Datenbank zu verlagern. Ob das tatsächlich sinnvoll ist (und in welchem Umfang), hängt von vielen Faktoren ab[1].

---

1 Eine ausführliche Diskussion finden Sie in einem Blogartikel von Fowler (2003b), siehe www.martinfowler.com/articles/dblogic.html

2.4 Was ist eine Datenbank?

Eine ausgezeichnete Definition des Begriffes Datenbank[1] finden Sie in Wikipedia[2].

---

1 Gemeint ist *Datenbank* im Sinne der EDV. In anderen Bereichen, z. B. Recht, gibt es andere Bedeutungen.
2 en.wikipedia.org/wiki/Database

> *Eine Datenbank ist eine strukturierte Sammlung von Daten, die verwaltet werden, um die Bedürfnisse einer Gruppe von Anwendern zu decken. Die Struktur entsteht durch die Organisation der Daten mit Hilfe eines Datenbankmodells*

Diese Definition beinhaltet drei wichtige Aussagen:

- Eine ***Datenbank*** ist eine strukturierte Sammlung von Daten.
- Die Struktur entsteht durch die Organisation der Daten mit Hilfe eines ***Datenbankmodells***.
- Die Daten werden *verwaltet*, um die Bedürfnisse einer Guppe von Anwendern zu decken.

2.4.1 Struktur

> *Eine **Datenbank** ist eine strukturierte Sammlung von Daten.*

Strukturiert bedeutet, dass die Daten nicht »wild verstreut« sind, sondern in irgend einer Art organisiert. Es gibt verschiedene Möglichkeiten, Daten strukturiert abzulegen, z. B.

- als Liste oder Aufzählung in einer *Textverarbeitung* (z. B. *LibreOffice Writer* oder *Microsoft Word*),
- in einer *Tabellenkalkulation* (z. B. *OpenOffice.org Spreadsheet* oder *Microsoft Excel*),
- als *Dateien* in ein Dateisystem,
- in *Auszeichnungssprachen* wie XML usw.

Sie könnten also z. B. auch eine Liste in einer Textverarbeitung als Datenbank bezeichnen – das ist nicht gemeint. Struktur alleine reicht nicht aus. Der zweite Satz sagt, dass der Struktur einer Datenbank ein bestimmtes Datenbankmodell zugrunde liegt.

2.4.2 Datenbankmodelle

> *Die Struktur entsteht durch die Organisation der Daten mit Hilfe eines **Datenbankmodells**.*

Es gibt verschiedene sogenannte ***Datenbankmodelle***, die sich in der Praxis etabliert haben. Die bekanntesten sind:

- Netzwerkmodell
- hierarchisches Modell
- relationales Modell
- objektorientiertes Modell

- dokumentenorientiertes Modell

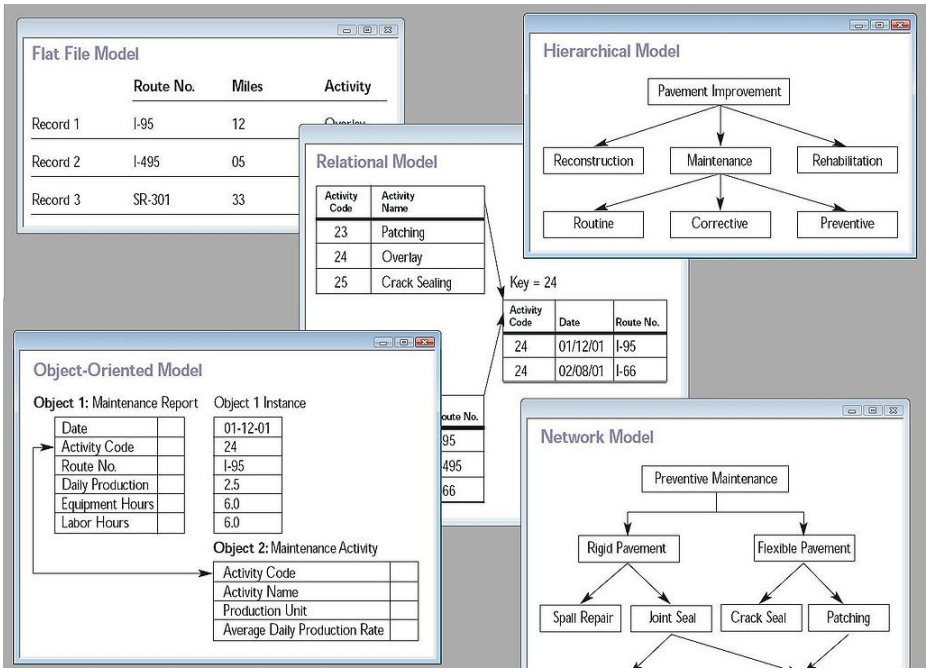


Abb. 2.3 Datenbankmodelle (Quelle: de.wikipedia.org/wiki/Datenbankmodell)

Das Datenbankmodell bestimmt die Struktur zur Ablage der Daten und wie Verbindungen zwischen den Daten repräsentiert werden. Beispielsweise folgt das hierarchische Modell einer **Baumstruktur**. Wollten Sie z. B. Unternehmen repräsentieren, könnten Sie auf der obersten Ebene die Unternehmen angeben und darunter (als Äste) deren Abteilungen (Vertrieb, Service, Einkauf, usw). Diese wiederum untergliedern sich in mehrere Mitarbeiter. Abteilungsübergreifend arbeitende Mitarbeiter lassen sich so nur schwer darstellen.

Das relationale Model dagegen verwendet (vereinfacht ausgedrückt) **Tabellen** zur Datenorganisation. Das Unternehmensbeispiel könnten Sie mit Hilfe der Tabellen *unternehmen*, *abteilungen* und *mitarbeiter* realisieren.

Das relationale ist das mit Abstand meistverbreitete Datenbankmodell. Es ist seit über 40 Jahren im Einsatz und hat sich vielfach bewährt. Andere, längst totgeglaubte, Modelle erleben jedoch seit 2009 eine Renaissance. Unter dem Begriff ***NoSQL*** hat sich eine Gegenbewegung von Datenbanksystemen gebildet, die nicht auf dem relationalen Modell basieren. Der Begriff ***NoSQL*** ist an dieser Stelle etwas irreführend. Tatsächlich müsste es »nicht-relational« heißen.

Das relationale Modell ist ein universales Modell, das sich für eine sehr breite Palette von Problemstellungen einsetzen lässt. Im Gegenzug haben andere Modelle Vorteile, wenn die Problemstellung genau zum Modell passt. So lassen sich z. B. soziale Graphen, wie sie Social Networks verwenden (wer ist mit wem befreundet?), besser in einer spezialisierten Graphendatenbank speichern. Einen guten Überblick über den Stand der NoSQL-Datenbanken gibt Stefan Endlich (2012)[1].

---

1 www.infoq.com/articles/State-of-NoSQL

In diesem Buch möchte ich Ihnen das vorherrschende relationale Model näher bringen,

dass für eine große Menge von Anwendungsfällen bestens geeignet ist. In Lektion 7 lernen Sie das Model im Detail kennen.

2.4.3 Das Datenbank Management System

Betrachten Sie den noch verbleibenden Teil der Definition:

> *Die Daten werden verwaltet, um die Bedürfnisse einer Guppe von Anwendern zu decken.*

Um Daten zu verwalten, benötigen Sie ein Programm, das diese Aufgabe übernimmt. Solche Programme werden als ***Datenbank Management System*** (kurz: ***DBMS***) bezeichnet. Ein DBMS, das Datenbanken nach dem relationalen Modell verwaltet, ist ein ***relationales DBMS***, kurz ***RDBMS***.

Um es noch einmal klar herauszustellen:

Eine ***Datenbank (DB)*** ist eine Sammlung von Daten, denen ein Datenbankmodell zugrundeliegt. Ein ***Datenbank Management System (DBMS)*** ist ein Programm, das die Daten bzw. Datenbanken verwaltet.

Im Sprachgebrauch wird häufig der Begriff Datenbank gebraucht, obwohl eigentlich ein DBMS gemeint ist. Ich werde aber, um Verwechslungen zu vermeiden, die beiden Begriffe immer in ihrer korrekten Bedeutung verwenden.

Christopher sagt…

> *Leider wird [der Begriff »Datenbank«] auch noch allzu oft im Sinne von DBMS verwendet, aber diese spezielle Lesart ist streng zurückzuweisen. Wenn wir »Datenbank« für DBMS sagen, wie wollen wir dann eine Datenbank nennen?*
>
> Date (2007)

Im übrigen kann ein DBMS auch mehrere Datenbanken verwalten, z. B. eine Datenbank für Kunden und Bestellungen, eine weitere für Lieferanten usw.

2.5 Anforderungen an ein DBMS

Ein DBMS ist aber kein einfaches Stück Software. Es muss eine ganze Reihe spezieller Anforderungen erfüllen. Welche Anforderungen das aber genau sind – nun, da gehen die Meinungen »ein wenig« auseinander. Ich beschränke mich deswegen auf lediglich vier Anforderungen, die allgemein anerkannt sind und gerade auch für Webanwendungen als essentiell gelten.

- Unterstützung spezieller Suchabfragen
- Datensicherheit
- Datenintegrität
- Mehrbenutzerfähigkeit

Betrachten Sie die Anforderungen im Detail.

Unterstützung spezieller Suchabfragen

Soll beispielsweise in einer Personaldatenbank nach allen Mitarbeitern gesucht werden, die mehr als 2000,- Euro verdienen, höchstens 40 Jahre alt sind und in der Abteilung *Softwareentwicklung* arbeiten, so ist eine spezielle Suchfunktion erforderlich.

Insbesondere müssen Sie sogenannte ***Ad-Hoc-Querys*** absetzen können. Das sind Suchanfragen, die Sie bei der ursprünglichen Datenbankplanung nicht berücksichtigt haben. In einem realen Projekt kommt es aber immer wieder vor, dass Sie unvorhergesehene Anfragen stellen müssen. Glücklicherweise unterstützen alle gängigen modernen relationalen DBMS solche Anfragen mit Hilfe der Sprache *SQL*.

Datensicherheit

Durch Betriebssystemfehler, Stromausfall, Anwendungsfehler und andere Probleme können Datenverluste auftreten. Ein DBMS verfügt deswegen über Mechanismen zum Schutz der Daten. Beispielsweise kann ein DBMS einen sogenannten ***Dump*** erstellen. Ein Dump ist eine Sicherung (***Backup***) des aktuellen Stands einer Datenbank, den Sie im Notfall wieder einspielen können. Weitergehende Features sind z. B. das ***Point-In-Time-Recovery***, mit dem sich jeder beliebige Zeitpunkt des Datenstandes wiederherstellen lässt. Falls Sie auf eine schnelle Wiederherstellung oder hohe Ausfallsicherheit Wert legen, bieten viele Datenbank Management Systeme ***Replikationsmechanismen***, die es erlauben, den Datenbestand auf einem zweiten System aktuell zu halten. Falls gewünscht, kann das zweite System dann beim Ausfall des ersten automatisch einspringen.

Bestehende Datenbank Management Systeme unterscheiden sich sehr stark in ihren Datensicherheitsmechanismen. Zum einen unterstützen verschiedene Systeme unterschiedliche Mechanismen. Zum anderen unterscheiden sich selbst die gleichen Mechanismen erheblich in ihrer Umsetzung. Allein mit den verschiedenen Konzepten der Replikation ließen sich mehrere Bücher füllen.

Datenintegrität

Datenintegrität bedeutet, dass die abgelegten Daten konsistent, d. h. nicht widersprüchlich

sind. Beispielsweise darf der gleiche Benutzer nicht zweimal mit unterschiedlichem Geburtsdatum hinterlegt sein.

Außerdem können weitere Regeln hinterlegt werden, die zusätzliche logische Fehler verhindern. Als logischer Fehler könnte z. B. das Eintragen eines negativen Preises gelten. So ein Fehler kann dramatische Folgen haben. Stellen Sie sich vor, ein Mitarbeiter eines Onlineshops hat für den Preis eines Buches -30 € eingetragen. Ich bin mir sicher, dass das Buch reißenden Absatz findet – über den sich die Geschäftsführung aber nicht unbedingt freuen wird…

Viele Anwenderfehler können durch sogenannte Integritätsprüfungen schon vor dem Speichern in der Datenbank abgefangen werden. Bei der Webentwicklung müssen Sie sich entscheiden, welche Integritätsprüfungen Sie in der Datenbank (mit SQL) und welche Sie in der Anwendung (z. B. mit PHP oder Ruby) hinterlegen.

Mehrbenutzerfähigkeit

Zugriffssynchronisations-Mechanismen regeln den gleichzeitigen Zugriff mehrerer Benutzer auf die Daten einer Datenbank. Änderungen dürfen nicht zu Inkonsistenzen führen. Wenn etwa zwei Reisebüros gleichzeitig versuchen, den letzten Platz des gleichen Flugs zu buchen, darf dieser Platz natürlich nur einmal vergeben werden. Um Probleme beim gleichzeitigen Zugriff zu vermeiden, verwenden Datenbank Management Systeme üblicherweise sogenannte ***Sperren*** (***Locks***) und ***Transaktionen*** (***Transactions***).

2.6 Verbreitete RDBMS

DBMS	Hersteller/Entwickler	Lizenzmodell
MySQL www.mysql.com	MySQL AB	OpenSource / kommerziell
PostgreSQL www.postgresql.org	PostgreSQL Global Development Team	OpenSource
MariaDB mariadb.org	Monty Program Ab	OpenSource
SQLite www.sqlite.org	SQLite International Team	OpenSource
HSQL-DB hsqldb.sourceforge.net	HSQL-DB Development Team	OpenSource
Oracle www.oracle.com	Oracle	kommerziell
DB2 www.ibm.com	IBM	kommerziell

Tabelle 2.1 Beispiele für relationale Datenbank-Management-Systeme

Tabelle 2.1 zeigt einen kleinen Ausschnitt gängiger relationaler DBMS. Eine umfangreiche Liste finden Sie auf Wikipedia[1]. In den letzten zehn Jahren haben sich im Bereich der Webentwicklung insbesondere OpenSource-Produkte etabliert. Sie lassen sich kostenlos einsetzen und sind auch im Quellcode erhältlich[2] . Außerdem müssen Sie sich nicht hinter ihren kommerziellen Gegenstücken verstecken. Im Funktionsumfang sind zwar geringfügige Abstriche zu machen[3], in Sachen Performance, Administration und Support sind sie aber gleichauf und teilweise sogar überlegen (Horstmann, 2006)[4]. Die zusätzlichen Features sind für die Praxis von Webanwendungen auch weitestgehend irrelevant. Deswegen gilt:

---

1 en.wikipedia.org/wiki/Comparison_of_relational_database_management_systems
2 mit all den bekannten Vorteilen der OpenSource-Entwicklung, siehe z. B. Raymond (2000), www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/
3 z. B. fehlen allen OpenSource-Datenbanken ***Materialized Views***[1]
4 www.heise.de/open/Freie-Datenbanken-im-Unternehmenseinsatz-Ein-Vergleich—/artikel/70100/0

> OpenSource-DBMS lassen sich für den Einsatz in Webanwendungen uneingeschränkt empfehlen.

Einen ausführlichen Vergleich aktueller OpenSource-Datenbanken finden Sie bei Horstmann (2006).

2.7 MySQL

In diesem Buch lernen Sie das DBMS *MySQL* kennen. Es ist der Markführer unter den relationalen Datenbankmanagementsystemen (siehe Abb. 2.4) und gilt als erste Wahl für Webanwendungen. Deswegen ist es in vielen Hostingpaketen bereits integriert. Als Hauptvorteile gelten **Hohe Performance** und **einfache Handhabung**

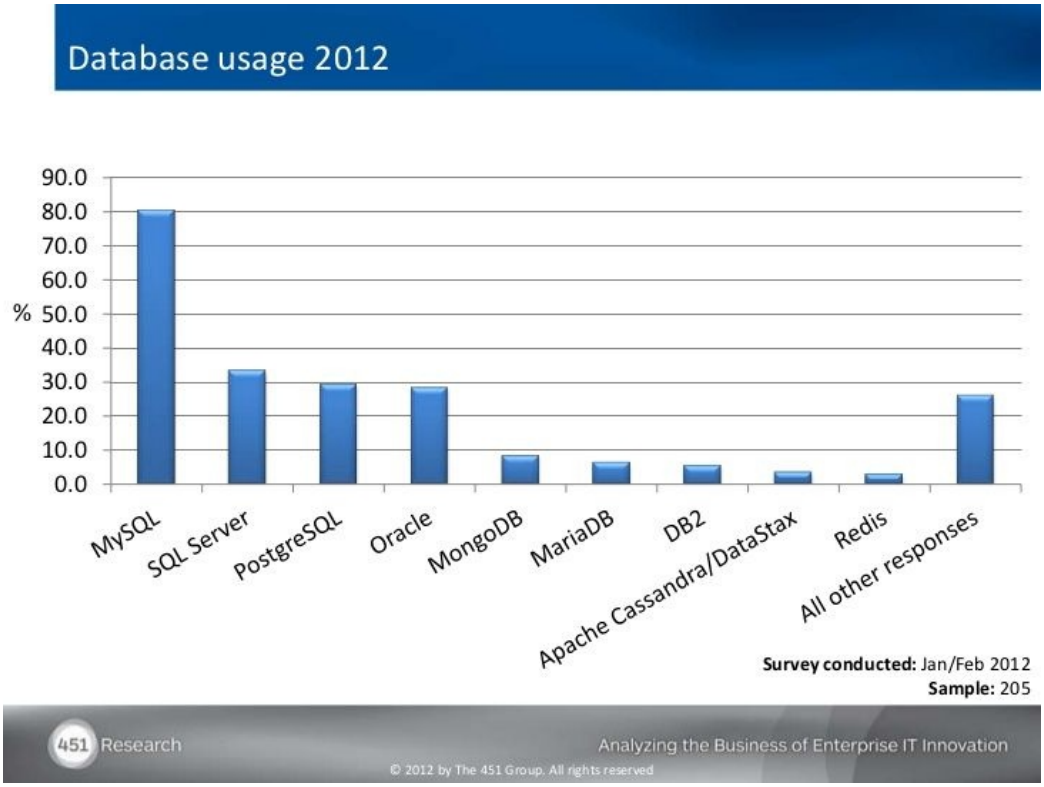


Abb. 2.4 Datenbankeinsatz 2012, Quelle: Matthew Aslett (2012)[1]

Im OpenSource-Bereich sind die RDBMS *MariaDB* und *PostgresQL* direkte Konkurrenten. Bei MariaDB handelt es sich um einen voll-kompatiblen Ersatz für MySQL, so dass alle Codebeispiele und Übungen in diesem Buch auch mit MariaDB funktionieren. PostgresQL ist ebenfalls eine sehr gute Alternative, erfordert aber stellenweise andere SQL-Syntax als MySQL, so dass Sie nicht alle Beispiele direkt übertragen können.

Fun-Fakt

MySQL ist nach Monty Widenius' (dem MySQL-Erfinder) ältester Tochter »My« benannt. Seine beiden anderen DBMS-Produkte/Kinder heißen Max(DB) und Maria(DB).

2.8 Das Client-Server-Prinzip

Ein DBMS liegt meist als ***Serversystem*** vor[1] , das dann als ***Dienst*** (engl. ***Service***) arbeitet. Dienste verfügen nicht über grafische Benutzerschnittstellen und werden auch nicht von Benutzern gestartet, sondern vom Betriebssystem verwaltet. Ein Dienst bietet Ihnen keine direkte Bedienmöglichkeit an, er arbeitet im Hintergrund. Um auf die Daten zugreifen zu können, sind ***Datenbank-Clients*** (kurz: ***DB-Clients***) notwendig.

---

1 Die Alternative dazu sind DBMS als Bibliothek (engl. Library) – sogenannte eingebetteten Datenbanksysteme (engl.: Embedded Databases). Sie werden direkt in eine Anwendung integriert. Viele Anwendungen, die lokal auf dem Rechner ausgeführt werden, verwenden im Hintergrund ein DBMS, um ihre Daten zu organisieren. So benutzen z. B. der Firefox-Browser und das Fotoalbum Digikam zur Speicherung ihrer Daten das DBMS SQLite.

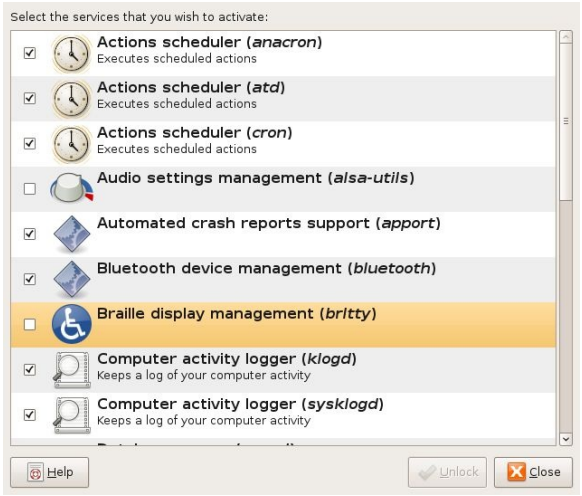


Abb. 2.5 Dienste unter Ubuntu-Linux

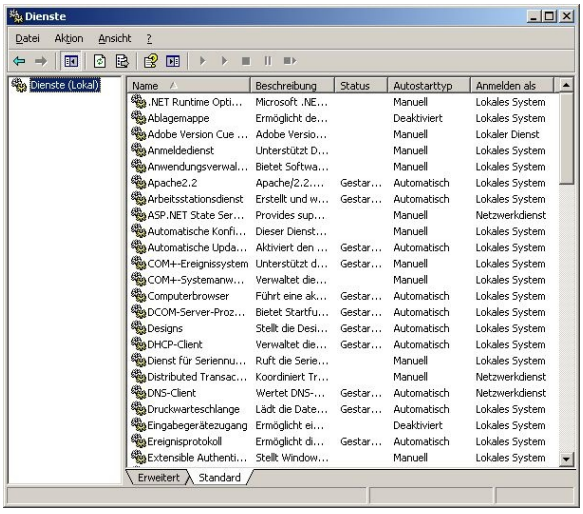


Abb. 2.6 Dienste unter Windows

Stellen Sie sich einen Dienst einfach wie einen guten englischen Butler vor. Er ist immer da, fällt nicht weiter auf und nimmt gerne jeden Auftrag entgegen.

Anwender: *Besorg' mir bitte eine Liste aller Kunden, die letzte Woche für mehr als 100 € bei uns eingekauft haben.*

Dienst: *Sehr wohl, Sir.*

2.9 Datenbank-Clients

Der Zugriff auf die Daten einer Datenbank erfolgt über einen Datenbank-Client, der sich mit dem DBMS verbindet. Ein Client kann ganz unterschiedlich aussehen. Im Grunde ist jede Webanwendung ein Datenbank-Client. Es ist dabei unerheblich, ob es sich um die öffentliche Ansicht für Kunden und Besucher oder um die interne Ansicht für Mitarbeiter und Administratoren handelt (siehe Screenshots Abb. 2.1 und Abb. 2.2).

Bei der Entwicklung einer Datenbank können Sie aber auch bereits fertige Clients verwenden, die es Ihnen als Entwickler erlauben, direkt mit der Datenbank zu kommunizieren. In diesem Kurs werden Sie den MySQL-Kommandozeilenclient verwenden, den Sie in Abb. 2.7 sehen.

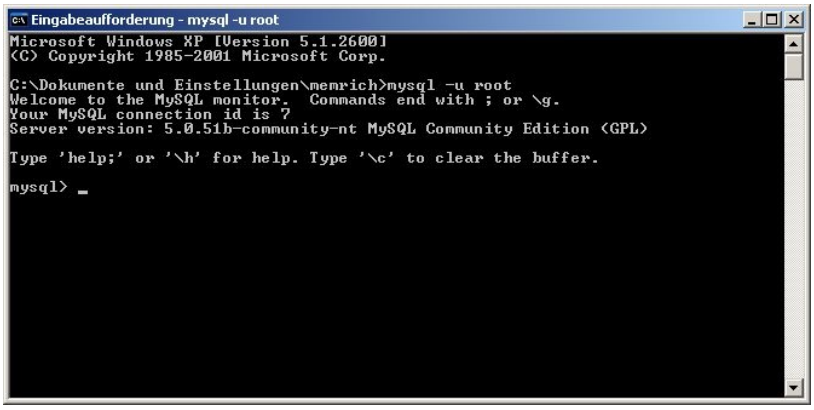


Abb. 2.7 Der MySQL-Kommandozeilen-Client

Zusammenfassung

- Relationale Datenbanken sind eine der größten Industrien im IT-Sektor.
- Viele Webanwendungen verwenden Datenbanken zur persistenten Speicherung – so können sie auf Bedarf dynamische Seiten generieren.
- Eine ***Datenbank (DB)*** ist eine Sammlung von Daten, denen ein Datenbankmodell zugrundeliegt.
- Ein ***Datenbank Management System (DBMS)*** ist ein Programm, das die Daten bzw. Datenbanken verwaltet.
- Ein ***Datenbank-Client*** ist ein Programm mit dem Sie auf die Daten einer Datenbank zugreifen können.
- Die meisten DBMS im Web arbeiten nach dem ***Client-Server-Prinzip***.

2.10 Aufgaben zur Selbstkontrolle

Testen Sie Ihr Wissen

1. Wozu werden Datenbanken benötigt?
2. Nennen Sie drei **Arten von Webanwendungen**, die Datenbanken verwenden!
3. Geben Sie drei **konkrete Beispiele für Webanwendungen**, die Datenbanken verwenden!
4. Nennen Sie drei **Arten von Webkomponenten**, die Datenbanken verwenden!
5. Unterscheiden Sie die Begriffe *Datenbank (DB)* und *Datenbank Management System (DBMS)*.
6. Was ist *MySQL*?
7. Nennen Sie mindestens 3 relationale Datenbank Management Systeme! Geben Sie jeweils an, ob es sich um Open Source oder kommerzielle Software handelt.

3 Datenbanken in der Praxis

In dieser Lektion lernen Sie:

- welche Programme Sie zur Datenbankentwicklung benötigen.
- wie Sie Datenbanken erstellen, auflisten und löschen.
- wie Sie in BNF jemanden an einen Löwen verfüttern.

3.1 Einrichten einer Entwicklungsumgebung

Damit Sie Datenbanken erstellen können, müssen Sie zumindest ein DBMS und einen Datenbank-Client installieren. In diesem Buch setzen Sie das freie[1] DBMS *MySQL* ein. *MySQL* ist mit über 4 Millionen aktiven Installationen[2] das am weitesten verbreitete Open-Source DBMS. Es ist eine kostengünstige und effektive Lösung für kleine und mittlere Projekte und wird oft sogar von großen Organisationen wie z. B. *Yahoo!*, *Slashdot* oder der *NASA* eingesetzt.

---

1 MySQL ist sowohl unter der OpenSource-Lizenz *GPL* als auch kommerziell erhältlich.
2 laut www.mysql.com – Stand: 07.01.2004

Abb. 3.1 XAMPP Logo

Eine vollständige Entwicklungsumgebung besteht, neben dem DBMS, aus weiteren Paketen wie z. B. Webserver, Programmiersprachen, Editor und Datenbank-Client. Die Installation und Konfiguration all dieser Werkzeuge, Sprachen etc. ist nicht immer ganz einfach. Deswegen gibt es ***Distributionen***, die eine Auswahl an Paketen bündeln und somit sowohl eine einfache Installation, als auch eine bereits angepasste Konfiguration bieten.


Abb. 3.2 MAMP Logo

Ich empfehle Ihnen, eine der kostenlosen Distributionen ***XAMPP*** (für Windows) oder ***MAMP*** (für Mac OS X) zu verwenden. Beide Distributionen enthalten unter anderem folgende Pakete:

- *MySQL* (DBMS)
- *Apache* (Webserver)
- *PHP* (Programmiersprache)

Unter www.apachefriends.org/en/xampp.html erhalten Sie *XAMPP* und die dazugehörige Installationsanleitung – *MAMP* entsprechend unter www.mamp.info.

Sollten Sie unter einem Debian-basierten Linux – wie z. B. Ubuntu – arbeiten, installieren Sie MySQL mittels: `apt-get install mysql-server mysql-client`.

Achtung

XAMPP und *MAMP* sind Distributionen, die ausschließlich für die Entwicklung verwendet werden sollten. In Ihrer Basiskonfiguration erfüllen Sie nicht die Sicherheitsanforderungen, die notwendig sind, um einen Produktivbetrieb (insbesondere im Internet) zu gewährleisten.

Für den produktiven Betrieb von Webanwendungen sind meist Linuxserver im Einsatz.

Es empfiehlt sich, im Produktivbetrieb eine Linuxinstallation mit den entsprechenden Paketen zu verwenden, die vorher sorgfältig manuell konfiguriert und auf Sicherheitsaspekte getestet wurde. Wenn Sie später Anwendungen bei einem Webhoster betreiben, wird dieser das für Sie übernehmen.

Aufgabe 1: MySQL-Installation

- Installieren Sie nun MySQL für Ihr Betriebsstem.

3.2 Starten und Stoppen von MySQL

unter Windows und MAC OS X

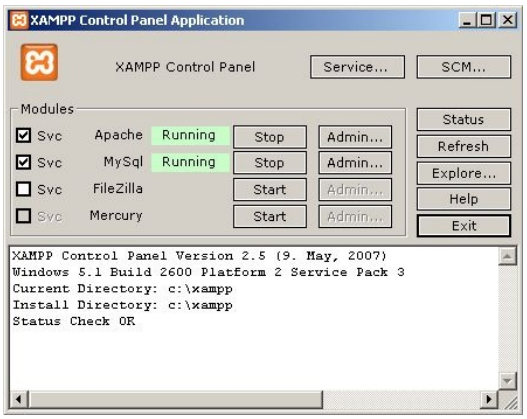


Abb. 3.3 Das XAMPP Control-Panel

Abb. 3.4 Das MAMP Control-Panel

Sowohl MAMP als auch XAMPP enthalten einen Control Panel. Mit diesem können Sie den MySQL-Server jederzeit starten und stoppen.

unter Linux

Unter Linux können Sie MySQL mit dem Befehl

```
/etc/init.d/mysql start
```

starten und mit

```
/etc/init.d/mysql stop
```

wieder anhalten.

Aufgabe 2: MySQL-Server starten und stoppen

Probieren Sie nun das Starten und Stoppen aus! Achten Sie darauf, dass der Server nach der Übung wieder gestartet ist!

3.3 SQL

Innerhalb des DBMS gibt es mehrere Datenbanken. Wenn Sie wissen möchten, welche Datenbanken es bereits gibt, können Sie das DBMS (oder genauer den DB-Server) einfach fragen. Sie können eine Frage nicht auf deutsch stellen. Verwenden Sie die Sprache, die der Server versteht. Praktischerweise verstehen fast alle modernen relationalen DBMS die Sprache *SQL*. Dazu benötigen Sie einen Client, der die Anfrage stellen kann. Später werden Sie Ihre eigenen Clients in Form von Webanwendungen programmieren. Im einfachsten Fall reicht uns zum Testen und Üben aber der von MySQL mitgelieferte Kommandozeilenclient. Damit Sie diesen Client von überall aus aufrufen können, müssen Sie ihn noch in den Pfad der ausführbaren Dateien eintragen.

- **Linux/MySQL:** geschieht automatisch. Sie müssen nichts tun.
- **MAC OS X/MAMP:** Fügen Sie den Pfad [1] `/Applications/MAMP/Library/bin/mysql` zu den Systempfaden hinzu – siehe www.systemfeld.de/2011/11/29/setzen-der-path-umgebungsvariable-unter-mac-os-x
- **Windows/XAMPP:** Drücken Sie win+Pause um die Systemeigenschaften zu öffnen. Wählen Sie den Reiter *Erweitert* bzw. *erweiterte Systemeinstellungen* und klicken Sie dort auf *Umgebungsvariablen*. Bearbeiten Sie die Variable `PATH` und ergänzen Sie `;C:\xampp\mysql\bin`.

---

1 documentation.mamp.info/de/mamp/howtos/mysql-ueber-die-komandozeile-nutzen

Aufgabe 3: MySQL-Client starten

Starten Sie jetzt den Kommandozeilenclient:

1. Öffnen Sie eine ***Kommandzeile*** in ein ***Konsolen***- bzw. ***Terminal-Fenster***.

   Windows: Entweder per *Start* > *Programme* > *Zubehör* > *Eingabeaufforderung* oder durch Drücken der Taste win+R (run) und Eingabe von `cmd`.

   MAC OS X: Öffnen Sie *Spotlight* mit der Tastenkombination cmd-Space. Geben Sie *Terminal* ein.

   Linux: Unter Ubuntu dient die Tastenkombination ctrl-alt-T als Shortcut.

2. Starten Sie den MySQL-Client in der Konsole mittels `mysql -u root`.

`mysql` ist das Programm. Die Angabe `-u root` bedeutet, dass es unter dem Benutzer (user) *root* gestartet wird. Dieser Benutzer besitzt alle Rechte und ist bereits angelegt. Normalerweise benötigen Sie noch ein Passwort. Nach der Installation ist das Passwort aber standardmäßig deaktiviert. Da Sie die Datenbank im Moment nur lokal zur Entwicklung einsetzen, stellt das kein Problem da. Für den Produktivbetrieb später müssen Sie selbstverständlich ein Passwort festlegen.

Sie sollten nun den MySQL-Prompt `mysql>` sehen, der es Ihnen erlaubt, SQL-Befehle einzugeben.

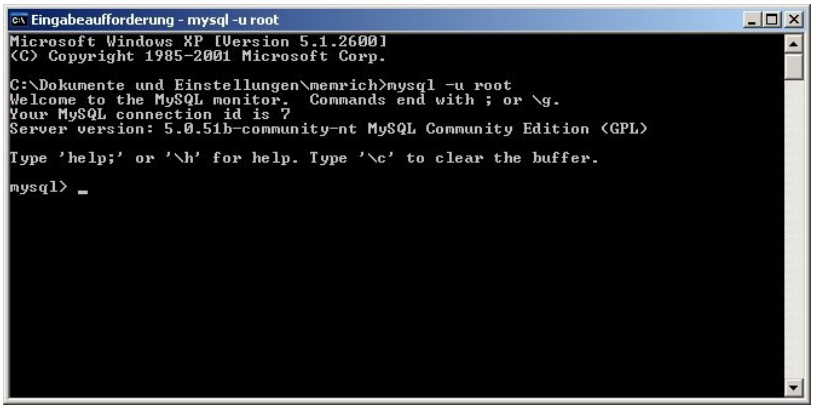


Abb. 3.5 MySQL-Kommandozeilen-Client

MySQL liefert bereits einige Datenbanken mit. Welche das sind, können Sie mit einem SQL-Befehl herausfinden.

Aufgabe 4: Vorhandene Datenbanken auflisten

Geben Sie Ihren ersten SQL-Befehl ein:

```
SHOW DATABASES;
```

Achten Sie insbesondere darauf, dass Sie den Befehl mit einen Semikolon »;« abschließen. Sie sollten folgende Liste erhalten:

```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| cdcol              |
| mysql              |
| phpmyadmin         |
| test               |
| webauth            |
+--------------------+
6 rows in set (0.00 sec)

mysql> _
```

Abb. 3.1 Ausgabe von `show databases`

Falls Sie eine Datenbank mehr oder weniger als in Abb. 3.1 sehen, ist das nicht weiter schlimm. Die mitgelieferten Datenbanken können sich je nach MySQL-Version und Distribution (MAMP, XAMPP, …) geringfügig unterscheiden.

In Zukunft werde ich Ihnen übrigens bei MySQL-Ausgaben statt eines Screenshots lediglich die reine Ausgabe zeigen – immer direkt nach der dazugehörigen SQL-Anweisung. Der Informationsgehalt ist der gleiche, die Ausgabe in Codeform ist aber besser lesbar. Die obige Ausgabe lässt sich also auch so darstellen:

```
SHOW DATABASES;
```

```
+--------------------+
| Database           |
+--------------------+
| information_schema |
| cdcol              |
| mysql              |
```

```
| phpmyadmin         |
| seminarverwaltung  |
| test               |
| webauth            |
+--------------------+
7 rows in set (0.00 sec)
```

Ihr erstes Beispielprojekt ist eine *Seminarverwaltung* für eine private Akademie – mehr dazu erfahren Sie in der nächsten Lektion. Für diese Seminarverwaltung benötigen Sie eine eigene Datenbank.

Aufgabe 5: Datenbank erstellen

Legen Sie die Datenbank *seminarverwaltung* mit folgendem SQL-Befehl an:

```
CREATE DATABASE seminarverwaltung;
```

```
Query OK, 1 row affected (0.01 sec)
```

Ein SQL-Befehl wird auch als ***Anfrage*** (engl. ***Query***) bezeichnet. Sie erhalten die Bestätigung `Query OK` immer dann, wenn der Befehl etwas verändert. Die Zeit in Klammern `(0.01 sec)` sagt Ihnen, dass das Anlegen der neuen Datenbank insgesamt gerade mal eine hundertstel Sekunde gedauert hat.

Um zu prüfen, ob es wirklich funktioniert hat, können Sie sich auch nochmals die vorhandenen Datenbanken anzeigen lassen.

Aufgabe 6: Datenbankerstellung prüfen

Geben Sie die vorhandenen Datenbanken aus:

```
SHOW DATABASES;
```

```
+--------------------+
| Database           |
+--------------------+
| information_schema |
| cdcol              |
| mysql              |
| phpmyadmin         |
| seminarverwaltung  |
| test               |
| webauth            |
+--------------------+
7 rows in set (0.00 sec)
```

Sie sollten sehen, dass Ihre neue Datenbank *seminarverwaltung* nun dabei ist.

Bei dieser Gelegenheit können Sie noch die überflüssige Datenbank *cdcol* löschen. *cdcol* ist eine Beispieldatenbank. Sie benötigen sie nicht weiter.

Aufgabe 7: Datenbank Löschen

1. Löschen Sie die Datenbank *cdcol* mit folgenden Befehl:

   ```
   DROP DATABASE cdcol;
   ```

   ```
   Query OK, 0 rows affected (0.00 sec)
   ```

2. Ein erneutes Auflisten der Datenbanken sollte zeigen, dass die *cdcol* tatsächlich gelöscht wurde.

```
SHOW DATABASES;
```

```
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| phpmyadmin         |
| seminarverwaltung  |
| test               |
| webauth            |
+--------------------+
6 rows in set (0.00 sec)
```

3.4 Was ist nun eigentlich SQL?

Kurz gesagt ist SQL eine sogenannte ***Abfragesprache***. Sie wird von vielen DBMS verwendet, um mit einer Anwendung (d. h. einem Client) zu kommunizieren. Abfragesprache bedeutet, dass Sie damit das DBMS ansteuern können. Sie können Daten in jede Richtung übertragen oder Befehle zur Verwaltung der Daten absenden. Auch Konfigurationseinstellungen lassen sich per SQL übermitteln.

Heute wird SQL als Abkürzung für ***Structured Query Language*** gedeutet. Ursprünglich kommt der Begriff aber von ***SEQUEL*** (Structured English Query Language bzw. Nachfolger). Die Buchstaben dazwischen wurden wegen einer Markenrechtsverletzung gestrichen.

SQL ist aber keine Programmiersprache – Sie können keine vollständigen Anwendungen in SQL schreiben. Dazu müssen Sie einen Client in einer anderen Sprache programmieren (wie z. B. PHP oder Ruby). Von dieser Sprache aus greifen Sie dann mit Hilfe von eingebundenem SQL auf das DBMS zu.

Es gibt einen SQL-Standard, der die Sprache grundsätzlich beschreibt. Kein DBMS hält sich aber genau an den Standard. Stattdessen verwendet jedes DBMS seinen eigenen Dialekt. Der lässt dann meist große Teile des Standards außer acht, bietet dafür aber viele eigene Erweiterungen. Dies gilt insbesondere auch für MySQL.

3.5 BNF

Um SQL-Befehle in einer allgemeinen Form angeben zu können, verwende ich die sogenannte ***BNF***. BNF steht für ***Backus-Naur-Form***, nach den Namen der beiden Entwickler. Es handelt sich dabei um eine sogenannte ***Metasprache***, mit der sich andere Sprachen, wie z. B. Programmiersprachen beschreiben und definieren lassen. Sie lernen im Laufe dieses Buches immer mal wieder neue BNF Symbole kennen. Von der BNF gibt es viele verschiedene Versionen. Einen guten Überblick finden Sie in Braun (2007)[1]. Dieses Buch verwendet die BNF-Version von MySQL. Dadurch können Sie jederzeit das offizielle MySQL-Handbuch (MySQL AB, 2008)[2] lesen und müssen keine abweichende Syntax lernen. Die MySQL-spezifische Version ist angelehnt an den ISO/IEC 14077-Standard der BNF, der auch im SQL-Standard Verwendung findet.

---

1 www-cgi.uni-regensburg.de/~brf09510/grammartypes.html
2 dev.mysql.com/doc

In der einfachsten Form lässt sich ein Befehl durch zwei Konzepte beschreiben – durch ***Schlüsselwörter*** und andere fest-definierte Anweisungen auf der einen Seite und ***Platzhaltern*** auf der anderen. Schlüsselwörter und andere fest-definierte Anweisungen müssen Sie exakt notieren. Für Platzhalter dürfen Sie eigene Werte einsetzen. Zeichen, die Sie direkt angeben, nennt die BNF ***Terminale***. Das bedeutet, sie sind »zu Ende«, im Sinne von »sie werden nicht weiter ersetzt«. Die Platzhalter, die Sie später ersetzen, nennt die BNF dementsprechend ***Nichtterminale*** oder ***Variablen***. Zur Unterscheidung setzen Sie einfach Groß- und Kleinschreibung ein. Terminale schreiben Sie groß, Nichtterminale dagegen klein.

Beispiel

Betrachten Sie etwa folgenden BNF Ausdruck:

```
ICH WERDE DICH DEN gefrässige_tierart ZUM FRAß VORWERFEN.
```

Der folgende BNF-Ausdruck etwa könnte wahlweise von einem wütenden Römer oder einem wütenden Bauern stammen. Es kommt eben darauf an, ob Sie für `gefrässige_tierart` Löwen oder Schweine einsetzen. `gefrässige_tierart` ist hier ein Nichtterminal, alle anderen Wörter dieses Ausdrucks sind Terminale. Es lassen sich beliebig viele Nichtterminale in einem Ausdruck verwenden, wie z. B. in folgendem:

```
HEUTE ABEND GEHE ICH ZU MEINEM FREUND name_eines_freundes NACH ort AUF DIE
FEIER, ANLÄSSLICH SEINES freudiges_ereignis UND WERDE ZIEMLICH VIEL getränk
TRINKEN.
```

Beachten Sie, dass die Variablen ein Underscore »_« verwenden, um einzelne Wörter zu verbinden. Sie können kein Leerzeichen verwenden, da das Wort sonst als zwei Variablen gelesen wird. Schließlich könnten zwei Variablen direkt aufeinander folgen.

Beispiel

```
ZU MEINEM GEBURTSTAG WÜNSCHE ICH MIR EIN farbe fahrzeugtyp.
```

Diese Variablen können Sie z. B. mit *rotes Fahrrad* oder *blaues Auto* ausprägen.

Datenbanken zeigen, anlegen und löschen in BNF

Da Sie die BNF nun kennen, kann ich Ihnen auch die BNF-Form der drei SQL-Befehle zeigen, die Sie schon verwendet haben.

Bestehende Datenbanken anzeigen:

```
SHOW DATABASES;
```

Hier gibt es keinen Unterschied zum SQL, da Sie nur Schlüsselwörter verwenden.

Eine Datenbank anlegen:

```
CREATE DATABASE db_name;
```

Eine Datenbank löschen:

```
DROP DATABASE db_name;
```

3.6 Konfiguration

Um von Anfang an durchgängig Internationalisierung zu ermöglichen, empfehle ich Ihnen, MySQL auf die Zeichenkodierung *UTF-8* umzustellen. Außerdem sollten Sie den SQL-Modus umstellen, damit sich MySQL möglichst konform zum SQL-Standard verhält

Aufgabe 8:

1. **Windows/XAMPP, Linux:** Öffnen Sie die MySQL-Konfigurationsdatei in einem Texteditor. Sie finden diese Datei entweder unter `C:\xampp\mysql\bin\my.ini` (XAMPP), oder unter `/etc/mysql/my.cnf` (Linux).

   Mac OS X/MAMP: Legen Sie die Datei `my.cnf` unter `/Applications/MAMP/conf` neu an.

2. Tragen Sie folgende Konfigurationen ein:

   ```
   [mysqld]
   character-set-server=utf8
   sql-mode="TRADITIONAL"
   ```

3. Starten und Stoppen Sie den MySQL-Server.

Zusammenfassung

- MAMP und XAMPP helfen Ihnen dabei, schnell eine funktionierende Entwicklungsumgebung einzurichten.
- Verwenden Sie SQL, um mit dem DBMS zu kommunizieren.
- Die BNF ist eine Metasprache, die die Syntax eines SQL-Befehls darstellen kann. Lesen Sie BNF-Ausdrücke, wenn Sie neue SQL-Befehle verstehen möchten!

BNF

Konzept	Darstellung
Terminale	Wort in Großbuchstaben
Nichtterminale (Variablen)	Wort in Kleinbuchstaben

Tabelle 3.1 Bisherige BNF-Konzepte

SQL

Zweck	BNF
Datenbank anlegen	`CREATE DATABASE db_name`
Datenbank löschen	`DROP DATABASE db_name`

Datenbanken auflisten	`SHOW DATABASES`

Tabelle 3.2 Bisheriges SQL

3.7 Aufgaben zur Selbstkontrolle

Testen Sie Ihr Wissen

1. Was ist SQL?
2. Was ist eine Distribution?
3. Aus welchen Teilen besteht XAMPP?
4. Was sind Terminale und Nichtterminale?
5. Was hat MySQL mit SQL zu tun?

Übungen

Aufgabe 9: BNF ausprägen

Entwickeln Sie jeweils zwei Ausprägungen der folgenden BNF-Ausdrücke:

- farbe IST MEINE LIEBLINGSFARBE.
- IN MEINEM NÄCHSTEN JOB WERDE ICH MINDESTENS gehalts_betrag währung VERDIENEN.

Aufgabe 10: Datenbanken Löschen und Anlegen

1. Zeigen Sie erneut alle Datenbanken an.
2. Löschen Sie die Datenbank *test*.
3. Legen Sie folgende neuen Datenbanken an:
    - fluggesellschaft
    - filmverleih
    - partnervermittlung
4. Prüfen Sie, ob das Anlegen der neuen Datenbanken gelungen ist.

4 Datenbankdesign: Das Domänenmodell

In dieser Lektion lernen Sie:

- wie Sie Kundenanforderungen in ein Domänenmodell übertragen.
- die Modellierungssprache UML kennen.
- wie Sie Klassendiagramme zeichnen.

Das Schöne an Standards ist, dass es so viele gibt, aus denen man wählen kann.

Andrew S. Tanenbaum

Willkommen zu Ihrem ersten Projekt. Natürlich ist es Ihr erklärtes Ziel, mit der Entwicklung von Software genug Geld zu verdienen, damit Sie sich bald möglichst in die Karibik absetzen können. Dazu brauchen Sie aber Kunden – und diese sollten Sie optimal zufrieden stellen. Dann kommen weitere, die auch wieder Geld mitbringen.

Die folgenden Lektionen werden Sie anhand eines kleines Projektes durcharbeiten. Das Projekt verwendet ein so genanntes iterativ-inkrementelles Vorgehen, wie es in agilen Entwicklungmethoden üblich ist. Eine ausführliche Erläuterung dazu finden Sie im Anhang.

4.1 Ein Kundengespräch

Stellen Sie sich vor, Ihr erster Kunde ist eine private Akademie, die ihre Kurse im Internet anbieten möchte. Mit der aktuellen Website des Unternehmens ist das zwar möglich, die Pflege der statischen Seiten ist jedoch sehr aufwändig und fehleranfällig. Da die neue Site datenbank-basiert sein soll, hat sich die Akademie an einen Spezialisten gewandt: An Sie!

Sie treffen sich also mit einer Mitarbeiterin der Akademie, die ihre Situation wie folgt schildert:

Wir benötigen eine Anwendung, um unsere Kurse zu verwalten. Unsere Kunden sollen auf der Website jeweils Titel, Beschreibung und Preis der Seminare einsehen. Unsere Mitarbeiter müssen neue Seminare anlegen, löschen und editieren. Natürlich benötigen wir auch eine Übersicht, die alle Seminare anzeigt. Ach ja, und eine Suchfunktion wäre ganz nützlich, es sind recht viele Seminare.

Sie legt Ihnen eine Liste von Beispieldaten vor:

- Ruby on Rails
  Beschreibung: Ruby on Rails ist das neue, sensationelle OpenSource-Framework, das auf…
  Preis: 1900,00 €

- Ajax & DOM
  Beschreibung: Ajax ist längst dem Hype-Stadium entwachsen. Mehr und mehr professionelle…
  Preis: 1699,99 €

- JavaScript
  Beschreibung: JavaScript ist eine Programmiersprache mit vielseitigen …
  Preis: 2500,00 €

- Datenbanken & SQL
  Beschreibung: Nahezu alle modernen Webapplikationen speichern ihren Datenbestand …
  Preis: 975,00 €

Sie können die Anforderungen jetzt aufnehmen. In welcher Form Sie das tun, ist weniger entscheidend und hängt vom jeweiligen Vorgehensmodell ab. Alle agilen Vorgehensmodelle sind sich aber darin einig, dass menschliche Kommunikation essentiell ist (Beck K. et al, 2001). Geben Sie sich also nicht mit einer reinen textuellen Beschreibung zufrieden, sondern verlangen Sie immer ein persönliches Gespräch mit einem Mitarbeiter, der die Anforderungen kennt! Die Gefahr von Missverständnissen bei

sogenannten Lasten- und Pflichtenheften oder anderen textuellen Beschreibungen sind einfach zu hoch, wie viele gescheiterte Projekte beweisen[1].

---

1 siehe Standish Group (1994) und de.wikipedia.org/wiki/Chaos-Studie

Die geschilderten Anforderungen bilden die erste Iteration. Bevor die Kundin Sie nun noch stundenlang über weitere geplante Features informiert[1] , bieten Sie ihr an, diesen Kern an Funktionen erst einmal umzusetzen. Bei einem weiteren Treffen kann sie sich das Ergebnis anschauen und entscheiden, was ihre Firma als nächstes benötigt. Damit vermeiden Sie das ***BDUF*** -Problem (***Big Design Up Front***[2]), bei dem am Anfang zu viel Zeit in das Design von Anwendung und Datenbank gesteckt wird – nur um dann später festzustellen, dass der Kunde eigentlich doch etwas anderes möchte.

---

1 …und glauben Sie mir, die Dame kann reden ohne Luft zu holen!
2 zu deutsch: (zu-)viel Design am Anfang

Es ist möglich, dass Sie Ihre Kundin missverstehen – das ist leider durchaus nicht ungewöhnlich, sonder liegt am Wesen der menschlichen Kommunikation. Vielleicht ändert sie auch einfach ihre Meinung. Änderungen am Markt oder ein wachsendes Verständnis der Bedürfnisse der Anwender führen oft zu Änderungswünschen an den Anforderungen – aber das ist auch gut so[1]!

---

1 zumindest für den Anwender. Dem Entwickler kann es unter Umständen schlaflose Nächte bereiten.

In einem realen Projekt würden Sie nun die Anforderungen erheben und dokumentieren. Das ist aber abhängig vom Vorgehensmodell und würde den Rahmen dieses Buches sprengen.[1]. Stattdessen gehen Sie gleich zur Planung über.

---

1 Eine gute Übersicht, über agile ***Erhebungstechniken*** für Anforderungen (sogenanntes ***Requirements Engineering***) finden Sie bei Hruschka, Rupp & Starke (2004).

Beginnen Sie nicht sofort mit der Programmierung, sondern legen Sie sich zunächst eine sinnvolle Struktur zurecht. Damit sparen Sie Zeit und vermeiden das typische »*Ich sehe den Wald vor lauter Bäumen nicht mehr*«-Problem. Nur sollten Sie es auch nicht übertreiben, indem Sie tagelang Ihre Planung perfektionieren, ohne sich an den Code heranzutrauen.

Die Goldlöckchen-Lösung

Eines der Kernprobleme der Softwareentwicklung ist, dass es viele gute Techniken und Prinzipien gibt, die aber durch *Zuviel des Guten*, schnell zu schlechten Techniken und Prinzipien werden. Mit ein wenig Zeit und Erfahrung finden Sie in allem den richtigen Mittelweg. Eine sehr interessante Darstellung des Problems ist das Märchen *Die drei Bären*[1]. In diesem Märchen versucht ein Mädchen, in verschiedenen Situationen immer einen optimalen Kompromiss zu finden. Wegen der gold-gelockte Haare des Mädchens hat sich in der Softwareentwicklung der Begriff *Goldlöckchen-Lösung* (***Goldilocks-Solution***)[2] geprägt.

---

1 de.wikipedia.org/wiki/Goldl%C3%B6ckchen_und_die_drei_B%C3%A4ren
2 siehe www.c2.com/cgi/wiki?GoldilocksSolution

Zuerst kostete sie aus der größten Schüssel. »Uh«, sagte sie, “das ist viel zu heiß!«, und spuckte den Brei einfach wieder aus. Dann versuchte sie es mit der mittelgroßen Schüssel. »Uh«, schrie sie, »das ist viel zu kalt«. […]

Schließlich kostete Goldlöckchen aus der kleinsten Schüssel. Da sagte sie nichts mehr, denn sie war zu beschäftigt damit, alles aufzuessen. Der Brei war nämlich genau richtig.[1]

---

1 Zitat aus www.hekaya.de/txt.hx/goldloeckchen-und-die-drei-baeren—maerchen—southey_1

Abb. 4.1 Aus dem Projekt Gutenberg E-Book, *English Fairy Tales*[1]

Im Umfeld der Softwareentwicklung gibt es eine Unmenge an verschiedenen Analyse-, Modellierungs- und Designtechniken, einhergehend mit einer genauso großen Unmenge an Diagrammtypen und Notationsformen. Statt nun massenhaft Techniken zu besprechen, zeige ich Ihnen nur einige wenige. Diese wenigen sind aber ausreichend, um eine moderne Webanwendung zu entwickeln.

4.2 Das Domänenmodell

Zunächst erstellen Sie ein sogenanntes ***Domänen-*** oder ***Konzept-Modell***. Dieses Modell ist noch Technologie-unabhängig und beschreibt die Kernkonzepte der Anwendung. Es heißt ***Domänenmodell***, da es die Sicht der Anwendungsdomäne (z. B. Seminare & Lehre, Büchershop, Addressverwaltung usw.) vertritt. Anders ausgedrückt: Es verwendet ausschließlich Begriffe aus dem Fachvokabular des Anwenders – nicht des Entwicklers!

In welcher Form lässt sich ein solches Modell darstellen? Üblicherweise ist ein Diagramm das Mittel der Wahl. Rein textuelle Beschreibungen sind aber durchaus auch möglich. Früher begannen Datenbankentwickler meist mit einem sogenannten ***Entity-Relationship-Diagramm*** (kurz ***ERD***). In der modernen Softwareentwicklung hat sich jedoch die Modellierungssprache ***UML*** durchgesetzt[1] . Falls Sie noch nicht mit der UML in Berührung gekommen sind, hier ein kurzer Abriss:

---

1 Auch heute arbeiten viele Firmen noch mit ERDs. Letztendlich ist es auch etwas Geschmacksache.

4.2.1 UML

UML steht für ***Unified Modeling Language***. Es handelt sich dabei um eine grafische Modellierungssprache. In der Vergangenheit gab es häufig das Problem, dass verschiedene Entwicklergruppen unterschiedliche Diagrammnotationen verwendeten[1] . Das erschwerte es einem Entwickler, die Diagramme anderer zu verstehen. Schließlich haben sich 1995 drei führende Methodiker [2] zusammengeschlossen, um eine einheitliche Notation hervorzubringen: die UML. Heute ist UML ein Industriestandard der *OMG* (*Object Management Group*). Sie besteht aus einer Reihe von Diagrammtypen. Aus der Menge dieser Diagrammtypen sind für Sie zunächst nur ***Klassendiagramme*** relevant.

---

1 z. B. eine der folgenden: OOSE, RDD, OMT, OBA, OODA, SOMA, MOSES, OPEN/OML. Viele Teams definierten gar ihre eigene Notation – was die geradezu babylonische Sprachverwirrung nur noch weiter förderte.
2 Die drei Amigos: James Rumbaugh, Ivar Jacobson und Grady Booch. Mehr zur Geschichte der UML finden Sie unter de.wikipedia.org/wiki/Unified_Modeling_Language.

4.2.2 Klassendiagramm

Sie erstellen nun das Domänenmodell der Seminarverwaltung in Form eines Klassendiagrammes. Beginnen Sie damit, dass Sie Basiskonzepte identifizieren und als Klasse darstellen. Eine Klasse stellt dabei eine Art Stellvertreter oder auch Blaupause für alle Ihre Exemplare dar. In diesem Fall möchten Sie verschiedene Exemplare von Seminaren verwalten. Somit lässt sich das Seminar als Klasse identifizieren. Andere Beispiele für Klassen in typischen Webanwendungen sind:

Beispiel

Nomen: Benutzer, Gruppen, Adressen, Forumseinträge

Es sollten jedenfalls immer Dinge – keine Tätigkeiten oder Eigenschaften – sein. Eine erster Check ist deswegen auch die Prüfung der Wortart. Normalerweise handelt es sich um Nomen. Verben und Adjektive sind als Klassen eher ungeeignet.

Beispiel

- **Verben**: lernen, teilnehmen, bearbeiten, löschen
- **Adjektive**: groß, blau, essbar, gefährlich, schön

Grundsätzlich sollten Sie im Klassendiagramm des Domänenmodells auch nur Klassen modellieren, die aus der realen Welt oder aus der Fachsprache des Anwenders stammen. Rein technische Klassen wie z. B. *Cache*, *Stack* oder *Fensterdialog* haben im Domänenmodell nichts verloren.

Da es sich also bei diesen Klassen im Domänenmodell um Dinge aus der realen Welt oder der Fachsprache handelt, werden sie als ***Entities*** bezeichnet.

Sie stellen eine Klasse in der UML grafisch dar, indem Sie sie als Rechteck zeichnen.

Abb. 4.1 Die Klasse *Seminar* im Domänenmodell

Um das Klassendiagramm detaillierter auszugestalten, versuchen Sie, Eigenschaften zu finden, die Sie dem Klassendiagramm zuordnen können. Schauen Sie sich nochmal die Anforderungen an.

> *Unsere Kunden sollen auf der Website jeweils Titel, Beschreibung und Preis der Seminare einsehen.*

Offensichtlich lassen sich einem Seminar die Eigenschaften *Titel*, *Beschreibung* und *Preis* zuordnen. Da diese Eigenschaften für alle Exemplare von Seminaren potentiell vorhanden sind, gehören sie in die Klasse. Die einzelnen Seminare können sich in den Werten für diese Eigenschaften durchaus unterscheiden, aber jedes Seminar verfügt prinzipiell über Titel, Beschreibung und Preis.

In UML erfolgt die Darstellung unterhalb des Namens der Klasse – in einem eigenen Bereich.

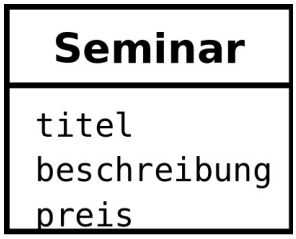


Abb. 4.2 Die Klasse *Seminar* mit Attributen im Domänenmodell

Angelehnt an objekt-orientierte Programmiersprachen bezeichnet die UML die Eigenschaften als ***Attribute***. Sie haben nun die Möglichkeit, das Diagramm um Verhalten zu erweitern – wiederum in einem eigenen Bereich.

Abb. 4.3 Die Klasse *Seminar* mit Attributen und Methoden im Domänenmodell

Die Tätigkeiten, die Sie auf der Klasse ausführen können, bezeichnet die UML als ***Methoden***. Die Klammern nach jeder Methode dienen als zusätzliche Kennzeichnung und sind der Syntax objekt-orientierter Programmiersprachen entliehen. Bei Methoden handelt es sich grundsätzlich um Verben.

Es ist auch nicht immer zwingend erforderlich, alles anzugeben. Führen Sie nur soviele Details an, wie Sie für die nächsten Schritte benötigen[1]. Oftmals ist es nicht einmal notwendig, überhaupt Methoden anzugeben. Andererseits dient das Diagramm auch als Dokumentation und hilft späteren Wartungsprogrammierern, sich in Ihren Code und Ihre Datenstrukturen einzufinden. Mit zunehmender Erfahrung werden Sie sicherer bei der Entscheidung, wie viel und wie detailreich Sie modellieren müssen.

---

1 siehe ***Agile Model Driven Development*** (***AMDD***) unter www.agilemodeling.com/essays/amdd.htm.

Scott sagt…

> *Designmodelle müssen nur gerade so gut genug sein. Man muss nicht jedes einzelne Detail modelieren, sie müssen nicht perfekt sein, und ganz sicher müssen sie nicht vollständig sein. Erinnern Sie sich, wann Sie das letzte mal zu einer Designspezifikation programmiert hatten (falls überhaupt)? Haben Sie sich wirklich die ganzen Feinheiten angesehen? Nein, weil Sie kompetent genug waren, die Detail selbst auszurbeiten.*

Dr. Dobb's Magazin May/04: 15 Design Tips[1]

Das fertige Domänenmodell können Sie übrigens Ihrer Kundin zeigen. Es hat

insbesondere den Vorteil, dass es auch Nicht-Entwickler ohne Vorwissen verstehen können.

4.3 Richtlinien für Klassendiagramme

Über die Basisregeln von UML hinausgehend, haben sich bei Klassendiagrammen einige Richtlinien eingebürgert:

- Schreiben Sie Klassennamen groß und im Singular.
  Beispiel: Benutzer, Gruppe, Adresse, Forumseintrag
- Schreiben Sie Attributnamen klein.
  Beispiel: titel, beschreibung, preis
- Schreiben Sie Methodennamen klein.
  Beispiel: buche, kaufe, speichere

4.4 Mehrere Klassen

Sie haben die Seminarklasse gerade fertig modelliert, da ruft Ihre Kundin wieder an:

Hallo. Ich hoffe, ich störe Sie nicht. Mir ist noch etwas eingefallen. Wir benötigen noch ganz dringend die Möglichkeit, dass sich Benutzer in der Anwendung registrieren können. Dabei muss der Zeitpunkt der Registrierung erfasst werden und die Angabe von Vor- und Nachname ist Pflicht.

Anmelden können sie sich dann mit E-Mail-Adresse und Passwort.

In der Realität würden Sie diese Anforderungen zwar aufnehmen, jedoch noch nicht umsetzen. Im Sinne des agilen Vorgehens setzen Sie zuerst ein Feature komplett um, bevor Sie weitere analysieren, designen etc. Im Wesentlichen möchte ich Ihnen an dieser Stelle aber einfach nur zeigen, dass ein Diagramm normalerweise mehrere Klassen enthält.

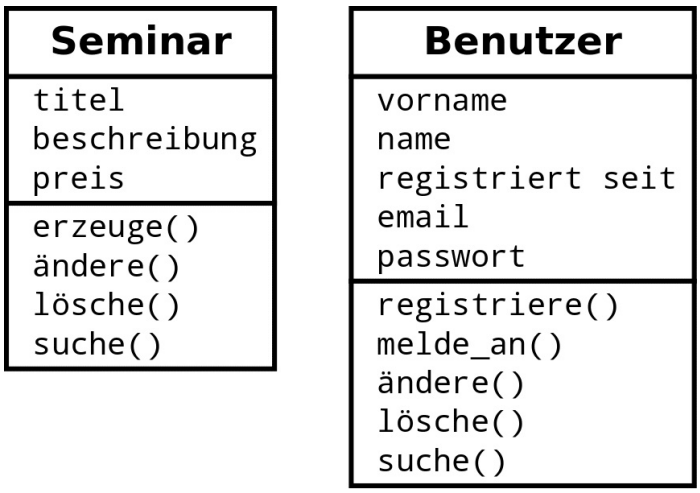


Abb. 4.4 Seminare und Benutzer im Domänenmodell

Zusammenfassung

- Reden Sie mit Ihrem Kunden! Verlassen Sie sich nicht nur auf schriftliche Spezifikationen.
- Ein Domänenmodell beschreibt die Kernkonzepte einer Anwendung aus Sicht des Anwenders - nicht des Entwicklers.
- UML ist eine grafische Modellierungssprache.
- In einem Klassendiagramm stellen Sie Klassen mit Methoden und Attributen dar.

4.5 Aufgaben zur Selbstkontrolle

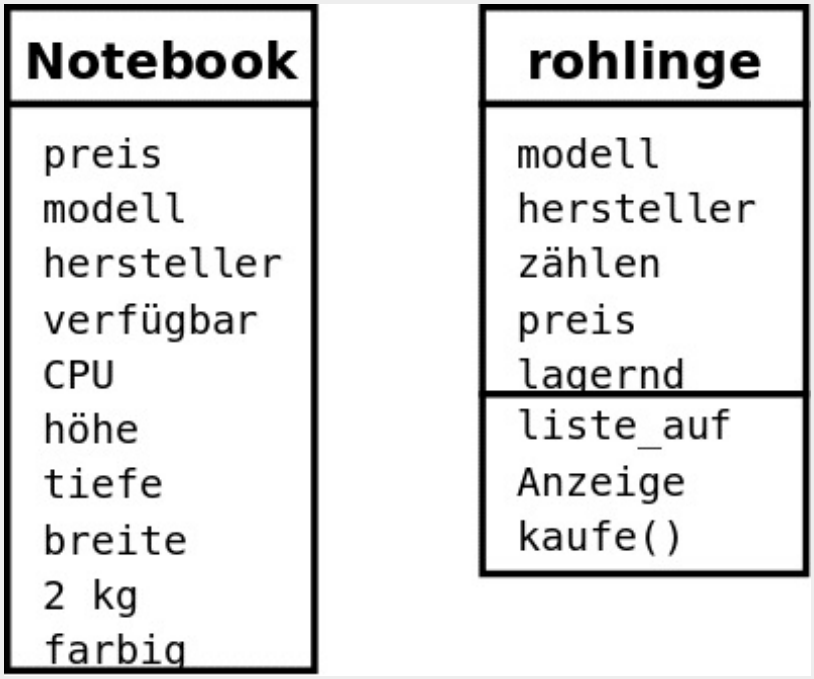
Testen Sie Ihr Wissen

1. Was ist eine Entität?
2. Welches Problem löst die UML?
3. Mit welchem Diagrammtyp begannen Datenbankentwickler die Modellierung, bevor die UML zur Verfügung stand?
4. Welche Hinweise können Sie verwenden, um Klassen, Attribute und Methoden des Domänenmodells aus einer Kundenanforderung zu extrahieren?

Übungen

Aufgabe 1: Finden Sie die Fehler

In folgendem Domänenmodell eines Online-Shops haben sich mehrere Fehler eingeschlichen. Finden Sie sie und kreisen Sie sie ein! Notieren Sie Verbesserungsvorschläge, falls möglich.



Aufgabe 2: Begriffe

Welche der folgenden Begriffe könnten Entities in einem Domänenmodell sein? Begründen Sie Ihre Wahl!

Begriff	Entity?	Begründung falls kein Entity
blau	nein	kein Nomen

Cache	nein	technischer Begriff	
Seminar	ja		
JavaScript-Programmierung			
Person			
Entity			
Unterrichten			
Organisation			
Teilnahme			
Rechnungstabelle			
Variable			
Veranstaltung			
Dienstschnittstelle			
Mahnung			
Cron-Job			

5 Einführung in Dia

In dieser Lektion lernen Sie:

- das Illustrationsprogramm Dia kennen.
- wie Sie mit Dia UML-Diagramme zeichnen.

5.1 Vorteile von UML-Werkzeugen

UML-Diagramme können Sie natürlich problemlos mit Stift & Papier zeichnen. Es hat allerdings auch Vorteile, ein spezielles Werkzeug einzusetzen:

- Sie haben Ihre Diagramme in elektronischer Form und können sie zu den anderen Projektdateien ablegen.
- Sie müssen Ihre Zeichnungen nicht erst einscannen, um Sie z. B. per E-Mail zu verschicken.
- Ein UML-Werkzeug kennt bereits die UML-Konzepte und erzwingt somit die Einhaltung des Standards.

Im Grunde spielt es keine große Rolle, welches UML-Werkzeug Sie einsetzen. Da Sie hier nur grundlegende Features benötigen, sind fast alle geeignet. Ein Werkzeug, das ich Ihnen empfehlen kann, ist *Dia*. Es handelt sich bei Dia um einen Illustrationsprogramm, das unter anderem auch Schablonen für UML besitzt. Dia ist open source. Sie können es für Windows, Linux und Mac OS X kostenlos unter dia-installer.de herunterladen. Unter Linux ist es in den meisten modernen Distributionen bereits enthalten. Es ist sicher nicht das umfangreichste Programm seiner Art. Es ist aber einfach zu bedienen und für unsere Zwecke völlig ausreichend.

5.2 Klassen in Dia

Dia verfügt über eine Werkzeugpalette und ein Zeichenfenster. Um ein UML-Diagramm zu erstellen, wählen Sie in der Werkzeugpalette zunächst das UML-Sheet aus.

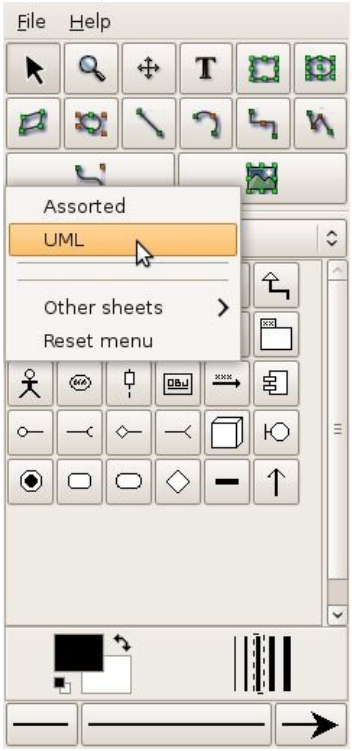


Abb. 5.1 Werkzeugpalette von Dia

Um eine Klasse zu erstellen, markieren Sie das Klassen-Icon.

Abb. 5.2 Dia Klassen-Icon

Klicken Sie dann ins Zeichenfenster, um die Klasse zu platzieren.

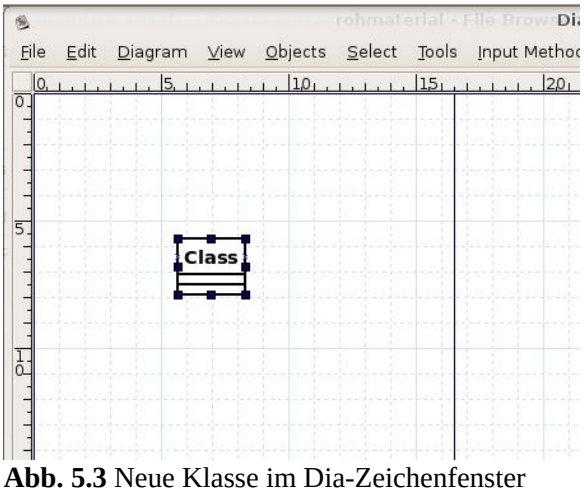


Abb. 5.3 Neue Klasse im Dia-Zeichenfenster

Durch einen Doppelklick auf die neue Klasse gelangen Sie in den Properties-Dialog der Klasse. Dort können Sie die Klasse benennen (*Class name*). Unter dem Reiter *Attributes* haben Sie die Möglichkeit, mit dem Button *New* neue Attribute hinzuzufügen und anschließend zu editieren. Der *Operations*-Reiter bietet das gleiche – nur für Methoden. Die vielen Zusatzeigenschaften können Sie im Moment noch außer Acht lassen.

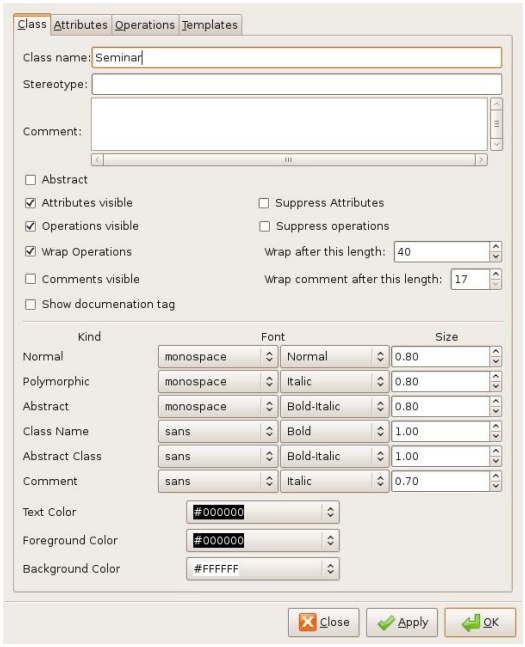


Abb. 5.4 Properties-Dialog einer Klasse in Dia

Auf diese Weise erhalten Sie im Handumdrehen UML-Diagramme, wie die in der letzten Lektion vorgestellten[1].

1 Ich habe für die Diagramme in diesem Buch nämlich auch Dia benutzt, aber verraten Sie es nicht weiter.

Zusammenfassung

- Es gibt viele Werkzeuge zur Erstellung von UML-Diagrammen.
- Dia ist ein einfaches opensource-Programm, mit dem Sie UML-Diagramme zeichnen können.
- Benutzen Sie in Dia die UML-Schablone, um standardkonforme UML-Diagramme zu erzeugen.

5.3 Aufgaben zur Selbstkontrolle

Übungen

Aufgabe 1: Projekt »Filmverleih«: Anforderungsanalyse

Ein Auftraggeber möchte, dass Sie einen webbasierten Filmverleih für ihn entwickeln. Er sieht für seine spezielle Zielgruppe einen Markt und fürchtet sich nicht vor der Konkurrenz. Der Verleih muss über Frontends für Kunden und Mitarbeiter verfügen.

Ihr Auftraggeber formuliert folgende Anforderung:

> *Meine Mitarbeiter sollen Filme verwalten. Filme bestehen aus Titel und Spieldauer. Letztere wird in Minuten angegeben. Manche Leute interessieren sich auch für das Erscheinungsjahr. Außerdem wollen wir später eine Internetsuche anbieten, die es erlaubt, anhand des Regisseurs (Name, Vorname und Geburtsdatum) zu suchen. Zum gefundenen Film muss dann eine Kurzbeschreibung angezeigt werden. Im Moment ist es noch nicht wichtig, welcher Film von welchem Regisseur stammt. Meine Mitarbeiter sollen aber schon mal eine Liste von Regisseuren erfassen können.*

Markieren Sie Entities und Attribute im Anforderungstext. Kreisen Sie dazu die Entities ein und unterstreichen Sie die Attribute.

Aufgabe 2: Projekt »Filmverleih«: Domänenmodell

Erarbeiten Sie ein Domänenmodell in UML zum Projekt *Filmverleih*.

Zusatzübungen

Aufgabe 3: Projekt »Fluggesellschaft«: Domänenmodell

Eine Fluggesellschaft beauftragt Sie, eine Inventardatenbank zu erstellen. Dabei sind alle vorhandenen Flugzeugtypen mit folgenden Daten zu erfassen:

- Model
- Hersteller
- Flugstunden (aller Piloten auf diesem Flugzeugtyp) – d. h. gewissermaßen die Erfahrung, die die Gesellschaft mit diesem Model gesammelt hat.
- Höchstgeschwindigkeit
- Erstflug
- typische Sitzanzahl

Außerdem möchten sie alle ihre Piloten erfassen mit:

- Name

- Vorname
- Geburtsdatum
- Flugstunden (Erfahrung)

Erarbeiten Sie ein Domänenmodell in UML. Zeichnen Sie es mit Dia.

Aufgabe 4: Projekt »Partnervermittlung«: Domänenmodell

Einer Ihrer Auftraggeber möchte eine Partnervermittlung gründen. Er verfügt über gute Kontakte und das entsprechende Startkapital, aber Ihm fehlt noch eine passende Webanwendung.

Seine erste Anforderung lautet:

Es müssen Personen mit folgenden Eigenschaften erfasst werden:

- Name, Vorname
- Geschlecht
- Haarfarbe
- Größe in cm
- Gewicht in kg
- Geburtsdatum

Erarbeiten Sie ein Domänenmodell in UML. Zeichnen Sie es mit Dia.

Aufgabe 5: Erweiterungen zum Filmverleih

Erweitern Sie das erstellte Domänenmodell des Filmverleihs um eigene Attribute, die Sie für sinnvoll oder gar notwendig erachten. In der Praxis würden Sie das natürlich mit Ihrem Auftraggeber absprechen.

Aufgabe 6: Erweiterung der Partnervermittlung

Schlagen Sie Ihrem Auftraggeber weitere Attribute vor und ergänzen Sie Ihr Design.

6 Datenbankdesign: Das physische Datenmodell

In dieser Lektion lernen Sie:

- welche Schritte sich an die Domänenmodellierung anschließen.
- wie Sie Entities in Tabellen überführen.
- die erste Transformationsregel kennen.

Der Kode ist eher dass, was man Richtlinien nennt, statt tatsächlicher Regeln

Captain Barbossa,
Pirates of the Caribbean

Sie haben ein Domänenmodell erstellt und es Ihrer Kundin vorgelegt. Nachdem Sie bestätigt hat, dass das Modell den Anforderungen gerecht wird, können Sie zum nächsten Schritt übergehen.

6.1 Physisches Datenmodell

Aus dem Domänenmodell können Sie zwei weitere Modelle ableiten[1]:

1 Statt direkt das physische Modell aus dem Domainenmodell abzuleiten, können Sie auch zuerst ein logisches Modell ableiten. Das ist aber meistens überflüssiger Aufwand (Ambler, 2006). In der klassischen Datenbankentwicklung wird eine Datenbank nicht für eine spezielle Anwendung benötigt, sondern die Datenbank entsteht schon zuvor unabhängig – und mit der Absicht, mehrere Anwendungen darauf aufzusetzen. In diesem Fall macht es Sinn, ein logisches Modell zu verwenden. Dieser Ansatz wird beispielsweise auch von Theorey, Lightstone & Nadeau (2006) verfolgt.

- Das Klassenmodell für die Programmierung (durch Anwendung von ***Objektmodellierung***)
- Das physische Datenmodell (PDM) (durch Anwendung von ***physischer Datenmodellierung***)

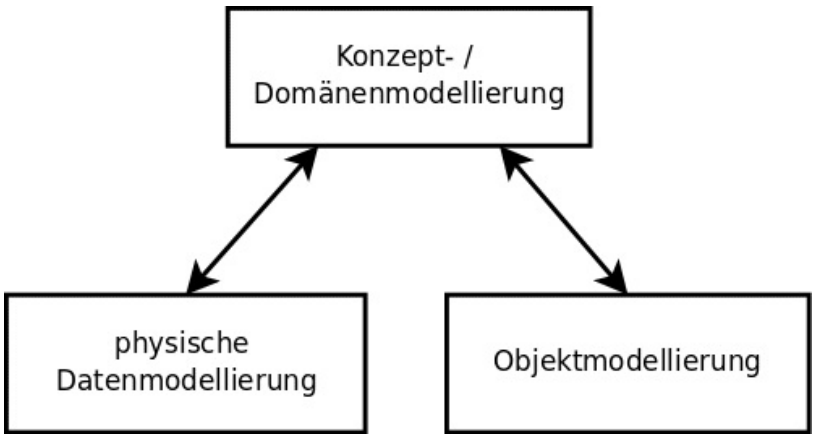


Abb. 6.1 Modelierungstätigkeiten nach Ambler (2003)

Das Diagramm zeigt die übliche Abfolge der Modellierungstätigkeiten. Beachten Sie, dass die Pfeile beidseitig gerichtet sind. Das bedeutet, dass Sie jederzeit zum vorherigen Modell zurückwechseln können, um Erweiterungen und Verbesserungen vorzunehmen.

Das Klassenmodell für die Programmierung unterscheidet sich vom Domänenmodell in Anzahl und Art der Klassen. Eine Klasse des Domänenmodells kann z. B. aus technischen Gründen in mehrere Einzelklassen zerfallen. Außerdem kommen Klassen dazu, die für das Konzept keine Relevanz haben – beispielsweise eine Cache-Klasse zur Performancesteigerung, oder Klassen, die für die Bedienung der Anwendung erforderlich sind. Da dieses Buch sich aber vornehmlich mit Datenbanken beschäftigt, ist dieses Modell hier nicht weiter wichtig. Wenden Sie sich stattdessen dem physischen Datenmodell zu.

Das physische Datenmodell beschreibt die Struktur der Daten in der Form, in der sie die Datenbank letztendlich speichert. Deswegen kann sich das Modell auch von DBMS zu DBMS unterscheiden. Es ist auch hier wieder möglich, auf das UML-Klassendiagramm zurückzugreifen. Das einzige Problem dabei ist, dass es für das physische Klassendiagramm keinen einheitlichen UML-Standard gibt. Deswegen empfehle ich Ihnen das UML-Profil von Scott Ambler (2006)[1]. Ein ***UML-Profil*** ist eine Erweiterung zum UML-Standard. Der Standard beschreibt bereits, wie solche Profile definiert werden.[2]

1 www.agiledata.org/essays/umlDataModelingProfile.html

2 Andere UML-Profile zur Datenbank-Modellierung finden Sie z. B. bei Muller (1999), Gornik (2002) und Sparks (1999).

Das erste physische Datenmodell unterscheidet sich nicht sonderlich vom Domänenmodell (siehe Abschnitt 6.2).

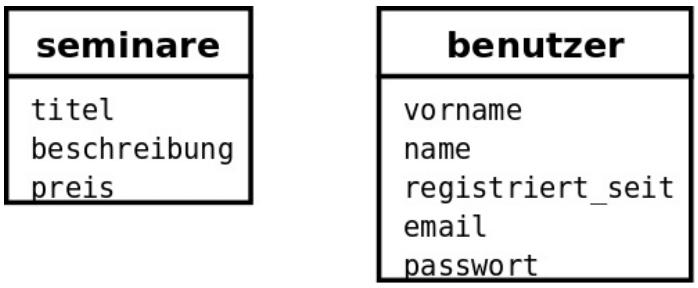


Abb. 6.2 Physisches Datenmodell der Seminarverwaltung (noch unvollständig)

Im Gegensatz zum Domänenmodell beschreibt dieses Diagramm kein Entity, sondern direkt eine physische Tabelle der realen Datenbank. Sie können sich die Tabelle *seminare* etwa so vorstellen:

titel	beschreibung	preis
…	…	…
…	…	…
…	…	…

Tabelle 6.1 Tabelle *seminare*

Das physische Datenmodell beschreibt dabei nur die Struktur der Daten. Über den späteren Inhalt der Tabellen, die Anzahl der Zeilen usw., trifft es keine Aussage.

6.2 Unterschiede zum Domänenmodell

Worin genau liegt nun der Unterschied zum Domänenmodell?

Zunächst einmal fällt auf, dass die Methoden fehlen. Das ist nicht weiter verwunderlich. In diesem Modell geht es – wie der Name schon sagt – ausschließlich um die Daten. Das Verhalten ist für das Entwickeln einer Datenbank unerheblich. Wichtig wird es während der Objektmodellierung.

Ein weiterer Unterschied ist der Pluralbegriff *seminare* statt dem Singular *Seminar*. Zudem ist das Wort *seminare* kleingeschrieben. Das liegt daran, dass das physische Modell bereits die Namen der Bezeichnungen aus der realen Datenbank verwendet. Bei *seminare* handelt es sich um eine ***Tabelle***. Für diese gelten (hier) folgende SQL-Programmierrichtlinien:

- Geben Sie Tabellennamen für Tabellen, die aus Entities entstanden sind, im Plural an.
- Schreiben Sie alle Bezeichner (z. B. Tabellen- und Attributnamen) klein.
- Trennen Sie Begriffe, die aus mehreren Wörtern bestehen, mit einem Underscore »_« (z. B. registriert_seit).
- Vermeiden Sie Umlaute (ö, ü, ä) und Sonderzeichen (z. B. ?, !, &).

Beispiel

- **Tabellenbezeichner:** artikel, personen, veranstaltungen, gefoerderte_weiterbildungen
- **Attributbezeichner:** name, vorname, geburtsdatum, start_termin

Für das PDM sind die genannten Einschränkungen notwendig, da sich diese Bezeichner später im SQL-Code wiederfinden. Das Domänenmodell dagegen ist ausschließlich für menschliche Konsumenten gedacht. Deswegen können Sie dort problemlos auf Umlaute, Sonderzeichen und Leerzeichen zurückgreifen.

Sobald Sie umfangreichere und komplexere Datenbanken designen, nehmen die Unterschiede zum Domänenmodell weiter zu. Generell ist es aber sinnvoll, die Unterschiede möglichst zu minimieren. Das erleichtert das Zusammenspiel von Datenbank und Anwendung.

Um es nochmal deutlich zu sagen:

Das ***Domänenmodell*** beschreibt die Konzepte einer Anwendung aus der Sicht eines Anwenders oder Fachexperten (Domänenexperten). Eine nicht anders deklarierte Klasse im UML-Diagramm eines Domänenmodells ist ein **Entity**.

Das ***physische Datenmodell*** ist eine direkte Abbildung der realen Datenbank. Aus dem UML-Diagramm, das das physische Datenmodell zeigt, kann mit Hilfe von SQL eine entsprechende Datenbank angelegt werden. Eine nicht anders deklarierte Klasse im UML-Diagramm eines physischen Datenmodells ist eine **Tabelle**.

6.3 Vom Domänenmodell zum physischen Datenmodell

Um ein Domänenmodell in ein physisches Datenmodell zu überführen, sind mehrere Transformationen erforderlich. Da Sie es momentan noch mit einzelnen, nicht miteinander in Beziehung stehenden Tabellen zu tun haben, gestaltet sich die Transformation erfreulich einfach.

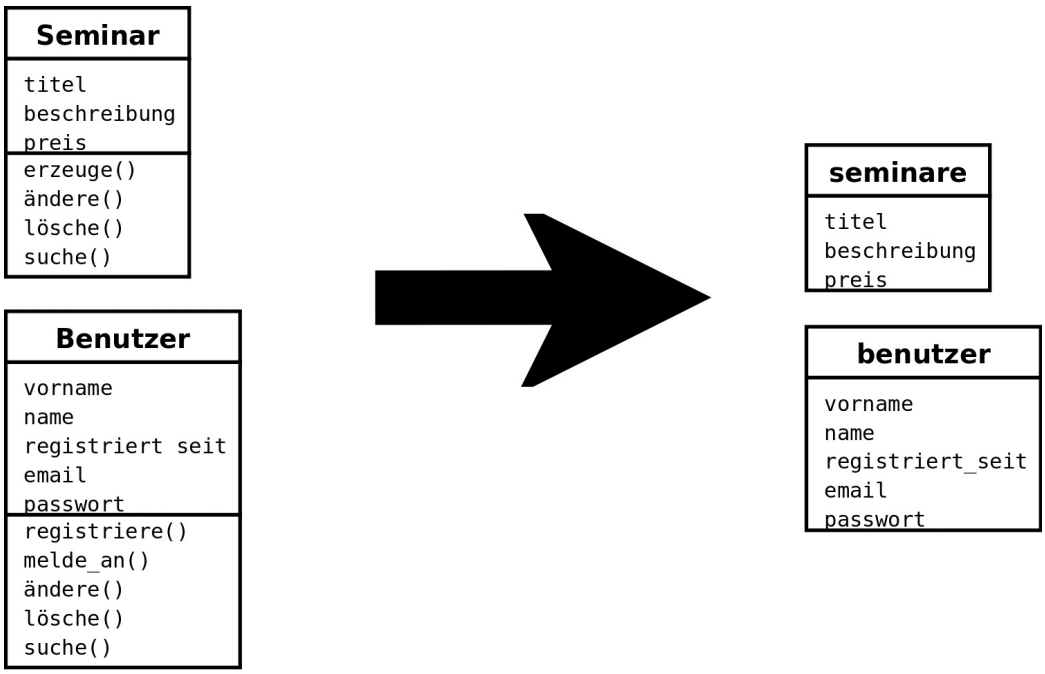


Abb. 6.3 Transformation vom Domänenmodell zum physischen Datenmodell

Transformation von Entities

- Legen Sie zu jedem Entity aus dem Domänenmodell eine Tabelle im physischen Datenmodell an. Sie können Name und Attribute der Tabelle aus dem Entity übernehmen. Beachten Sie dabei die SQL-Programmierrichtlinien.

Sie werden bald weitere Transformationsregeln kennen lernen. Dabei wachsen auch die Unterschiede zwischen den Modellen. Sie werden außerdem noch sehen, dass sich diese Transformationsregeln nicht immer stur umsetzen lassen. Betrachten Sie sie eher als Richtlinien statt als verbindliche Regeln. Es gibt viele Fälle, wo bei der Transformation Ihr Verstand[1] gefragt ist. Modellierung[2] ist ein kreativer Prozess!

---

1 Arbeiten Sie immer im Brain-On-Mode :)
2 sowohl Domänenmodellierung als auch physische Datenmodellierung

Das physische Datenbankmodell der Seminarverwaltung ist auch noch unvollständig. Sie erfahren bald, was noch fehlt. Zunächst einmal ist es wichtig, das zugrundeliegende Modell zu verstehen. Dem physischen Datenmodell liegt nämlich das relationale Modell zugrunde. In der nächsten Lektion erfahren Sie, was es damit auf sich hat.

Zusammenfassung

- Auf die Domänenmodellierung folgt die Objektmodellierung und die physische Datenmodellierung.
- Das Domänenmodell beschreibt die Konzepte einer Anwendung aus Sicht eines Anwenders oder Fachexperten.
- Eine nicht anders deklarierte Klasse im UML-Diagramm eines Domänenmodells ist ein Entity.
- Das physische Datenmodell ist eine direkte Abbildung der realen Datenbank.
- Eine nicht anders deklarierte Klasse im UML-Diagramm eines physischen Datenmodells ist eine Tabelle.
- Es gibt Transformationsregeln, die beschreiben, wie ein physisches Datenmodell aus einem Domänenmodells abgeleitet werden kann. Diese sind jedoch nicht immer anwendbar.

6.4 Aufgaben zur Selbstkontrolle

Testen Sie Ihr Wissen

1. Nennen Sie mindestens zwei Unterschiede zwischen dem Domänenmodell und dem physischen Datenmodell.
2. Was ist ein UML-Profil?
3. Nennen Sie mindestens zwei SQL-Programmierrichtlinien.

Übungen

Aufgabe 1: Physisches Modell des Filmverleihs

Zeichnen Sie ein physisches Modell des Filmverleihs (siehe Übungen von Lektion 5).

Zusatzübungen

Aufgabe 2: Physisches Modell der Partnervermittlung

Zeichnen Sie ein physisches Modell der Partnervermittlung (siehe Übungen von Lektion 5).

Aufgabe 3: Physisches Modell der Fluggesellschaft

Zeichnen Sie ein physisches Modell der Fluggesellschaft (siehe Übungen von Lektion 5).

7 Das Relationale Modell

In dieser Lektion lernen Sie:

- das berühmte relationale Modell kennen.
- aus welchen Teilen das Modell besteht.
- wie Sie die Mengenschreibweise verwenden.

Wer die Praxis übt, ohne sich vorher mit der Theorie beschäftigt zu haben, gleicht einem Steuermann, der sein Schiff ohne Kompass und Steuer besteigt und nun nicht weiss, wohin er fährt.

Leonardo da Vinci[1]

7.1 Was ist das relationale Modell?

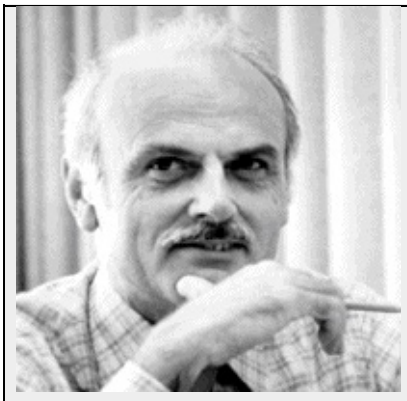
Was ist nun eigentlich das ***relationale Modell***[1], von dem alle reden?

1 Das relationale Modell wird manchmal auch relationales Datenbankmodell oder relationales Datenmodell (RDM) genannt. Alle Begriffe meinen das gleiche. Ich halte mich hier an den Originalbegriff von Codd/Date, siehe z. B. Date (2007).

Vorab muss ich gestehen, dass ich gar nicht vor habe, Ihnen alle theoretischen Aspekte des relationalen Modells vollständig zu erklären. Viele Aspekte sind für die praktische Webentwicklung irrelevant und sind sicherlich auch nicht im Sinne eines Einführungsbuches. Falls es Sie dennoch brennend interessiert, so empfehle ich Ihnen Date (2005) zu lesen. Er kann es Ihnen ohnehin besser erklären als ich.

Ich werde Ihnen aber nun die Grundlagen näher bringen, die Sie für Ihre tägliche Arbeit benötigen.

Das relationale Modell beschreibt die Theorie, die allen relationalen Datenbanken zugrunde liegt. Codd hat das Modell aus der ***Prädikatenlogik*** abgeleitet – einem Teilgebiet der Mathematik. Da in der Webentwicklung relationale Datenbanken eine wichtige Rolle spielen, ist ein gutes Verständnis des Modells unerlässlich – insbesondere auch für die tägliche Praxis.



Ted sagt…

> *Die Adaption des relationalen Modells von Daten […], erlaubt die Entwicklung einer universellen Datensubsprache basierened auf einem angewandten Predikatenkalkühl.*

7.2 Aspekte des relationalen Modells

Das relationale Modell besteht aus 3 Aspekten:

- Struktur
- Integrität
- Manipulation

Zunächst einmal beschreibt das Modell eine Möglichkeit, Daten zu strukturieren bzw. zu organisieren. Im Gegensatz zu anderen Modellen dienen sogenannte ***Relationen***[1] und ***Relationsvariablen*** als wichtigstes Strukturelement – daher der Name.

---

1 Der Einfachheit halber unterscheide ich hier nicht zwischen Relationen und Relationsvariablen. Genaueres können Sie bei Date (2005) nachlesen. Eine Tabelle in SQL entspricht übrigens eher einer Relationsvariablen, da sich die Inhalte der Tabelle ändern können.

7.3 Relationen

Was ist eine Relation? Am einfachsten ist es, wenn Sie sich eine Relation wie eine ***Tabelle*** vorstellen. In einer Tabelle können Sie die Seminare und ihre Attribute unterbringen.

titel	beschreibung	preis
Ruby on Rails	Ruby on Rails ist das neue, sensationelle OpenSource-Framework, das auf der modernen…	1900,00
Ajax & DOM	Ajax ist längst dem Hype-Stadium entwachsen. Mehr und mehr professionelle Websites folgen …	1699,99
JavaScript	JavaScript ist eine Programmiersprache mit vielseitigen Einsatzmöglichkeiten. Sie gilt…	2500,00
Datenbanken & SQL	Nahezu alle modernen Webapplikationen speichern ihren Datenbestand …	975,00

Tabelle 7.1 Tabelle *seminare*

Die Relation **ist nicht wirklich** eine Tabelle. Aber sie lässt sich auf dem Papier als eine Tabelle darstellen. Im Folgenden verwende ich der Einfachheit halber meistens den Begriff *Tabelle* – auch wenn es sich genau genommen um eine Relation handelt.

Jede ***Zeile*** (***row***) der Tabelle beschreibt ein Seminar mit den dazugehörigen Informationen. Eine solche Zeile wird im Sprachgebrauch des relationalen Modells ***Tupel*** genannt. Eine andere geläufige Bezeichnung aus dem Datenbankjargon ist ***Datensatz*** (***record***). Eine ***Spalte*** (***column***) in der Tabelle nennt sich ***Attribut***. Das deckt sich praktischerweise mit der Bezeichnung aus UML. Die Attribute dieser Relation sind also *titel*, *beschreibung* und *preis*. Jedes Tupel hat an jeder Stelle eines Attributes einen ***Wert***. Dabei ist z. B. ***JavaScript*** ebenso ein Wert wie ***975,00***. Das Behältnis des Wertes nennt sich ***Zelle***.

Hier sehen Sie nochmals die Zuordnung der Begriffe:

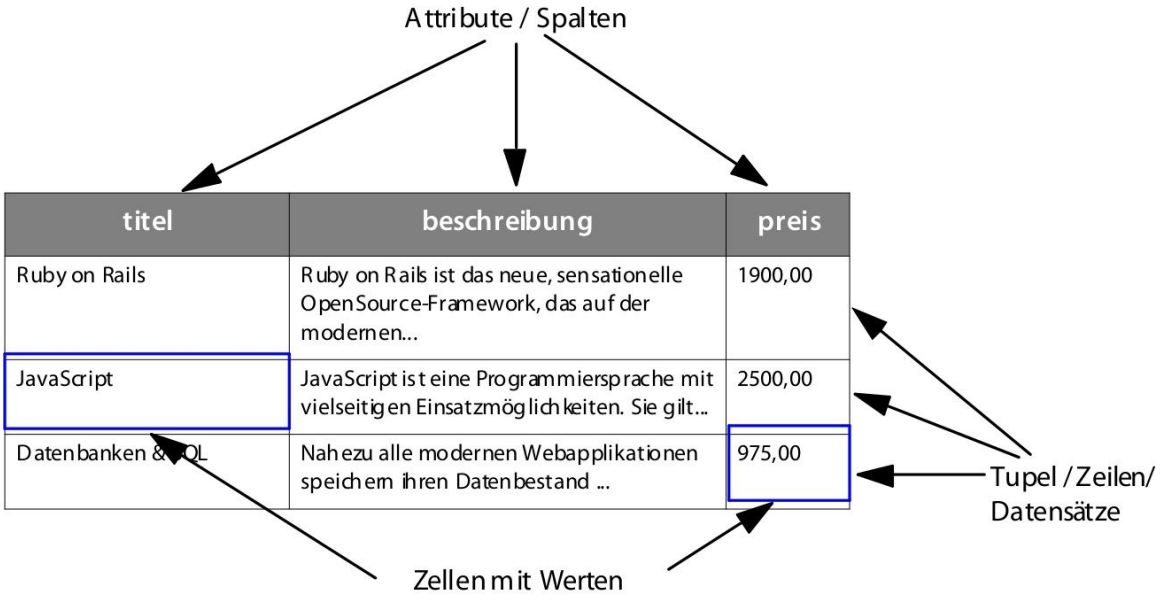


Abb. 7.1 Begriffe einer Tabelle/Relation

Relationes Modell	Darstellung als Tabelle
Attribut	Spalte
Tupel	Zeile
Relation bzw. Relationsvariable	Tabelle
Wert	Inhalt einer Tabellenzelle

Tabelle 7.2 Gegenüberstellung der Begriffe von Relation und Tabelle

Nun kann ich Ihnen auch etwas genauer[1] erklären, was eine Relation eigentlich ist. Eine Relation besteht aus einem ***Heading*** (***Überschrift***) und einem ***Body*** (***Körper***). Die Überschrift ist dabei eine ***Menge*** von Attributen. Der Körper ist eine Menge von Tupeln, die der Überschrift entsprechen, d. h. für jedes Attribut einen entsprechenden Wert aufweisen. Ein Heading, das im Code vorliegt, wird übrigens auch als ***Tabellenschema*** bezeichnet. Die Menge aller Tabellenschemata einer Datenbank wird ***Datenbankschema*** genannt.

---

1 wenn auch nicht formal exakt

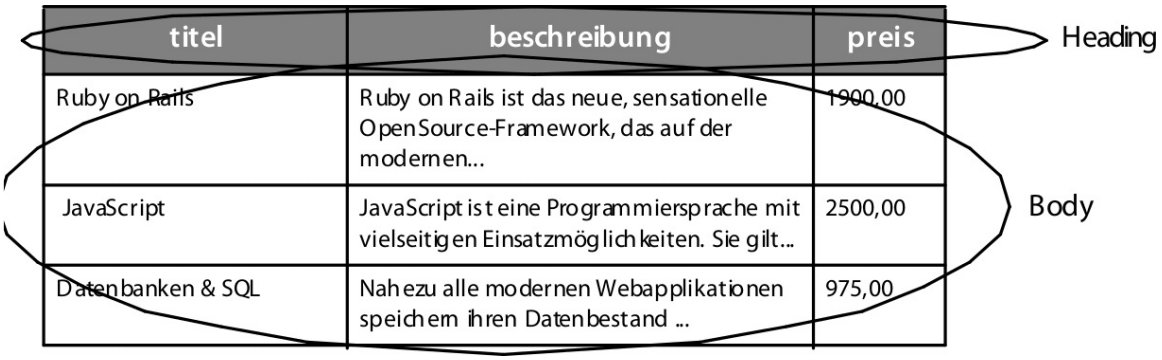


Abb. 7.2 Heading und Body einer Relation in Tabellendarstellung

7.4 Was war nochmal eine Menge?

Hier eine kleine Auffrischung, falls Ihnen der Begriff *Menge* etwas suspekt erscheint. Menge beschreibt ein grundlegendes Konzept aus der Mathematik. Sie ist nach Georg Cantor eine »Zusammenfassung bestimmter, wohlunterschiedener Objekte«[1]. Stellen Sie sich einfach einen Sack vor, in dem mehrere Dinge liegen. Die Dinge in der Menge sind praktisch beliebig. Sie können sowohl abstrakt als auch konkret sein. Es kann sich z. B. um Zahlen handeln, oder um Farben, um Menschen[2], um Autos, usw.

---

1 Mehr zum Mengenbegriff finden Sie unter de.wikipedia.org/wiki/Menge.
2 Zugegeben brauchen Sie dann einen ziemlich großen Sack und für die Insassen könnte es sehr ungemütlich werden.

7.5 Mengenschreibweise

Um eine Menge anzugeben, verwenden Sie die ***Mengenschreibweise***. Diese bedient sich der geschweiften Klammern. Sie können z. B. die Menge der Farben *Rot*, *Grün* und *Blau* schreiben als:

- Farben = {Rot, Grün, Blau}.

Im Fall unserer Seminare bedeutet das, dass die Menge `{titel, beschreibung, preis}` das Heading der Relation `seminare` darstellt. Oftmals ist es für uns Datenbankentwickler nicht weiter wichtig, welche Inhalte eine bestimmte Relation hat. Deswegen müssen Sie nicht immer eine aufwendige Zeichnung anfertigen. Stattdessen notieren Sie die Relation in Mengenschreibweise. Verwenden Sie dazu die Form:

- *name_der_relation = menge_der_attribute*

Die Relation *seminare* schreiben Sie entsprechend als

- seminare = {title, beschreibung, preis}

Streng genommen ist das nicht korrekt. Bei der Menge handelt es sich nur um das Heading und nicht um die ganze Tabelle. Diese Vereinfachung erlaubt es Ihnen aber, sich auf die Struktur zu konzentrieren und nicht jedesmal Werte angeben zu müssen.

7.6 Konsequenzen des Relationsbegriffs

Da Sie sich nun schon etwas mehr unter einer Relation vorstellen können, kann ich Ihnen einige wichtige Konsequenzen aufzeigen.

> Die Reihenfolge der Attribute spielt keine Rolle.

In einer Menge – im mathematischen Sinn – ist die Reihenfolge der Elemente unerheblich. Auf Tabellen übertragen, bedeutet das, dass Sie Attribute vertauschen können und es handelt sich immer noch um die gleiche Relation. Beispielsweise sind die beiden Ausdrücke

- seminare = {titel, beschreibung, preis}
- seminare = {preis, titel, beschreibung}

im Sinne des relationalen Modells völlig equivalent. Natürlich sehen Sie in der grafischen Darstellung als Tabelle einen Unterschied. Dennoch handelt es sich um die gleiche Relation.

titel	preis	beschreibung
Ruby on Rails	1900,00	Ruby on Rails ist das neue, sensationelle OpenSource-Framework, das auf der modernen…
Ajax & DOM	1699,99	Ajax ist längst dem Hype-Stadium entwachsen. Mehr und mehr professionelle Websites folgen …
JavaScript	2500,00	JavaScript ist eine Programmiersprache mit vielseitigen Einsatzmöglichkeiten. Sie gilt…
Datenbanken & SQL	975,00	Nahezu alle modernen Webapplikationen speichern ihren Datenbestand …

Tabelle 7.3 Tabelle *seminare* mit vertauschten Attributen

> Die Reihenfolge der Tupel spielt keine Rolle.

Der Körper einer Relation besteht – wie schon gesagt – aus einer Menge von Tupeln. Da es sich um eine Menge handelt, können Sie auch Tupel vertauschen, ohne die Relation zu verändern.

titel	beschreibung	preis
Datenbanken & SQL	Nahezu alle modernen Webapplikationen speichern ihren Datenbestand …	975,00
Ruby on Rails	Ruby on Rails ist das neue, sensationelle OpenSource-Framework, das auf der modernen…	1900,00
Ajax & DOM	Ajax ist längst dem Hype-Stadium entwachsen. Mehr und mehr professionelle Websites folgen …	1699,99

JavaScript	JavaScript ist eine Programmiersprache mit vielseitigen Einsatzmöglichkeiten. Sie gilt…	2500,00

Tabelle 7.4 Tabelle *seminare* mit vertauschten Tupeln

> Es darf keine zwei verschiedenen Attribute mit gleichem Namen geben.

Da der Name eines Attributs eine Spalte eindeutig kennzeichnet und damit die Werte in den jeweiligen Tupeln, würden zwei Attribute mit gleichem Namen schnell zu Widersprüchen führen. Außerdem darf in einer Menge jedes Element nur einmal vorhanden sein. Die Tabelle

- seminare = {titel, beschreibung, preis, preis}

ist deshalb ungültig. In der folgenden Darstellung sehen Sie das Problem:

titel	beschreibung	preis	preis
Ruby on Rails	Ruby on Rails ist das neue, sensationelle OpenSource-Framework, das auf der modernen…	1900,00	1900,00
Ajax & DOM	Ajax ist längst dem Hype-Stadium entwachsen. Mehr und mehr professionelle Websites folgen …	1699,99	1699,99
JavaScript	JavaScript ist eine Programmiersprache mit vielseitigen Einsatzmöglichkeiten. Sie gilt…	**1500,00**	**2500,00**
Datenbanken & SQL	Nahezu alle modernen Webapplikationen speichern ihren Datenbestand …	975,00	975,00

Tabelle 7.5 Tabelle *seminare* mit doppeltem Attribut

Wie viel kostet das Seminar *JavaScript*? 1500,00 € oder 2500,00 €? Die Tabelle ist widersprüchlich. Das mag hier offensichtlich sein. Denken Sie aber mal an folgende Anforderung:

Für die Benutzer müssten wir auch mehrere E-Mail-Adressen erfassen können.

Eine Lösungsidee könnte die Tabelle

- benutzer = {name, vorname, **email**, **email**, passwort, registriert_seit}

darstellen. Wie Sie aber gerade erfahren haben, ist das nicht zulässig. Abgesehen davon müssten Sie schon vorher wissen, wie viele E-Mail-Adressen Sie pro Benutzer speichern müssen. Falls die Anzahl feststeht und Sie die E-Mail-Adressen semantisch unterscheiden können – z. B. in private und berufliche – gibt es durchaus eine einfache Lösung.

- benutzer = {name, vorname, registriert_seit, **private_email**, **berufliche_email**, passwort}

Für den Fall, dass aber beliebig viele E-Mail-Adressen zu speichern sind und/oder es keine Unterscheidungskriterien gibt, können Sie das Problem jetzt noch nicht lösen. Ich werde Ihnen später einen Ausweg verraten. Bis dahin müssen Sie sich noch ein wenig gedulden.

> Es kann keine zwei verschiedene Tupel geben, die für alle Attribute die gleichen Werte aufweisen.

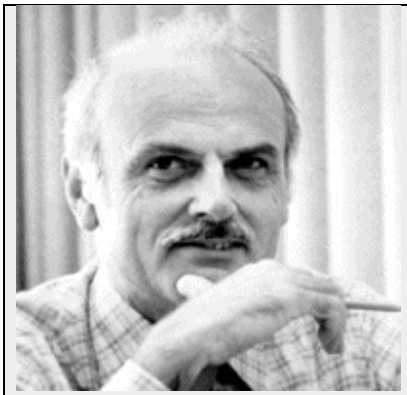
In der folgenden Tabelle ist der erste Datensatz doppelt vorhanden. Da es sich um eine Menge von Tupeln handelt, ist kein doppeltes Element erlaubt. Es macht auch wenig Sinn, da der Datensatz komplett redundant ist und keine Zusatzinformationen liefert. Er bläht die Datenbank nur unnötig auf und verbraucht zusätzlichen Speicherplatz[1].

---

1 Genau genommen ist das nur einer von vielen Gründen – weitere finden Sie unter www.dbdebunk.com/page/page/638922.htm.

titel	beschreibung	preis
Ruby on Rails	Ruby on Rails ist das neue, sensationelle OpenSource-Framework, das auf der modernen…	1900,00
Ajax & DOM	**Ajax ist längst dem Hype-Stadium entwachsen. Mehr und mehr professionelle Websites folgen …**	**1699,99**
Ajax & DOM	**Ajax ist längst dem Hype-Stadium entwachsen. Mehr und mehr professionelle Websites folgen …**	**1699,99**
JavaScript	JavaScript ist eine Programmiersprache mit vielseitigen Einsatzmöglichkeiten. Sie gilt..	1500,00
Datenbanken & SQL	Nahezu alle modernen Webapplikationen speichern ihren Datenbestand …	975,00

Tabelle 7.6 Tabelle *seminare* mit doppeltem Datensatz



Ted sagt…

Wenn irgendetwas war ist, wird es nicht wahrer dadurch, dass man es

zweimal sagt.

Zusammenfassung

- Das relationale Modell ist von der Prädikatenlogik abgeleitet.
- Relationen lassen sich als Tabellen darstellen.
- Relationen lassen sich in Mengenschreibweise notieren.
- Die Reihenfolge der Attribute spielt keine Rolle.
- Die Reihenfolge der Tupel spielt keine Rolle.
- Es kann keine zwei verschiedenen Tupel geben, die für alle Attribute die gleichen Werte aufweisen.

7.7 Aufgaben zur Selbstkontrolle

Testen Sie Ihr Wissen

1. Wie werden Daten in einem relationalen DBMS organisiert?
2. Erläutern Sie die folgenden Begriffe in Ihren eigenen Worten: Menge, Attribut, Tupel, Relation, Wert, Tabelle, Heading, Körper

Übungen

Aufgabe 1: Fehlersuche

Markieren Sie alle Fehler in folgender Tabelle:

name	vorname	telefon_nr	telefon_nr
Meier	Michael	0911/12345	0171/12345
Schulze	Stefanie	0911/12345	0171/1234
Becker	Andreas	0911/12345	0171/1234
Meier	Michael	0911/12345	0171/12345

Tabelle 7.1 kunden

Aufgabe 2: Identische Headings

Welche der folgenden Mengen sind gleich?

1. autos1 = {modell, hersteller, leistung_in_ps}
2. autos2 = {modell, hersteller}
3. autos3 = {hersteller, leistung_in_ps, modell}
4. autos4 = {modell, hersteller, leistung_in_kw}
5. buecher1 = {title, autor, erscheinungsdatum, verlag}
6. buecher2 = {title, autoren, erscheinungsdatum, verlag}
7. buecher3 = {erscheinungsdatum, verlag, title, autoren}
8. buecher4 = {erscheinungsdatum, verlag, title, autor}

Aufgabe 3: Mengenschreibweise in Tabellen umwandeln

Zeichnen Sie zu den folgenden Mengen Tabellen mit je 3 beispielhaften Datensätzen.

- autos = {modell, hersteller, leistung_in_ps, von_0_auf_100_in_sec}
- buecher = {titel, autor, erscheinungsdatum, verlag, seitenzahl}

Zusatzübungen

Aufgabe 4: Mengenschreibweise nach UML

Überführen Sie folgende Tabellen (Mengenschreibweise) in ein physisches Datenmodell in UML.

- autos = {modell, hersteller, leistung_in_ps, von_0_auf_100_in_sec}
- buecher = {title, autor, inhaltsangabe, verlag, seitenzahl}

8 Datentypen

In dieser Lektion lernen Sie:

- die wichtigsten Datentypen kennen.
- welche Typen MySQL verwendet.
- wie Sie besagte Typen im physischen Datenmodell in UML kennzeichnen.

8.1 Datentypen

Ein wichtiges Konzept fehlt noch, bevor Sie den Bereich der theoretischen Grundlagen wieder verlassen. In Abschnitt 2.5 haben Sie Anforderungen kennengelernt, die ein DBMS erfüllen sollte. Eine der Anforderungen ist *Integrität*. Damit eine Datenbank keine völlig unsinnigen Werte speichert, gibt es bereits eine erste Hürde: den ***Datentyp*** (kurz ***Typ***).

Beispielsweise ist nicht sinnvoll, für den Preis eines Seminars einen Wert wie *Meier* zu hinterlegen. Sollte das passieren, handelt es sich mit Sicherheit um einen Eingabe- oder Programmierfehler. Glücklicherweise können Sie dieses Problem mit Hilfe von Typen vermeiden.

Christopher sagt…

> *Was genau ist ein Typ? Grundsätzlich es ist eine benannte unendliche Menge von Werten.*

Ein Typ ist eine benannte endliche Menge von Werten. Der Typ *DECIMAL* beispielsweise ist die Menge aller rationalen Zahlen[1] - d. h. vereinfacht ausgedrückt, Zahlen mit Nachkommastellen. Da jeder Wert einen Typ hat und von diesem beschränkt wird, können Sie z. B. für einen DECIMAL die Zahl *975,00* einsetzen – nicht aber *Meier*.

---

1 Das stimmt nicht ganz. Die Menge der rationalen Zahlen ist eigentlich unendlich. Auf einem Computer ist aber sowohl die Anzahl der Nachkommastellen als auch der Wertebereich immer irgendwo beschränkt.

Wenn Sie nun für das Attribut *preis* den Typ *DECIMAL* festlegen, stellen Sie damit sicher, dass sich für Preise nur Zahlen eintragen lassen. Damit verbessern Sie die Integrität der Datenbank.

Nun wird klarer, was ein Attribut eigentlich ist.

> Ein Attribut ist ein Paar aus einem Namen und einem Typ, z. B. `preis:DECIMAL`.

Das relationale Modell verlangt lediglich ein offenes Typensystem. Das heißt, welche Typen es gibt, ist nicht festgelegt, sondern hängt vom jeweiligen DBMS ab. Idealerweise bietet das DBMS dem Programmierer sogar die Möglichkeit eigene Typen zu definieren.

8.2 Datentypen in MySQL

Um die Anforderungen Ihrer Kundin zu erfüllen, müssen Sie für die Attribute *titel*, *beschreibung* und *preis* noch Typen vergeben. Da Sie sich bereits dazu entschieden haben, das DBMS *MySQL* zu verwenden, stellt sich die berechtigte Frage: *Welche Datentypen bietet MySQL an?*

Die wichtigsten Gruppen von Datentypen in MySQL sind:

- ***Zeichenketten*** (Strings)
- ***Zahlen***
- ***Zeit- und Datumsangaben***
- ***Wahrheitswerte***

8.2.1 Zeichenketten

Von den Strings ist für Sie zunächst der Datentyp `VARCHAR` interessant. Er ermöglicht die Speicherung unterschiedlich langer Zeichenketten.

Beispiel

- Ajax & DOM
- Meier
- m.emrich@webmasters.de

Dieser Datentyp lässt sich also auch ideal zur Speicherung von Seminartiteln verwenden. VARCHAR steht für *variable* und *character* – gemeint ist eine variable Anzahl von Zeichen.

Sobald Sie VARCHAR einsetzen, müssen Sie angeben, wie lang die längste erwartete Zeichenkette sein kann. Wenn Sie also der Meinung sind, dass ein Seminartitel nie länger als 80 Zeichen sein kann, verwenden Sie einen `VARCHAR(80)`. Falls Sie einmal ein Attribut *Nachnamen* in Ihrer Tabelle haben, so ist dafür vermutlich ein `VARCHAR(40)` ausreichend.

Die maximale Länge, die ein VARCHAR haben kann, ist 255[1] . Für längere Texte benötigen Sie einen weiteren Datentyp: `TEXT`. Für diesen Datentyp müssen Sie keine Länge angeben. Verwenden Sie ihn immer dann, wenn Ihre Strings nicht in einen `VARCHAR(255)` passen. Sie sollten TEXT aber auch nur in dieser Situation verwenden. Einfach grundsätzlich TEXT statt VARCHAR zu verwenden, ist keine gute Idee. Sie verschenken sonst Speicherplatz und Performance.

---

1 Allerdings gilt das nur für MySQL-Versionen bis 5.0.3. Ab Version 5.0.3 kann ein VARCHAR bis zu 65535 Zeichen fassen.

Zu kleinlich sollten Sie aber auch nicht sein. Wenn Sie einen zu kleinen Datentyp verwenden und der Anwender versucht einen längeren Text zu speichern, schneidet MySQL den Rest einfach ab! Das nächste Beispiel zeigt, wie ungünstig das enden kann.

Beispiel

Um potentielle Teilnehmer zu einer früheren Anmeldung zu bewegen, erweiterte eine

Sekretärin die Seminarbeschreibung des JavaScript-Seminars um folgenden Text.

> *Hinweis: Auf dieses Seminar erhalten Sie einen Preisnachlass von 50%, wenn Sie sich bis zum 01.05. anmelden und mindestens ein weiteres Seminar buchen.*

Zusammen mit dem ca. 180 Zeichen langen, vorhanden Beschreibungstext des Seminars sprengte der Text den vorgegebenen VARCHAR(250). Was die Datenbank noch einfügte, war:

> *Hinweis: Auf dieses Seminar erhalten Sie einen Preisnachlass von 50%*

Letztendlich müssen Sie immer den richtigen Mittelweg finden – nicht zu groß und nicht zu klein. Ich bin mir sicher, dass ich die Goldlöckchen-Lösung schon einmal erwähnt habe.

Aus agiler Sichtweise betrachtet, ist es aber auch möglich, sicherheitshalber erst einmal einen zu großen Wert zu wählen – schlimmstenfalls VARCHAR(255) oder TEXT. Sie können zu einem späteren Zeitpunkt immer noch kürzen. Oftmals können Sie nämlich zu Beginn eines Projektes noch nicht vorhersehen, wie groß die Strings später im Alltag tatsächlich werden.

8.2.2 Zahlen

In MySQL gibt es – wie in den meisten Programmiersprachen – verschiedene Zahlentypen. Grundsätzlich ist zu unterscheiden zwischen

- ganzen Zahlen und
- reellen Zahlen (solche mit Fließkommaanteil).

Beispiel

- **ganze Zahl**: 42
- **reelle, nicht ganze Zahl**: 1699,99

Im einfachsten Fall verwenden Sie für ganze Zahlen den Datentyp `INTEGER` und für reelle Zahlen den Datentyp `DECIMAL`. Nehmen wir an, Ihre Kundin möchte, dass Sie zu jedem Seminar die Dauer des Seminars in Tagen speichern. Falls es keine halbtägigen Seminare gibt, könnten Sie ein Attribut `dauer_in_tagen` vom Typ `INTEGER` hinzufügen. Für das Attribut `preis` bietet sich offensichtlich der Typ `DECIMAL` an. Ähnlich dem `VARCHAR`, müssen Sie auch den `DECIMAL` konfigurieren. Geben Sie in runden Klammern zwei Zahlen an - getrennt durch ein Komma. Die erste beschreibt die maximale Gesamtstellenzahl. Mit der zweiten können Sie die Anzahl der Nachkommastellen festlegen.

Beispiel

Für die Preise der Seminare bietet es sich an, einen `DECIMAL(6,2)` zu verwenden. Damit haben die Preise 4 Stellen vor und 2 Stellen nach dem Komma. So lässt sich z. B. der Preis *1699,99 €* problemlos speichern. Das teuerste Seminar, dass die Akademie anbieten könnte, hätte dann einen Preis von *9999,99 €*.

Auch hier gilt es, einen Datentyp optimaler Größe für das jeweilige Attribut zu finden.

MySQL kennt noch viele weitere Datentypen, aber für die momentanen Anforderungen reichen die genannten erst einmal aus.

8.2.3 Zeit und Datum

Um den Registrierungszeitpunkt in der Benutzertabelle speichern zu können, benötigen Sie einen Datentyp für Zeitangaben. MySQL unterscheidet dabei zwischen den Datentypen `DATE`, `TIME` und `DATETIME`. Alle drei Datentypen verwenden das ANSI-Format. Was das genau bedeutet, können Sie folgender Tabelle entnehmen:

Datentyp	Bedeutung	Beispiel	Beispiel im ANSI-Format	ANSI-Format
DATE	Datum	29.07.1978	‘1978-07-29’	‘YYYY-MM-DD’
TIME	Uhrzeit	13 Uhr 25 und 17 Sekunden	‘13:25:17’	‘HH:MM:SS’
DATETIME	Ein Zeitpunkt, d. h. die Kombination aus Datum und Uhrzeit	Der 29.07.1978 um 13:25 und 17 Sekunden	‘1978-07-29 13:25:17’	‘YYYY-MM-DD HH:MM:SS’

Tabelle 8.1 Zeit und Datumsangaben in MySQL

Für das Speichern des Registrierungsdatums bietet sich `DATE` an.

8.2.4 Wahrheitswerte

Wir konzipieren hin und wieder neue Seminare. Diese sind aber nicht immer gleich öffentlich verfügbar. Wir möchten deswegen Seminare als inaktiv markieren können. Diese sollen dann noch nicht auf der Website erscheinen.

Wahrheitswerte sind Werte von Typ `BOOLEAN`. Es gibt genau zwei davon: `TRUE` (wahr) und `FALSE` (unwahr). Mithilfe dieses Datentyps definieren Sie Attribute, für die es nur zwei Möglichkeiten gibt. Für Seminare, die noch nicht fertiggestellt sind und deswegen auch noch nicht öffentlich angezeigt werden bietet sich deshalb ein Attribut *aktiv* vom Datentyp BOOLEAN an.

Sie sollten allerdings noch wissen, dass MySQL eigenlich gar keinen BOOLEAN anbietet. In Wirklichkeit verwendet MySQL stattdessen einen INTEGER[1] und interpretiert die Zahl `0` als `FALSE` und die Zahl `1` als `TRUE`. Die Literale TRUE und FALSE können Sie dennoch

verwenden, es sind aber Synonyme für die beiden zugehörigen Zahlen.

---

1 Genauer gesagt den Typ TINYINT(1)

8.3 Zurück zum physischen Datenmodell

Sie haben Datentypen nun sowohl im relationalen Modell, als auch in der praktischen Umsetzung von MySQL kennen gelernt. Somit können Sie das physische Datenmodell vervollständigen. Dazu müssen Sie lediglich die Datentypen im Klassendiagramm ergänzen. Hier sehen Sie das verbesserte Diagramm der Seminarverwaltung:

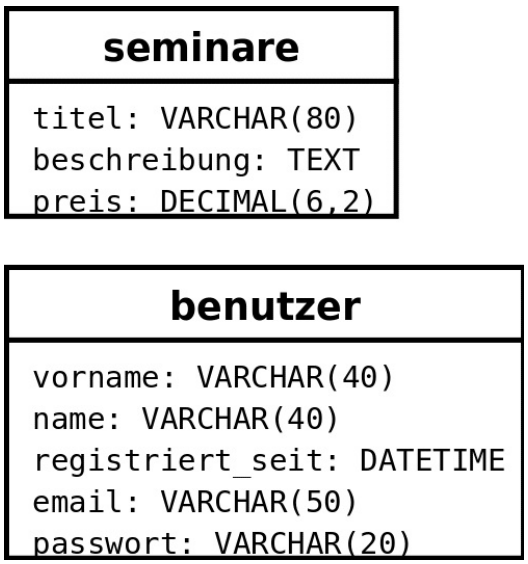


Abb. 8.1 Physisches Datenmodell der Seminarverwaltung mit Typen

Zusammenfassung

- Ein ***Typ*** (oder ***Datentyp***) ist eine benannte endliche Menge von Werten.
- Ein ***Attribut*** ist ein Paar aus einem Namen und einem Typ.
- Jedes DBMS kann andere Typen anbieten.
- Das physische Datenmodell zeigt die zugeordneten Typen.

MySQL-Typenübersicht

- ***Zeichenketten***: `VARCHAR, TEXT`
- ***Zahlen***: `DECIMAL, INTEGER`
- ***Zeit- und Datumsangaben***: `DATE, TIME, DATETIME`
- ***Wahrheitswerte***: `BOOLEAN`

8.4 Aufgaben zur Selbstkontrolle

Testen Sie Ihr Wissen

1. Nennen Sie 3 Datentypen und erklären Sie deren Einsatzzweck!
2. Welche Gruppen von Datentypen gibt es in MySQL?
3. Welchen Datentyp würden Sie für eine Telefonnummer verwenden?

Übungen

Aufgabe 1: Datentypen für Werte

Im Folgenden sind Reihen von Werten angegeben. Das Komma dient dabei als Trenner zwischen den Werten. Zeichenketten stehen in einfachen Anführungszeichen »'«.

Wählen Sie geeignete Datentypen für jede Reihe von Werten.

1. 9, 17, 28, 45, 12, 9, 17
2. 11.4, 7.3, 9.8, 99.2
3. 3, 4, 3467.278, 5, 6, 7, 8
4. 'Stefanie', 'Jan', 'Sabine', 'Michael', 'Udo', 'Sebastian', 'Katherina'
5. 0179534891, 00491721234, 091112345

Aufgabe 2: Datumsangaben

Notieren Sie folgende Zeitangaben im ANSI-Format:

1. August 23, 1923
2. 2. Dezember 1975
3. 11. September 2001 um 9:03

Aufgabe 3: Typen des Filmverleihs

Ergänzen Sie das physische Modell des Filmverleihs (Übung aus Lektion 6) um passende Typen.

Zusatzübungen

Aufgabe 4: Datentypen für Attribute

Wählen Sie geeignete Datentypen für die Attribute der folgenden Tabellen und zeichnen Sie die physischen Modelle in UML.

- autos = {modell, hersteller, leistung_in_ps, von_0_auf_100_in_sec}

- buecher = {titel, autor, erscheinungsdatum, verlag, seitenzahl}

Aufgabe 5: Typen der Fluggesellschaft

Ergänzen Sie das physische Modell der Fluggesellschaft (Übung aus Lektion 6) um passende Typen.

Aufgabe 6: Typen der Partnervermittlung

Ergänzen Sie das physische Modell der Partnervermittlung (Übung aus Lektion 6) um passende Typen[1].

---

1 Nein, nein – Datentypen. Die anderen passenden Typen tragen Sie erst ein, wenn die Anwendung läuft.

9 SQL: Tabellen anlegen und löschen

In dieser Lektion lernen Sie:

- aus welchen Teilen SQL besteht.
- wie Sie Tabellen anlegen.
- wie Sie Tabellen löschen.

9.1 Tabellen anlegen

Sie haben nun ein physisches Datenmodell als Diagramm auf dem Papier. Um einen praktischen Nutzen daraus zu gewinnen, müssen Sie es in eine reale Datenbank übertragen. Dazu benötigen Sie wieder SQL-Anweisungen. Der Befehl zum Anlegen einer Tabelle lautet sinnigerweise `CREATE TABLE`.

Starten Sie den MySQL-Kommandozeilenclient.

```
mysql -u root
```

Wechseln Sie in die Datenbank *seminarverwaltung*, die Sie in Lektion 3 angelegt haben:

```
USE seminarverwaltung;
```

Mit dem MySQL-Befehl `USE` können Sie eine bestimmte Datenbank zur Verwendung auswählen. Wenn Sie sich nicht vertippt haben und die Datenbank existiert, quittiert MySQL mit

```
Database Changed
```

USE hat übrigens die BNF

```
USE database_name
```

Ab sofort können Sie Befehle absetzen, die speziell die Datenbank *seminarverwaltung* betreffen.

Legen Sie nun die Tabellen an, die Sie zuvor im physischen Datenmodell beschrieben haben:

```
CREATE TABLE seminare (titel VARCHAR(80), beschreibung TEXT, preis
DECIMAL(6,2));
```

Der Client antwortet mit:

```
Query OK
```

Query heißt »Anfrage«. Alle Anweisungen werden in SQL als Anfragen bezeichnet. Deswegen ist `Query OK` die Erfolgsmeldung, die Sie meistens erhalten. Sie können zur Sicherheit überprüfen, ob die Tabelle tatsächlich angelegt wurde.

```
SHOW TABLES;
```

```
+----------------------------+
| Tables_in_seminarverwaltung |
+----------------------------+
| seminare                   |
+----------------------------+
1 row in set (0.00 sec)
```

Legen Sie nun noch die Tabelle Benutzer an:

CREATE TABLE benutzer (vorname VARCHAR(40), name VARCHAR(40), email VARCHAR(50), passwort VARCHAR(20), registriert_seit DATE);

`SHOW TABLES` zeigt nun beide Tabellen:

```
+----------------------------+
| Tables_in_seminarverwaltung |
+----------------------------+
| benutzer                   |
```

```
| seminare                    |
+-----------------------------+
2 rows in set (0.00 sec)
```

Außerdem können Sie überprüfen, ob die richtigen Spalten mit entsprechenden Datentypen vorhanden sind:

```
SHOW COLUMNS FROM seminare;
```

```
+--------------+--------------+------+-----+---------+-------+
| Field        | Type         | Null | Key | Default | Extra |
+--------------+--------------+------+-----+---------+-------+
| titel        | varchar(80)  | YES  |     | NULL    |       |
| beschreibung | text         | YES  |     | NULL    |       |
| preis        | decimal(6,2) | YES  |     | NULL    |       |
+--------------+--------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

```
SHOW COLUMNS FROM benutzer;
```

```
+-----------------+-------------+------+-----+---------+-------+
| Field           | Type        | Null | Key | Default | Extra |
+-----------------+-------------+------+-----+---------+-------+
| vorname         | varchar(40) | YES  |     | NULL    |       |
| name            | varchar(40) | YES  |     | NULL    |       |
| email           | varchar(50) | YES  |     | NULL    |       |
| passwort        | varchar(20) | YES  |     | NULL    |       |
| registriert_seit| date        | YES  |     | NULL    |       |
+-----------------+-------------+------+-----+---------+-------+
5 rows in set (0.01 sec)
```

Sie sehen die Spalten und ihre Typen. Die zusätzlichen Informationen zu *Null*, *Key*, *Default* und *Extra* können Sie vorerst außer Acht lassen.

Die BNF von SHOW COLUMNS ist übrigens ganz einfach:

```
SHOW COLUMNS FROM tbl_name
```

9.2 DML, DDL und DCL

SQL ist in drei logische Teile untergliedert:

- DDL (Data Definition Language)
- DML (Data Manipulation Language)
- DCL (Data Control Language)

Der Befehl `CREATE TABLE` gehört zur ***DDL***, der ***Data Definition Language***. Dieser Teil der Sprache beschäftigt sich mit dem Definieren von Daten d. h. die dazugehörigen Befehle betreffen die Struktur der Daten (Tabellen, Spalten, Datentypen) – nicht aber die Daten selbst. Sie können damit zwar Tabellen anlegen und verändern, jedoch nicht die Daten (also z. B. Seminare), die in diesen Tabellen gespeichert werden. Dafür benötigen Sie die ***Data Manipulation Language***, die Sie ebenfalls bald kennen lernen. Mit der ***Data Control Language*** schließlich können Sie die Zugriffsrechte auf die Daten steuern (Security).

Die `SHOW` -Anweisungen gehören übrigens nicht zum SQL-Standard selbst, sondern sind MySQL-spezifische Befehle, die der Datenbankadministration dienen, sogenannte ***Database Administration Statements***. Ähnlich verhält es sich mit `USE`, es gehört zu den ***MySQL Utility Statements***.

9.3 BNF: Wiederholung und Verschachtelung

Der Befehl CREATE TABLE besitzt folgende BNF[1]:

1 Beachten Sie, dass es sich dabei noch um eine vereinfachte Variante handelt. Sobald Sie weitere Möglichkeiten des Befehls kennen lernen, werde ich Ihnen auch eine erweiterte BNF von `CREATE TABLE` vorstellen. Die vollständige BNF finden Sie im MySQL-Manual: dev.mysql.com/doc/refman/5.0/en/create-table.html

```
1 CREATE TABLE tbl_name (create_definition,...)
2 create_definition: col_name data_type
```

Listing 9.1 BNF von CREATE TABLE

Schauen Sie sich die erste Zeile etwas genauer an:

```
CREATE TABLE tbl_name (create_definition,...)
```

Die Variable *tbl_name* steht für einen von Ihnen gewählten Tabellennamen. *create_definition* ist dagegen in einer eigenen BNF-Definition nochmal genauer erklärt:

```
create_definition: col_name data_type
```

Vor dem Doppelpunkt »:« steht die Variable (Nichtterminal), die hier genauer definiert wird. Nach dem Doppelpunkt steht der definierende BNF-Ausdruck. *col_name* meint dabei einen Spaltennamen, den Sie wiederum selbst wählen können (z. B. *titel* oder *preis*). Für die Variable *data_type* können Sie jeweils einen Datentyp einsetzen (z. B. VARCHAR(80) oder TEXT).

Wie Sie sehen, können BNF-Ausdrücke aus mehreren verschachtelten Regeln bestehen. Eine weitere Neuerung hat sich in die erste BNF-Regel eingeschlichen – die drei Punkte »…« . Sie stehen in der BNF für eine beliebige ***Wiederholung***. Dadurch können Sie mit `CREATE TABLE` nicht nur eine Spalte anlegen, sondern beliebig viele.

Zur Verdeutlichung möchte ich Ihnen detailliert darstellen, wie Sie von der abstrakten BNF-Form zu einer konkreten Ausprägung des Befehls kommen.

- BNF von `CREATE TABLE`

```
CREATE TABLE tbl_name (create_definition,...)
create_definition: col_name data_type
```

1. Auflösen der Wiederholung

   ```
   CREATE TABLE table_name (create_definition1, create_definition2,
   create_definition3)
   ```

2. Ersetzen der Variablen durch Ihre Definition

   ```
   CREATE TABLE table_name(col_name1 data_type1, col_name2 data_type2,
   col_name2 data_type2)
   ```

3. Ersetzen der Variablen durch konkrete Werte

   ```
   CREATE TABLE seminare (titel VARCHAR(80), beschreibung TEXT, preis
   DECIMAL(6,2))
   ```

Die Reihenfolge der Schritte ist dabei nicht von Bedeutung. Wichtig ist nur, dass Sie so zeigen können, dass ein spezieller SQL-Befehl tatsächlich der vorgegebenen BNF-Syntax

entspricht.

9.4 Löschen von Tabellen

Unverhofft klingelt das Telefon, Ihre Kundin ist am Apparat. Vorsichtig formuliert Sie ihre Anfrage:

Nach Durchsicht unserer Daten ist mir aufgefallen, dass wir Seminare anbieten, deren Titel die 80-Zeichengrenze durchbricht. Könnten Sie vielleicht die Titellänge auf 120 Zeichen erhöhen?

Nachdem Sie ihr versichert haben, dass das kein Problem ist, klingt Sie sichtlich erleichtert.

Danke, ich fürchtete schon, dass wir die Titel kürzen müssten.

Wie realisieren Sie das nun?

Sie müssen die Tabelle lediglich löschen und nochmal neu anlegen – diesmal mit größerer *titel* -Spalte. Verwenden Sie dazu folgende Anweisung:

```
DROP TABLE seminare;
```

Wie schon bei den Datenbanken heißt es `DROP` (fallenlassen) und nicht etwa `DELETE` (löschen). Das liegt daran, dass `DELETE` zur DML gehört. Um Verwechslungen zu vermeiden, verwendet die DDL durchgängig `DROP`.

Jetzt können Sie die verbesserte Tabelle neu anlegen:

```
CREATE TABLE seminare (titel VARCHAR(120), beschreibung TEXT, preis
DECIMAL(6,2));
```

Hier noch die BNF von `DROP TABLE`:

```
DROP TABLE tbl_name
```

Zusammenfassung

- Die SQL ist in die drei Teile DDL, DML und DCL gegliedert.

BNF

Konzept	Darstellung
Terminale	Wort in Großbuchstaben
Nichtterminale (Variablen)	Wort in Kleinbuchstaben
Wiederholung	…
Verschachtelte Definition	:

Tabelle 9.1 Bisherige BNF-Konzepte

SQL: DDL

Zweck	BNF
Datenbank anlegen	`CREATE DATABASE db_name`
Datenbank löschen	`DROP DATABASE db_name`
Tabelle anlegen	`CREATE TABLE tbl_name (create_definition,...)` `create_definition: col_name data_type`
Tabelle Löschen	`DROP TABLE tbl_name`

Tabelle 9.2 Bisherige DDL-Anweisungen

SQL: Utility und Administration

Zweck	BNF
Datenbanken auflisten	`SHOW DATABASES`
Datenbank wechseln	`USE database_name`
Tabellen der aktuellen DB anzeigen	`SHOW TABLES`
Spalten einer Tabelle anzeigen	`SHOW COLUMNS FROM tbl_name`

Tabelle 9.3 Bisherige Utility- und Administrations-Anweisungen

9.5 Aufgaben zur Selbstkontrolle

Testen Sie Ihr Wissen

1. Welche Art von SQL-Anweisungen werden durch die DDL zusammengefasst?
2. Welche durch die DML?
3. Welche durch die DCL?
4. Gibt es neben DDL, DML und DCL noch weitere Arten von SQL-Anweisungen?
5. Wie viele Tabellen kann eine Datenbank enthalten?

Übungen

Aufgabe 1: Fehler finden

Finden Sie die Fehler in folgender SQL-Anweisung. Notieren Sie die verbesserte Anweisung.

```
CRIÄDE TABEL kunden (kunden nr, name VARCHAR, INTEGER anzahl_bestellungen,
telefon_nr INTEGER, telefon_nr INTEGER;
```

Aufgabe 2: Korrektes SQL

Zeigen Sie durch schrittweise Ausprägung der BNF, dass folgende SQL-Anweisung korrekt ist:

```
CREATE TABLE notebooks (preis DECIMAL(7, 2), modell VARCHAR(20),
hersteller VARCHAR(20), liefer_status VARCHAR(20), cpu VARCHAR(15),
gewicht_in_kg INTEGER);
```

Aufgabe 3: BNF ausprägen

Prägen Sie die folgenden BNF-Ausdrücke mit jeweils 2 Beispielen aus:

1. ICH GEHE AN DIE BAR UND MIXE MIR EINEN cocktail. DAZU BENÖTIGE ICH zutaten UND zutat. ANSCHLIESSEND GARNIERE ICH DEN DRINK MIT frucht. zutaten: zutat, …
2. ZU ereignis WÜNSCHE ICH MIR geschenk geschenk: farbe fahrzeug VON firma firma: firmenname rechtsform AUS ort

Aufgabe 4: Mengenschreibweise nach DDL

Überführen Sie folgende, in Mengenschreibweise angegebenen Tabellen nach DDL.

1. autos = {hersteller, modell, leistung_in_ps, von_0_auf_100_in_sec}
2. buecher = {titel, autor, erscheinungsdatum, klappentext, seitenzahl}

Aufgabe 5: Tabellenschema des Filmverleihs

Legen Sie die Tabellen des Filmverleihs an, gemäß dem physischen Modell aus der Übung von Lektion 6.

Zusatzübungen

Aufgabe 6: Tabellenschema der Fluggesellschaft

Legen Sie die Tabellen der Fluggesellschaft an, gemäß dem physischen Modell aus der Übung von Lektion 6.

Aufgabe 7: Tabellenschema der Partnervermittlung

Legen Sie die Tabellen der Partnervermittlung an, gemäß dem physischen Modell aus der Übung von Lektion 6.

10 SQL: Datensätze einfügen und auslesen

In dieser Lektion lernen Sie:

- wie Sie mit SQL Daten in eine Tabelle einfügen.
- wie Sie mit SQL Daten aus einer Tabelle auslesen.

10.1 Daten einfügen

Nachdem Sie eine Tabelle für Seminare erstellt haben, besteht der nächste Schritt darin, Seminare anzulegen.

- Datenbanken & SQL
  Beschreibung: Nahezu alle modernen Webapplikationen speichern ihren Datenbestand …
  Preis: 975,00 €

Um das obige Seminar einzufügen, benötigen Sie den `INSERT` -Befehl (Einfügen). Geben Sie folgende SQL-Anweisung ein:

```
INSERT INTO seminare (titel, beschreibung, preis) VALUES ('Datenbanken & SQL', 'Nahezu alle modernen W…', 975.00);
```

MySQL antwortet mit

```
Query OK, 1 row affected (0.01 sec)
```

1 row affected (eine Zeile betroffen) bedeutet hier, dass das DBMS **eine** neue Zeile in die Tabelle eingefügt hat.

Genauso einfach können Sie einen Benutzer einfügen.

```
INSERT INTO benutzer (name, vorname, registriert_seit, email, passwort)
VALUES ('Reich', 'Frank', '2008-04-12', 'f.reich@example.com', 'kochtopf');
```

Achten Sie bei der Angabe des Registrierungsdatums darauf, dass Sie den Wert in Anführungszeichen setzen und im ANSI-Format angeben – d. h. zuerst das Jahr (vierstellig), dann der Monat (zweistellig) und als letztes der Tag (auch zweistellig).

Die BNF von `INSERT INTO` lautet:

```
INSERT INTO tbl_name (col_name,...) VALUES (value,...)
```

Die Variable *col_name* lässt sich wie gewohnt durch einen Spaltennamen ersetzen. Für die Variable *value* können Sie einen beliebigen Wert einsetzen. Zahlen können Sie direkt angeben. Strings und Datumsangaben dagegen müssen Sie in einfache Anführungszeichen »'« (shift-#) setzen.

Sie können die Reihenfolge der Spalten auch vertauschen – achten Sie aber darauf, dass Sie die Werte in der gleichen Reihenfolge wie die Spaltennamen angeben.

Beispiel

Die Anweisung

```
INSERT INTO seminare (preis, titel, beschreibung) VALUES (975.00, 'Datenbanken & SQL', 'Nahezu alle modernen…');
```

ist korrekt. Folgende Anweisung

```
INSERT INTO seminare (preis, titel, beschreibung) VALUES ('Datenbanken & SQL', 975.00, 'Nahezu alle modernen…');
```

ist jedoch fehlerhaft, da MySQL versucht für die Spalte *preis* den String `'Datenbanken & SQL'` einzufügen

10.2 Daten anzeigen

Prüfen Sie nun, ob das Seminar in der Tabelle angekommen ist. Das Zugreifen/Ausgeben von Datensätzen bezeichnet SQL als Selektierung von Datensätzen. Sie benötigen deswegen die `SELECT` -Anweisung.

```
SELECT * FROM seminare;
```

Diese Anweisung können Sie lesen als:

> *Wähle alle Spalten aus der Tabelle seminare*

Der Stern »*« gibt an, dass Sie alle Spalten benötigen. Hinter dem `FROM` geben Sie den Namen der Tabelle an.

Sie erhalten folgendes Ergebnis (hier leicht gekürzt):

```
+------------------+-------------------------+--------+
| titel            | beschreibung            | preis  |
+------------------+-------------------------+--------+
| Datenbanken & SQL | Nahezu alle modernen… | 975.00 |
+------------------+-------------------------+--------+
```

Hier noch die (vereinfachte) BNF von `SELECT`:

```
SELECT * FROM tbl_name;
```

Achtung SQL-Fehler

Beachten Sie bitte, dass SQL hier einen Fehler hat – nicht nur MySQL, sondern der SQL-Standard im Allgemeinen. Im relationalen Datenmodell ist es nicht erlaubt, mehrere exakt identische Datensätze in eine Tabelle einzufügen. SQL lässt das aber fälschlicherweise zu.

Geben Sie einfach mehrmals

```
INSERT INTO seminarverwaltung (preis, titel, beschreibung) VALUES
('Datenbanken & SQL', 975.00, 'Nahezu alle modernen Webapplikationen
speichern ihren Datenbestand…');
```

ein.

Wenn Sie anschließend mit

```
SELECT * FROM seminare;
```

die Tabelle anzeigen, sehen Sie, dass der gleiche Datensatz mehrfach vorhanden ist. Dieses Fehlverhalten kann gravierende Folgen haben.

> *Es gibt eine Vielzahl von praktischen Argumenten, die die Sichtweise unterstützen, dass doppelte Tuple nicht erlaubt sein dürften* *C.J. Date (2005)*

Sie lernen bald, wie Sie das Problem gänzlich vermeiden. Achten Sie momentan einfach darauf, denselben Datensatz nicht versehentlich mehrfach einzufügen.

Zusammenfassung

- Mit der `INSERT`-Anweisung fügen Sie Datensätze ein.
- Mit der `SELECT`-Anweisung lesen Sie Datensätze aus.
- Setzen Sie Strings und Datumsangaben in Anführungszeichen.
- Achten Sie darauf, denselben Datensatz nicht versehentlich mehrfach einzufügen.

SQL: DML

Zweck	BNF
Datensätze auslesen	`SELECT * FROM tbl_name`
Datensatz einfügen	`INSERT INTO tbl_name (col_name,...) VALUES (value,...)`

Tabelle 10.1 Bisherige DML-Anweisungen

10.3 Aufgaben zur Selbstkontrolle

Übungen

Aufgabe 1: Fehlersuche

Finden Sie die Fehler in den beiden folgenden SQL-Anweisungen:

- INSERT INTO seminare (preis, titel, beschreibung) VALUES (Im Lotto gewinnen – leicht gemacht, 5999, Sicherlich möchte jeder einmal im…)
- INSERT VALUES INTO “benutzer” (‘Log’, ‘Anna’, anna@log.de, ‘geheimes_passwort’, registriert_seit, ‘Frau’) (name, vorname, email, 2008-02-18, anrede);

Aufgabe 2: Weitere Seminare und Benutzer einfügen

Fügen Sie die restlichen Seminare aus der folgenden Tabelle in die *seminarverwaltung* ein:

titel	beschreibung	preis
Ruby on Rails	Ruby on Rails ist das neue, sensationelle OpenSource-Framework, das auf der modernen…	1900,00
Ajax & DOM	Ajax ist längst dem Hype-Stadium entwachsen. Mehr und mehr professionelle Websites folgen …	1699,99
JavaScript	JavaScript ist eine Programmiersprache mit vielseitigen Einsatzmöglichkeiten. Sie gilt…	2500,00
Datenbanken & SQL	Nahezu alle modernen Webapplikationen speichern ihren Datenbestand …	975,00

Tabelle 10.1 Tabelle *seminare*

Fügen Sie zwei weitere Benutzer in die Datenbank ein:

- Huana, Marie hat sich am 03.02.2009 registriert. Ihre E-Mail ist *huana@example.com* und sie verwendet das Passwort *reibekuche*.
- Meisenbär, Andreas hat sich am 15.07.2008 registriert. Seine E-Mail ist *a.meisenbär@example.com* und er verwendet das Passwort *schüssel*.

Aufgabe 3: Daten des Filmverleihs

Fügen Sie zum Testen der Datenbank folgende Daten ein:

Filme

- Cube (1997), 90 min. Sieben völlig Fremde mit sehr unterschiedlichen Charakterzügen werden unfreiwillig in ein endloses kafkaartiges Labyrinth voller

Fallen gesteckt.
- Der Herr der Ringe – Die Gefährten (2001), 178 min. In einem kleinen Dorf wird einem jungen Hobbit namens Frodo ein uralter, magischer Ring anvertraut. Er muss sich auf eine epische Reise zum Schicksalsberg begeben, um ihn zu zerstören.

Regisseure

- Vincent Natali, 06.01.1969
- Peter Jackson, 31.10.1961

Übungen

Aufgabe 4: Daten der Fluggesellschaft

Überlegen Sie sich je drei Beispieldatensätze pro Tabelle der Fluggesellschaft und tragen Sie diese ein.

Aufgabe 5: Daten der Partnervermittlung

Überlegen Sie sich je drei Beispieldatensätze pro Tabelle der Partnervermittlung und tragen Sie diese ein.

11 Schlüssel

In dieser Lektion lernen Sie:

- was Schlüssel sind.
- wie Sie Schlüssel bestimmen können.
- welches Muster die Schlüsselproblematik im ORM löst.

Date ist wie Brokoli; Er wird nie mein Lieblingsautor sein, aber ich weiß, dass das was er schreibt gut für mich ist.

Stuart Ainsworth (2009)

Ein wichtiges Prinzip relationaler Datenbanken ist die Möglichkeit, Datensätze eindeutig zu identifizieren, um sie wiederzufinden. Das wird insbesondere später wichtig, wenn Sie Datensätze aus verschiedenen Tabellen einander zuordnen – beispielsweise weil Sie wissen möchten, welcher Kunde welche Seminare gebucht hat.

Im relationalen Modell gibt es dafür Schlüssel. So wie ein echter Schlüssel genau zu einem bestimmten Schloss passt, so passt ein Datenbank-Schlüssel genau zu einem bestimmten Datensatz.

11.1 Was ist ein Schlüssel?

Christopher sagt…

> *Ein Schlüssel ist vereinfacht gesagt: ein eindeutiger Bezeichner.*
> *Genauer: Sei K eine Teilmenge des Headings von Relvar R; dann ist K […] ein Schlüssel für R genau dann, wenn (a) kein möglicher Wert für R zwei verschiedene Tupel mit demselben Wert für K; enthalten kann (Eindeutigkeit), während (b) dies nicht für eine echte Teilmenge von K gilt (Irreduzibiltät).*
>
> Date (2007)

Christophers Definition ist möglicherweise etwa schwierig zu lesen. Deswegen möchte ich Ihnen das Konzept Schritt für Schritt näherbringen. Im Grunde möchten Sie für jeden Datensatz (d. h. Zeile, Tupel) einen eindeutigen Bezeichner haben. Das ist der Sinn eines Schlüssels. Wie lässt sich das erreichen?

Betrachten Sie nochmal die Tabelle *seminare*.

- seminare = {titel, beschreibung, preis}

titel	beschreibung	preis
Ruby on Rails	Ruby on Rails ist das neue, sensationelle OpenSource-Framework, das auf der modernen…	2500,00
Ajax & DOM	Ajax ist längst dem Hype-Stadium entwachsen. Mehr und mehr professionelle Websites folgen …	1699,99
JavaScript	JavaScript ist eine Programmiersprache mit vielseitigen Einsatzmöglichkeiten. Sie gilt…	2500,00
Datenbanken & SQL	Nahezu alle modernen Webapplikationen speichern ihren Datenbestand …	975,00

Tabelle 11.1 Tabelle *seminare*

Als eindeutiger Bezeichner lässt sich der *titel* wählen. Sie können anhand des Titels *Datenbanken & SQL* das dazugehörige Seminar – also die ganze Zeile – eindeutig identifizieren.

Sie können in diesem Fall notieren:

- {titel} ist ein Schlüssel der Tabelle *seminare*.

Manchmal benötigen Sie aber auch mehrere Attibute. Betrachten Sie bitte folgende Tabelle:

- uebungen = {seminar_titel, kapitel_nr, uebungs_nr, punkte}

seminar_titel	kapitel_nr	uebungs_nr	punkte
Datenbanken & SQL	4	1	5
Datenbanken & SQL	4	2	7
Datenbanken & SQL	4	3	5
Datenbanken & SQL	5	1	10
JavaScript	2	1	3
JavaScript	4	1	7

Tabelle 11.2 Tabelle uebungen (Auszug)

Diese Tabelle wird im Schulungsunternehmen benötigt, um festzuhalten, welche Übung wie viele Punkte bringt. Es sind dabei alle Seminare in der Tabelle vorhanden, mit ihren jeweiligen Kapiteln und Übungen. Ein Zeile der Tabelle ist jeweils eine Übung. Um die Übung nachzulesen, müssen Sie das Schulungsskript des jeweiligen Seminars aufschlagen und dort im passenden Kapitel die entsprechende Übung finden. So ist z. B. die *Übung Nr. 2* aus *Kapitel 4* des Seminars *Datenbanken & SQL* 7 Punkte wert. Um einen Datensatz, d.h eine Übung eindeutig zu bestimmen, sind drei Attribute notwendig: *seminar_titel*, *kapitel_nr* und *uebungs_nr*. Sie können also sagen:

- {seminar_titel, kapitel_nr , uebungs_nr} ist ein Schlüssel der Tabelle *uebungen*.

Ein Schlüssel ist grundsätzlich eine Menge von Attributen. Es darf natürlich nicht irgendeine Menge sein. Beispielsweise kann die Menge *{name, schuhgroesse}* kein Schlüssel der Tabelle *seminare* sein. Die Attribute *name* und *schuhgroesse* kommen in *seminare* nicht einmal vor! Deswegen muss es sich bei einem Schlüssel einer Tabelle immer um eine Teilmenge des Headings (Überschrift) genau dieser Tabelle handeln.

> Ein Schlüssel ist eine Teilmenge des Headings einer Tabelle.

Das relationale Modell fordert zudem von einer solchen Menge zwei Eigenschaften, damit sie als Schlüssel gilt:

- Eindeutigkeit
- Irreduzibilität

Eindeutigkeit

> Ein Schlüssel ist immer eindeutig.

Eindeutigkeit bedeutet, dass Sie anhand des Schlüssels immer nur **einen** Datensatz finden – nicht mehrere. In der Tabelle *seminare* kann also z. B. {preis} kein Schlüssel sein, da es durchaus mehrere Seminare geben kann, die genauso viel kosten. Wenn Sie vor die Aufgabe gestellt werden

> *Finde das Seminar, das 2500,00 € kostet,*

so finden Sie in den Beispieldaten der Seminartabelle zwei Seminare, auf die das zutrifft. Es spielt aber auch keine Rolle, ob Sie in den Beispieldaten welche finden. Um die Eindeutigkeit zu gewährleisten, darf nicht einmal die Möglichkeit bestehen, das zwei Seminare jeweils gleich viel kosten könnten. Natürlich lässt sich bei den Seminarpreisen die Möglichkeit nie ausschließen – deswegen ist {preis} nicht eindeutig und damit auch kein Schlüssel.

Irreduzibilität

Ein Schlüssel lässt sich nicht reduzieren.

Betrachten Sie den Schlüssel {seminar_titel, kapitel_nr , uebungs_nr} aus der Tabelle *uebungen*. Er ist eindeutig. Wenn Sie z. B. wissen, dass die Übung Nr. 3 aus Kapitel 4 des Seminars *Datenbanken & SQL* gemeint ist, so können Sie genau die Zeile mit der Übung bestimmen. Es gibt nur eine Zeile, zu der der Schlüssel passt – und es könnte auch keine weiteren geben.

Versuchen Sie nun, den Schlüssel zu reduzieren, indem Sie ihm ein Attribut wegnehmen. Sie erhalten folgende Mengen:

- {kapitel_nr , uebungs_nr}
- {seminar_titel, uebungs_nr}
- {seminar_titel, kapitel_nr}

Prüfen Sie, ob diese Mengen noch in der Lage sind, einen Datensatz eindeutig zu identifizieren.

- Identifiziert die Menge *{kapitel_nr , uebungs_nr}* eindeutig?

Wenn Sie kapitel_nr und uebungs_nr kennen so können verschiedene Übungen gemeint sein, da es in einem anderen Seminar die gleiche Kombination aus Kapitel- und Übungsnummer geben kann (z. B. Übung 4 aus Kapitel 1).

- Identifiziert die Menge *{seminar_titel, uebungs_nr}* eindeutig?

Wenn Sie das Seminar und die Übungsnummer kennen, hilft Ihnen das auch nicht weiter, da es im gleichen Seminar natürlich in jedem Kapitel eine Übung mit z. B. der Nummer 1 geben kann.

- Identifiziert die Menge *{seminar_titel, kapitel_nr}* eindeutig?

Aus Seminar und Kapitelnummer wiederum können Sie die genaue Übung in diesem Kapitel nicht identifizieren.

Welches Attribut Sie auch wegnehmen, die Eigenschaft der Eindeutigkeit geht verloren. Anders ausgedrückt: Die Menge *{seminar_titel, kapitel_nr, uebungs_nr}* lässt sich nicht reduzieren – ist also irreduzibel.

Ich will Ihnen ein Gegenbeispiel geben. Die Menge *{titel, preis}* ist eindeutig für die Tabelle seminare. Der *titel* alleine genügt aber schon. d. h. Sie können diese Menge um das Attribut *preis* reduzieren, ohne die Eindeutigkeit zu verlieren. Damit ist *{titel, preis}* nicht irreduzibel und somit auch kein Schlüssel.

Die Menge *{titel}* dagegen ist sicher irreduzibel. Sie könnten nur noch *titel* wegnehmen. Dann erhalten Sie aber die leere Menge. Mit der leeren Menge (also ohne jediglichen Anhaltspunkt) können Sie keinen Datensatz identifizieren.

Zum Abschluss möchte ich Ihnen noch eine (etwas vereinfachte) Definition zeigen:

Ein Schlüssel ist eine Teilmenge eines Headings, die

- (a) einen Datensatz[1] **eindeutig** identifiziert und sich
- (b) **nicht reduzieren** lässt.

---

1 streng genommen ein Tupel

11.2 Schlüssel finden

Haben Sie schon mal Ihren Haustürschlüssel verloren? Das ist äußerst unangenehm. Zum Glück kann Ihnen das bei der Datenbankentwicklung nicht passieren. Hier gibt es ein einfaches Verfahren, wie Sie Schlüssel finden.

1. Notieren Sie alle (echten) Teilmengen des Headings Ihrer Tabelle.

   Beispiel

   Die Tabelle *seminare* hat das Heading {titel, beschreibung, preis}. Folgende Teilmengen sind möglich:

   - {titel, beschreibung}
   - {titel, preis}
   - {beschreibung, preis}
   - {titel}
   - {beschreibung}
   - {preis}

2. Streichen Sie alle Teilmengen, die keine eindeutige Identifizierung von Datensätzen erlauben.

   Beispiel

   - {titel, beschreibung}
   - {titel, preis}
   - ~~{beschreibung, preis}~~
   - {titel}
   - ~~{beschreibung}~~
   - ~~{preis}~~

3. Streichen Sie zusätzlich alle Teilmengen, die sich reduzieren lassen.

   Beispiel

   - ~~{titel, beschreibung}~~
   - ~~{titel, preis}~~
   - {titel}

   Für *seminare* ist nur *{titel}* ein Schlüssel.

Es ist für den täglichen Gebrauch nicht notwendig, dass Sie alle Ihre Tabellen immer einer solchen Schlüsselprüfung unterziehen. Mit etwas Erfahrung sehen Sie meist die Schlüssel einer Tabelle auf den ersten Blick. Falls Sie sich unsicher sind, können Sie aber immer auf dieses Verfahren zurückgreifen.

11.3 Primär- und Alternativschlüssel

Betrachten Sie als Beispiel folgende Anforderungen:

Wir möchten dem Benutzer alternativ zur E-Mail noch die Möglichkeit geben, sich mit einem selbst gewählten Loginnamen anzumelden. Sobald sich ein Benutzer registriert, erhält er außerdem vom System automatisch eine eindeutige Kundennummer.

Diese Anforderungen könnten Sie z. B. mit folgenden Anpassungen der Benutzertabelle realisieren:

- benutzer = {name, vorname, registriert_seit, email, **kunden_nr**, **login**, passwort}

Welche Schlüssel hat nun die neue Tabelle?

Das es potentiell auch mehrere Kunden mit gleichem Vor- und Nachnamen geben kann (z. B. mehrere Michael Müller) ist {name, vorname} sicherlich kein Schlüssel. Den Anforderungen zufolge sind aber {email}[1], {login} und {kunden_nr} Schlüssel. In diesem Fall, haben Sie also gleich drei Schlüssel. In der Praxis hat es sich aber als sinnvoll herausgestellt, einen Schlüssel zu wählen und mit diesem hauptsächlich zu arbeiten – der sogenannte ***Primärschlüssel*** (***primary key***). Alle anderen Schlüssel, die Sie nicht wählen, werden als ***Alternativschlüssel*** (***alternate keys***) bezeichnet. Wenn Sie sich also beispielsweise für *{kunden_nr}* als Primärschlüssel entscheiden, so sind {email} und *{login}* Alternativschlüssel.

---

1 Im Moment gehen wir davon aus, dass jedem Benutzer genau eine E-Mail-Adresse zugeordnet ist.

Sie kennzeichnen den Primärschlüssel in der Mengeschreibweise, indem Sie alle Attribute des Schlüssels unterstreichen.

- benutzer = {name, vorname, registriert_seit, email, login, passwort, kunden_nr}
- uebungen = {seminar_titel, kapitel_nr, uebungs_nr, punkte}

Schlüssel sind Teil der Implementierung, aber nicht des allgemeinen Konzeptes einer Anwendung. Deswegen tauchen die Schlüssel im Domänenmodell nicht auf, im physischen Datenmodell jedoch schon. Markieren Sie dazu die Attribute, die Teil des Primärschlüssels sind mit dem UML-Stereotype *PK* (für Primary Key). Ein Stereotyp ist ein Zusatz, den Sie in doppelten spitzen Klammern hinterlegen – z. B.`<<PK>>`. Attribute, die Teil eines alternativen Schlüssels sind, werden entsprechend mit `<<AK>>` markiert. Falls Sie mehrerere Alternativschlüssel finden, müssen Sie diese durchnummerieren, `<<AK1>>`, `<<AK2>>`, usw.

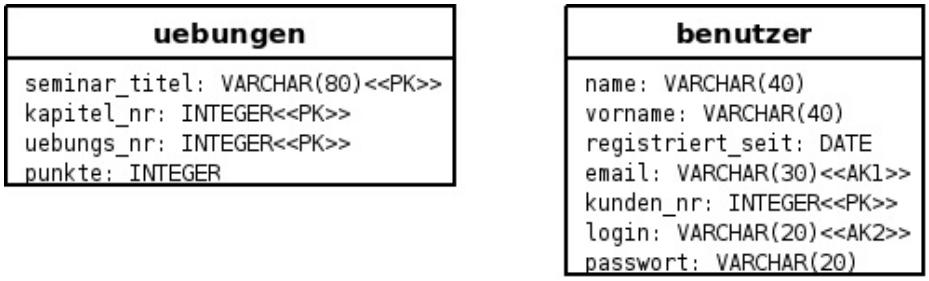


Abb. 11.1 Primärschlüssel und Alternativschlüssel in UML

Diese Anforderungen waren nur ein Beispiel. Sie müssen die Erweiterungen an der Benutzertabelle nicht durchführen. Im Folgenden verwenden Sie wieder die ursprüngliche Fassung der Tabelle.

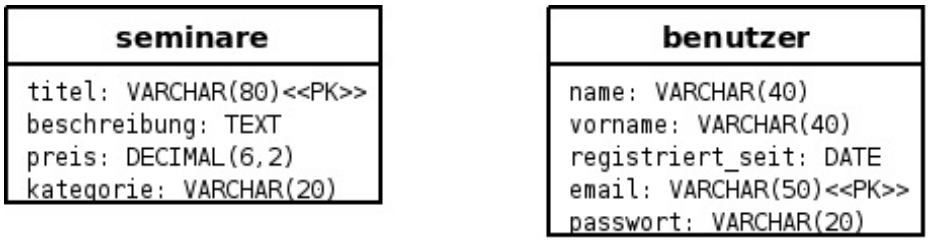


Abb. 11.2 Physisches Datenbankmodell mit Schlüsseln

11.4 Künstliche Schlüssel

11.4.1 Eigenschaften guter Primärschlüssel

Im obigen Beispiel haben Sie sich für *{kunden_nr}* als Primärschlüssel entschieden. Welche Kriterien sollten Sie aber generell für die Wahl des Primärschlüssels heranziehen? Was macht einen Schlüssel geeignet für die Wahl zum Primärschlüssel?

Aus Sicht der Datenbanktheorie sind alle Schlüssel gleichwertig. So gesehen ist es völlig gleichgültig, für welchen Schlüssel Sie sich entscheiden. In der Anwendungsentwicklung hat sich aber herausgestellt, dass es sehr wohl Unterschiede gibt. Durch die Wahl eines Primärschlüssels können Sie sich die Weiterentwicklung deutlich erleichtern oder erschweren, je nachdem, wie geeignet der Schlüssel ist.

Beispiel

Nehmen Sie mal an, Sie würden sich in der Tabelle Benutzer für {email} als Primärschlüssel entscheiden. Der Primärschlüssel wird später an vielen Stellen in der Anwendung zur Identifizierung verwendet. Ändert nun ein Benutzer seine E-Mail-Adresse, so entsteht ein erheblicher Wartungsaufwand. Alle Stellen, die sich anhand der E-Mail auf den Benutzer beziehen, müssten geändert werden. Die Kundennummer dagegen ist eine laufende Nummer, die jeder Benutzer automatisch zugewiesen bekommt und die sich normalerweise nicht ändern sollte.

Außerdem ist es einfacher, mit kurzen Schlüsseln zu arbeiten, als mit solchen, die aus vielen Attributen bestehen. Beispielsweise besteht der Schlüssel *{seminar_titel, kapitel_nr, uebungs_nr}* aus 3 Attributen, *{kunden_nr}* aber nur aus einem. Schlüssel, die aus mehreren Attributen bestehen, werden ***zusammengesetzte Schlüssel*** (***compound key***) genannt.

Deswegen lässt sich festhalten: Ein geeigneter Primärschlüssel[1]

---

1 Diese Aussagen gelten nicht unbedingt im Allgemeinen, sondern setzen bereits voraus, dass Sie später ORM einsetzen möchten. Eine detailiertere Begründung finden Sie bei Fowler & Rice (2002) beim Pattern *Identity Field*.

- ist kein zusammengesetzter Schlüssel.
- ist möglichst unveränderlich (***immutable***) über die Lebensdauer der zugehörigen Datensätze.

Martin sagt…

> *Die Gefahr eines bedeutungstragenden Schlüssels ist, dass obgleich sie in der Theorie gute Schlüssel abgeben, sie es in der Praxis nicht sind. Um überhaupt zu funktionieren müssen sie einzigartig sein; um gut zu funktionieren, dürfen sie sich nicht ändern.*
>
> Fowler & Rice (2002)

11.4.2 Tabellen ohne geeigneten Primärschlüssel

In der Tabelle *seminare* ist der einzige Schlüssel ***{titel}***. Im Grunde haben Sie, mangels Alternativen, gar keine andere Wahl, als ihn zum Primärschlüssel zu berufen. Prüfen Sie ihn dennoch auf Eignung.

- Er besteht aus nur einem Attribut, ist also nicht zusammengesetzt.
- Ist er auch stabil gegenüber Änderungen?

Fragen Sie doch Ihre Kundin.

Der Titel eines Seminars kann sich aus Marketingsgründen jederzeit ändern.

Offensichtlich ist *{titel}* nicht sonderlich gut als Primärschlüssel geeignet. In einem solchen Fall haben Datenbankentwickler schon immer etwas in der Trickkiste gekramt, und ein Konzept zu Tage gefördert, das sich ***künstlicher Schlüssel*** (***artificial key***, ***synthetic key***, ***surrogate key***) nennt. Der Trick besteht darin, dass Sie der Tabelle ein neues Attribut hinzufügen. Sie wählen das Attribut so, dass es die gewünschten Eigenschaften erfüllt.

Nennen Sie das Attribut *id*. Es hat den Datentyp `INTEGER` und enthält eine Zahl, die Sie einfach für jeden neuen Datensatz um eins erhöhen[1] .

---

1 Das ist nur eine mögliche Technik. Mehr dazu finden Sie bei Fowler & Rice (2002).

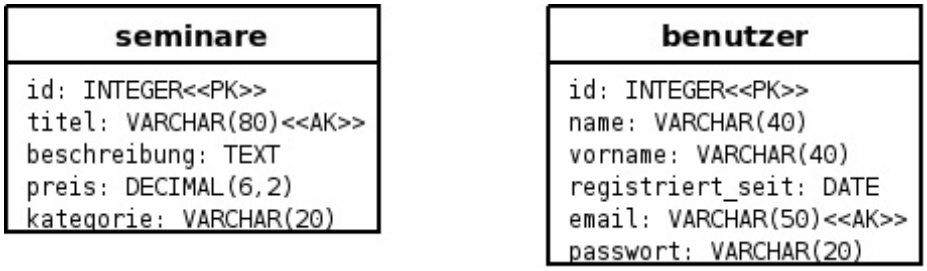


Abb. 11.3 Physisches Datenbankmodell mit ids

- seminare = {**id**, titel, beschreibung, preis}

id	titel	beschreibung	preis
1	Ruby on Rails	Ruby on Rails ist das neue, sensationelle OpenSource-Framework, das auf der modernen…	2500,00
2	Ajax & DOM	Ajax ist längst dem Hype-Stadium entwachsen. Mehr und mehr professionelle Websites folgen …	1699,99
3	JavaScript	JavaScript ist eine Programmiersprache mit vielseitigen Einsatzmöglichkeiten. Sie gilt…	2500,00
4	Datenbanken & SQL	Nahezu alle modernen Webapplikationen speichern ihren Datenbestand …	975,00

Tabelle 11.3 Tabelle *seminare* mit *id*

Scott sagt…

> *I rate Ihnen künstliche Schlüssel zu bevorzugen … Das fundamentale Problem ist, dass Schlüssel eine signifikante Quelle der Kopplung innerhalb des relationalen Schemas darstellen, und in Folge schwer zu ändern sind. DAS wiederum hat zur Folge, dass Sie Schlüssel mit Geschäftsbedeutung vermeinden wollen, denn Geschäftsbedeutungen ändern sich.*
>
> Ambler (2003)

11.4.3 Ids im objekt-relationalen Mapping

Wenn Sie später objekt-orientierten Code für Ihre Anwendung schreiben, ist es erforderlich, Code und Datenbank in Einklang zu bringen, indem Sie die verschiedenen Konzepte aufeinander abbilden. Das geschieht mit Hilfe von ***objekt-relationalem Mapping*** (kurz ***ORM***). In modernen ORM-Frameworks[1] wird meist das Muster ***Identity Field***[2] verwendet. Es ermöglicht eine 1:1-Zuordnung zwischen Objekten und Datensätzen. Damit das konsistent und zuverlässig funktioniert, versehen Sie einfach **jede** Tabelle mit einem Id-Attribut - selbst dann, wenn sie bereits einen brauchbaren Schlüssel (wie z. B. Kundennummer) besitzt.

---

1 z. B. Active Record oder Doctrine
2 siehe Fowler & Rice (2002)

Streitpunkt »künstlicher Schlüssel«

Ob künstliche Schlüssel (und insbesondere ein Id-Attribut) im Datenbankdesign sinnvoll sind oder nicht, ist umstritten. Beispielsweise vertritt Joe Celko (2005) die Ansicht, dass ein generelles id-Attribut, das automatisch erhöht wird, ganz schlechter SQL-Stil ist. Ich schließe mich an dieser Stelle jedoch der Meinung von Martin Fowler und vielen bekannten modernen Webframeworks, wie z. B. Ruby on Rails, Django und Symfony an. Diese haben gezeigt, dass das id-Attribut die Entwicklung objekt-orientierter, datenbank-basierter Webanwendungen (mit ORM) erheblich vereinfacht. Das gilt insbesondere unter den genannten Voraussetzungen (siehe Abschnitt 1.3). Für Datenbanken, die diese Voraussetzungen nicht erfüllen, mag Joe Celko recht behalten.

Scott sagt…

> *Schlüssel sind einer der religiösen Streitpunkte innerhalb der Daten-Community. Einige Leute bevorzugen komplett natürliche Schlüssel, wohingegen andere komplett künstliche vorziehen.*
>
> Ambler (2003)

Zusammenfassung

- Ein Schlüssel ist eine Teilmenge eines Headings, die (a) **eindeutig** identifiziert und sich (b) **nicht reduzieren** lässt.
- Der Primärschlüssel ist ein ausgewählter Schlüssel.
- Eine Relation kann auch mehrere Schlüssel haben.
- Verwenden Sie ein id-Attribut als Primärschlüssel.

11.5 Aufgaben zur Selbstkontrolle

Testen Sie Ihr Wissen

1. Erläutern Sie die Begriffe: ***eindeutig*** und ***irreduzibel***.
2. Geben Sie ein Beispiel für eine Teilmenge von *seminare = {titel, beschreibung, preis}*, die nicht eindeutig ist.
3. Geben Sie ein Beispiel für eine Teilmenge von *seminare = {titel, beschreibung, preis}*, die nicht irreduzibel ist.
4. Erläutern Sie die Begriffe: *Primärschlüssel* und *Alternativschlüssel*.
5. Was bedeutet die Unterstreichung bei folgender Menge?
   uebungen = {seminar_titel, kapitel_nr, uebungs_nr, punkte}
6. Wozu wird ein Identity Field benötigt?
7. Sind natürliche oder eher künstliche Schlüssel im Allgemeinen zu bevorzugen?
8. Kann eine Relation auch gar keine Schlüssel haben? Begründen Sie Ihre Antwort.

Übungen

Aufgabe 1: Teilmengen bestimmen

Bestimmen Sie alle echten Teilmengen der angegebenen Headings.

- {modell, hersteller, leistung_in_ps}
- {wert, farbe}
- {a, b, c}

Aufgabe 2: Schlüssel bestimmen

Bestimmen Sie **alle** Schlüssel der Tabelle *buch = {autor, erscheinungsdatum, isbn, titel}*. Mit isbn ist ausschließlich die ISBN-13 gemeint. Verwenden Sie dazu die vorgeschlagene Vorgehensweise aus Abschnitt 11.2. Falls Sie Zweifel bezüglich der Eindeutigkeit haben, können Sie folgende Beispiele heranziehen:

autor	erscheinungs-datum	isbn	titel
C.J. Date	01.09.2003	978-0321197849	Introduction to Database Systems
C.J. Date	15.11.2006	978-1590597460	Date on Database: Writings
C.J. Date	04.09.2001	978-0201787221	Introduction to Database Systems
T. Powell	14.07.2004	978-0072253573	JavaScript
D. Gosselin	07.09.2007	978-1423901501	JavaScript
S. Meier	01.01.2009	978-9999999991	Mathematik I

S. Meier	01.01.2009	978-9999999992	Mathematik I (Lösungsbuch)

Tabelle 11.1 Beispieldaten der Tabelle *buch*

Aufgabe 3: Filmverleih mit Schlüsseln

Nutzen Sie Ihr neu gewonnenes Wissen. Überarbeiten Sie das physische Modell des Projektes *Filmverleih* in Bezug auf Schlüssel. Kennzeichnen Sie Schlüssel und fügen Sie id-Attribute hinzu.

Zusatzübungen

Aufgabe 4: Fluggesellschaft mit Schlüsseln

Überarbeiten Sie auch das physische Model der Fluggesellschaft in Bezug auf Schlüssel.

Aufgabe 5: Partnervermittlung

Überarbeiten Sie nun auch noch das physische Modell der Partnervermittlung in Bezug auf Schlüssel.

12 Schlüssel in SQL

In dieser Lektion lernen Sie:

- wie Sie Primärschlüssel mit SQL kennzeichnen.
- wie Sie Alternativschlüssel mit SQL kennzeichnen.
- wie Sie die Vergabe von id-Werten automatisieren.

12.1 Primärschlüssel anlegen

Nachdem Sie nun wissen, dass die Tabelle *seminare* ein id-Attribut benötigt, sollten Sie dieses auch per SQL hinzufügen. In MySQL können Sie sowohl den Primärschlüssel als auch die Alternativschlüssel direkt beim Erzeugen einer Tabelle angeben. Zunächst müssen Sie die bestehende Tabelle nochmal löschen:

```
DROP TABLE seminare;
```

Jetzt können Sie die Tabelle mit der id als Primärschlüssel anlegen. Notieren Sie `PRIMARY KEY` einfach nach dem Datentyp von id:

CREATE TABLE seminare (id INTEGER PRIMARY KEY, titel VARCHAR(120), beschreibung TEXT, preis DECIMAL(6,2));

Prüfen Sie, ob MySQL die Tabelle korrekt angelegt hat:

```
SHOW COLUMNS FROM seminare;
```

```
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| id          | int(11)     | NO   | PRI | NULL    |       |
| titel       | varchar(120) | YES  |     | NULL    |       |
| beschreibung | text        | YES  |     | NULL    |       |
| preis       | decimal(6,2) | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
```

Beachten Sie die Spalte *Key*. Das Kürzel *PRI* gibt an, dass es sich bei der *id* um den Primärschlüssel handelt. Im Sinne der Integrität prüft MySQL nun, dass keine zwei Datensätze mit gleicher id eingetragen werden können. Versuchen Sie es!

```
INSERT INTO seminare (id, titel, beschreibung, preis) VALUES (1,
'Datenbanken & SQL', 'Nahezu alle modernen…', 975.00);
```

```
INSERT INTO seminare (id, titel, beschreibung, preis) VALUES (1, 'Ajax &
DOM', 'Ajax ist längst dem Hype-Stadium…', 1699.99);
```

Sie erhalten den Fehler:

```
ERROR 1062 (23000):
Duplicate entry '1' for key 1
```

Das bedeutet, dass die id *1* bereits vorhanden ist. Ein kurzes

```
SELECT * FROM seminare;
```

zeigt, dass MySQL nur den ersten Datensatz akzeptiert hat.

```
+----+------------------+-------------------------+--------+
| id | titel            | beschreibung            | preis  |
+----+------------------+-------------------------+--------+
| 1  | Datenbanken & SQL | Nahezu alle modernen… | 975.00 |
+----+------------------+-------------------------+--------+
```

12.2 Alternativschlüssel anlegen

Wie Sie schon wissen, ist {titel} ebenfalls ein Schlüssel. Es macht auch keinen Sinn, zwei Seminare mit gleichem Titel in der Datenbank zu führen. Deswegen sollten Sie *titel* als Alternativschlüssel kennzeichnen. In MySQL wird ein Alternativschlüssel mit `UNIQUE KEY` bezeichnet. Löschen Sie also nochmals die Tabelle und erzeugen Sie sie neu mit dem Titel als Alternativschlüssel:

```
DROP TABLE seminare;

CREATE TABLE seminare (id INTEGER PRIMARY KEY, titel VARCHAR(120) UNIQUE
KEY, beschreibung TEXT, preis DECIMAL(6,2));
```

Wenn Sie die Tabellenstruktur nun nochmals mit

```
SHOW COLUMNS FROM seminare;
```

analysieren, erhalten Sie folgende Ausgabe:

```
+--------------+--------------+------+-----+---------+-------+
| Field        | Type         | Null | Key | Default | Extra |
+--------------+--------------+------+-----+---------+-------+
| id           | int(11)      | NO   | PRI | NULL    |       |
| titel        | varchar(120) | YES  | UNI | NULL    |       |
| beschreibung | text         | YES  |     | NULL    |       |
| preis        | decimal(6,2) | YES  |     | NULL    |       |
+--------------+--------------+------+-----+---------+-------+
```

Sie sehen hier, dass *titel* in der Key-Spalte als UNI (Abkürzung für `UNIQUE`) gekennzeichnet ist. Daran können Sie in MySQL Alternativschlüssel erkennen. Wenn Sie nun versuchen, mehrere Seminare mit gleichem Titel einzufügen, erhalten Sie wiederum eine Fehlermeldung.

```
INSERT INTO seminare (id, titel, beschreibung, preis) VALUES (1,
'Datenbanken & SQL', 'Nahezu alle modernen…', 975.00);

INSERT INTO seminare (id, titel, beschreibung, preis) VALUES (2,
'Datenbanken & SQL', 'Nahezu alle modernen…', 975.00);
```

```
ERROR 1062 (23000): Duplicate entry 'Datenbanken & SQL' for key 2
```

Sie können im Übrigen immer nur einen Primärschlüssel, aber durchaus mehrere Alternativschlüssel in einer Tabelle haben. Deswegen ist z. B. folgende SQL-Anweisung syntaktisch korrekt:

```
CREATE TABLE seminare (id INTEGER PRIMARY KEY, titel VARCHAR(120) UNIQUE
KEY, beschreibung TEXT, preis DECIMAL(6,2) UNIQUE KEY);
```

Natürlich macht es wenig Sinn, *preis* als Alternativschlüssel zu definieren. Nicht einmal syntaktisch korrekt wäre folgendes SQL:

```
CREATE TABLE seminare (id INTEGER PRIMARY KEY, titel VARCHAR(120) UNIQUE
KEY, beschreibung TEXT, preis DECIMAL(6,2) PRIMARY KEY);
```

```
ERROR 1068 (42000):
Multiple primary key defined
```

12.3 Autoincrement

Recht unpraktisch ist momentan noch die Tatsache, dass Sie den Wert der id jedes mal angeben müssen.

id	titel	beschreibung	preis
1	Datenbanken & SQL	Nahezu alle modernen…	975,00
2	Ruby on Rails	Ruby on Rails ist das neue, sensation…	2500,00
3	Ajax & DOM	Ajax ist längst dem Hype-Stadium…	1699,99
4	JavaScript	JavaScript ist eine Programmier…	2500,00

Tabelle 12.1 Tabelle *seminare* mit *id*

Wollen Sie die Seminare aus obiger Tabelle einfügen, so sind folgende Inserts notwendig:

```
INSERT INTO seminare (id, titel, beschreibung, preis) VALUES ( 1 , 'Datenbanken & SQL', 'Nahezu alle
INSERT INTO seminare (id, titel, beschreibung, preis) VALUES ( 2 , 'Ruby on Rails', 'Ruby on Rails i
INSERT INTO seminare (id, titel, beschreibung, preis) VALUES ( 3 , 'Ajax & DOM', 'Ajax ist längst de
INSERT INTO seminare (id, titel, beschreibung, preis) VALUES ( 4 , 'JavaScript', 'JavaScript ist ein
```

Noch schwieriger wird es, wenn die Tabelle bereits Seminare enthält. In diesem Fall müssten Sie zuerst die höchste id in Erfahrung bringen.

Zum Glück hält MySQL eine praktische Erweiterung bereit. Sie können ein Attribut, das Primärschlüssel[1] ist und einen Zahlentyp besitzt, als ***autoinkrement***[2] definieren. Das bedeutet, dass jeder neue Datensatz automatisch den nächsthöheren ganzzahligen Wert erhält. Das ist quasi die ideale Ergänzung für ein id-Attribut. Definieren Sie die Tabelle *seminare* nun mit autoinkrement:

---

1 Genauer: Das Attribut muss das Einzige in einer Menge sein, die Primärschlüssel ist – z. B. ist das Attribut id in der Menge {id} alleine und {id} ist Primärschlüssel.
2 In anderen DBMS heißt dieses Feature oft auch ***Identity Column***.

```
DROP TABLE seminare;

CREATE TABLE seminare (id INTEGER PRIMARY KEY AUTO_INCREMENT, titel
VARCHAR(120) UNIQUE KEY, beschreibung TEXT, preis DECIMAL(6,2));
```

Führen Sie die übliche Prüfung durch:

```
SHOW COLUMNS FROM seminare;
```

```
+--------+--------------+------+-----+------+----------------+
| Field  | Type         | Null | Key | Def. | Extra          |
+--------+--------------+------+-----+------+----------------+
| id     | int(11)      | NO   | PRI | NULL | auto_increment |
| titel  | varchar(120) | YES  | UNI | NULL |                |
| beschr | text         | YES  |     | NULL |                |
| preis  | decimal(6,2) | YES  |     | NULL |                |
+--------+--------------+------+-----+------+----------------+
```

Die Spalte *Extra* zeigt an, dass *id* nun über die Eigenschaft *auto_increment* verfügt. Wenn Sie neue Seminare einfügen, müssen Sie das id-Attribut und den zugehörigen Wert nicht mehr angeben.

```
INSERT INTO seminare (titel, beschreibung, preis)
VALUES ('Datenbanken & SQL', 'Nahezu alle modern…', 975.00);
```

```
INSERT INTO seminare (titel, beschreibung, preis)
VALUES ('Ruby on Rails', 'Ruby on Rails ist…', 2500.00);
INSERT INTO seminare (titel, beschreibung, preis)
VALUES ('Ajax & DOM', 'Ajax ist längst de…', 1699.99);
INSERT INTO seminare (titel, beschreibung, preis)
VALUES ('JavaScript', 'JavaScript ist ein…', 2500.00);
```

Wenn Sie sich die Tabelle nochmals ausgeben lassen, sehen Sie, dass MySQL die ids automatisch erhöht hat.

```
SELECT * FROM seminare;
```

```
+----+------------------+------------------------+---------+
| id | titel            | beschreibung           | preis   |
+----+------------------+------------------------+---------+
| 1  | Datenbanken & SQL | Nahezu alle modern…  | 975.00  |
| 2  | Ruby on Rails     | Ruby on Rails ist…  | 2500.00 |
| 3  | Ajax & DOM        | Ajax ist längst de…  | 1699.99 |
| 4  | JavaScript        | JavaScript ist ein…  | 2500.00 |
+----+------------------+------------------------+---------+
```

12.4 Schlüssel in BNF

Die erweiterte Fassung der BNF von `CREATE TABLE` will ich Ihnen natürlich nicht vorenthalten.

```
CREATE TABLE tbl_name (create_definition,...)

create_definition:
  col_name data_type [AUTO_INCREMENT] [UNIQUE KEY | PRIMARY KEY]
```

Listing 12.1 BNF von CREATE TABLE mit Schlüsseln

Um diese Fassung zu verstehen, fehlen Ihnen noch zwei weitere BNF-Konzepte. Das erste Konzept ist die ***Option***. Alles was innerhalb eckiger Klammern »[]« steht, ist optional. Sie können `AUTO_INCREMENT` angeben oder auch weglassen.

Das zweite neue BNF-Konzept ist das ***Oder*** -Zeichen »|«. Mit dem Oder-Zeichen können Sie zwischen zwei oder mehreren Alternativen unterscheiden. In diesem Fall können Sie entweder `UNIQUE KEY` oder `PRIMARY KEY` angeben. Sie können auch gar keinen Key angeben, da beide Möglichkeiten zusammen in Klammern stehen.

Zusammenfassung

- Alternativschlüssel werden in MySQL mit `UNIQUE KEY` angegeben.
- Definieren Sie das id-Attribut als `PRIMARY KEY`.
- Verwenden Sie `AUTO_INCREMENT` für Ihre id-Attribute!

BNF

Konzept	Darstellung
Terminale	Wort in Großbuchstaben
Nichtterminale (Variablen)	Wort in Kleinbuchstaben
Wiederholung	…
Verschachtelte Definition	:
Optional	[]
Alternativ	|

Tabelle 12.2 Bisherige BNF-Konzepte

SQL: DDL

Zweck	BNF
Datenbank anlegen	`CREATE DATABASE db_name`
Datenbank löschen	`DROP DATABASE db_name`

Tabelle anlegen	`CREATE TABLE tbl_name (create_definition,...)`
	
	`create_definition:`
	`  col_name column_definition`
	`  | PRIMARY KEY (col_name,...)`
	`  | UNIQUE KEY (col_name,...)`
	
	`column_definition:`
	`  data_type [AUTO_INCREMENT] [UNIQUE KEY | PRIMARY KEY]`

Tabelle löschen	`DROP TABLE tbl_name`

Tabelle 12.3 Bisherige DDL-Anweisungen

12.5 Aufgaben zur Selbstkontrolle

Testen Sie Ihr Wissen

1. Wie viele Spalten in einer Tabelle können Sie als PRIMARY KEY deklarieren und wie viele als UNIQUE KEY?
2. Welche Voraussetzungen muss eine Spalte erfüllen, damit Sie sie mit AUTO_INCREMENT versehen können?

Übungen

Aufgabe 1: BNF ausprägen

Prägen Sie folgenden BNF-Ausdruck mit 2 Beispielen aus:

ICH WAR GESTERN IN lokalität. DORT HABE ICH personen GETROFFEN.
lokalität: name_der_lokalität IN ort personen: person, … person: vorname nachname [AUS ort]

Aufgabe 2: Projekt »Filmverleih«: Schlüssel anlegen mit SQL

Löschen Sie die bestehenden Tabellen des Filmverleihs. Erzeugen Sie sie erneut, wobei Sie dieses Mal die Schlüssel mit angeben. Verwenden Sie AUTO_INCREMENT, falls möglich.

Zusatzübungen

Aufgabe 3: Projekt »Fluggesellschaft«: Schlüssel anlegen mit SQL

Löschen und erzeugen Sie auch die Tabellen der Fluggesellschaft mit Schlüsseln.

Aufgabe 4: Projekt »Partnervermittlung«: Schlüssel anlegen mit SQL

Löschen und erzeugen Sie nun auch noch die Tabellen der Partnervermittlung mit Schlüsseln.

13 SQL: Ändern bestehender Tabellen

In dieser Lektion lernen Sie:

- wie Sie bei bestehenden Tabellen Spalten hinzufügen.
- wie Sie Spalten löschen.
- wie Sie Spalten ändern.
- wie Sie Tabellen umbenennen.

It really really sucks if you got some DBA Group and trying to get a new column to a table requires to sacrifice three virgins and one of your fingers.

Martin Fowler, at RailsConf 2006

13.1 DDL-Änderungen ohne Datenverlust

Nehmen Sie an, dass Sie bereits eine kleine Admin-Oberfläche in PHP entwickelt haben. Sie zeigen Ihrer Kundin das Ergebnis. Sie ist schlichtweg begeistert und verspricht die Webanwendung mit ihren Kollegen gleich ausgiebig zu testen. Schon zwei Tage später meldet sie sich mit folgendem Feedback:

Das Anlegen von Benutzern funktioniert hervorragend. Wir haben schon alle Mitarbeiter als Benutzer angelegt. Dabei ist uns aber aufgefallen, dass die Anrede – also z. B. Herr oder Frau – noch fehlt. Können Sie das Feld noch ergänzen? Die Benutzerdaten möchten wir aber nicht nochmal eingeben müssen. Geht das?

Ach ja, und bei den Seminaren würden wir gerne noch eine Kategorie mit angeben, z. B. »Programmierung« oder »Webdesign«.

Bisher haben Sie eine Tabelle bei Änderungen an der Datenstruktur immer gelöscht und neu angelegt. Das hat aber den Verlust der bereits eingegebenen Daten zur Folge. In diesem Fall wird Ihre Kundin das nicht tolerieren. Zum Glück gibt es in SQL die Möglichkeit, bestehende Tabellen nachträglich zu verändern. Ändern heißt auf englisch *alter*. `ALTER` heißt dementsprechend auch der passende SQL-Befehl.

Zunächst sollten Sie die Klassendiagramme für das Domänenmodell und das physische Datenmodell aktualisieren:

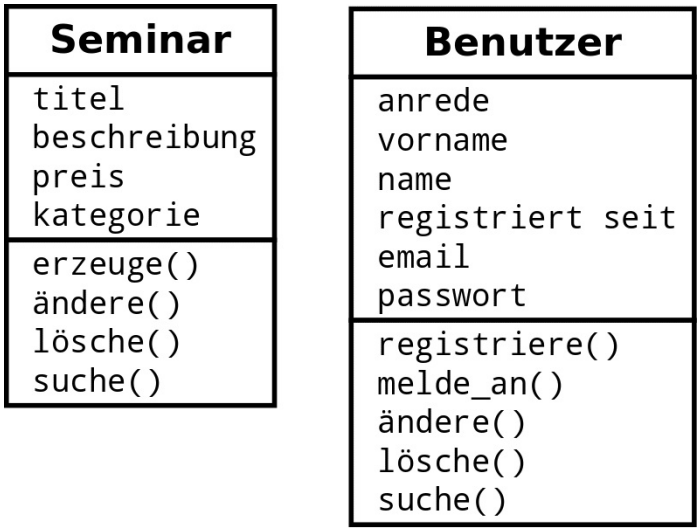


Abb. 13.1 Domänenmodell mit Anrede

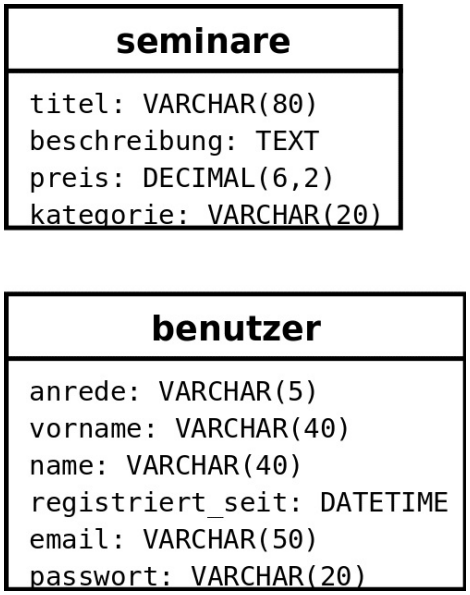


Abb. 13.2 Physisches Datenmodell mit Anrede

13.2 Spalten hinzufügen mit ADD

Fügen Sie die neue Spalte nun mit Hilfe von SQL in die Tabelle *benutzer* ein:

```
ALTER TABLE benutzer ADD anrede VARCHAR(5);
```

Ein anschließendes `SHOW COLUMNS FROM benutzer;` zeigt das geänderte Tabellenschema:

```
+-----------------+------------+------+-----+---------+-------+
| Field           | Type       | Null | Key | Default | Extra |
+-----------------+------------+------+-----+---------+-------+
| vorname         | varchar(40) | YES  |     | NULL    |       |
| name            | varchar(40) | YES  |     | NULL    |       |
| email           | varchar(50) | YES  |     | NULL    |       |
| passwort        | varchar(20) | YES  |     | NULL    |       |
| registriert_seit| date        | YES  |     | NULL    |       |
| anrede          | varchar(5)  | YES  |     | NULL    |       |
+-----------------+------------+------+-----+---------+-------+
```

MySQL versucht bei ALTER-Anweisungen, so wenige Daten wie möglich zu verlieren. Beim Hinzufügen neuer Spalten entsteht überhaupt kein Datenverlust. Es sind noch alle Benutzer vorhanden, wie Sie mittels `SELECT * FROM benutzer;` sehen können:

```
+----+---------+-------+-------+----------+------------+-------+
| id | vorname | name  | email | passwort | registr… | anrede|
+----+---------+-------+-------+----------+------------+-------+
| 1  | Frank   | Reich | f.r.. | kochtopf | 2008-04-12 | NULL  |
| 2  | Marie   | Huana | hua.. | reibek.. | 2009-02-03 | NULL  |
| 3  | Andreas | Mei.. | a.m.. | schüssel | 2008-07-15 | NULL  |
+----+---------+-------+-------+----------+------------+-------+
```

Die BNF zum Hinzufügen von Spalten lautet übrigens:

```
ALTER TABLE tbl_name alter_specification [,alter_specification] ...

alter_specification: ADD col_name column_definition
```

Listing 13.1 BNF zum Hinzufügen von Spalten

13.3 Spalten löschen mit DROP

Sie können Spalten natürlich auch löschen. Fügen Sie zum Ausprobieren erstmal eine überflüssige hinzu:

```
ALTER TABLE benutzer ADD unnoetige_spalte INTEGER;
```

Prüfen Sie, ob sie wirklich da ist:

```
SHOW COLUMNS FROM benutzer;
```

```
+------------------+-------------+-...
| Field            | Type        | ...
+------------------+-------------+-...
| vorname          | varchar(40) | ...
| name             | varchar(40) | ...
| email            | varchar(50) | ...
| passwort         | varchar(20) | ...
| registriert_seit | date        | ...
| anrede           | varchar(5)  | ...
| unnoetige_spalte | int(11)     | ...
+------------------+-------------+-...
7 rows in set (0.01 sec)
```

Löschen Sie die Spalte nun wieder mit:

```
ALTER TABLE benutzer DROP unnoetige_spalte;
```

`SHOW COLUMNS FROM benutzer;` zeigt, dass sie entfernt wurde.

```
+------------------+-------------+-...
| Field            | Type        | ...
+------------------+-------------+-...
| vorname          | varchar(40) | ...
| name             | varchar(40) | ...
| email            | varchar(50) | ...
| passwort         | varchar(20) | ...
| registriert_seit | date        | ...
| anrede           | varchar(5)  | ...
+------------------+-------------+-...
6 rows in set (0.01 sec)
```

Beim Löschen einer Spalte verlieren Sie natürlich auch deren Daten. Alle anderen Daten bleiben aber erhalten. Überprüfen Sie das mit `SELECT * FROM benutzer;` wenn Sie möchten.

Hier noch die BNF:

```
ALTER TABLE tbl_name alter_specification [,alter_specification] ...

alter_specification: DROP col_name
```

Listing 13.2 BNF zum Löschen von Spalten

13.4 Spalten ändern oder umbenennen mit CHANGE

Wollen Sie eine bestehende Spalte ändern, so können Sie den `CHANGE` -Befehl verwenden. Beispielsweise lässt sich die Anzahl der Zeichen, die für eine Anrede vorgesehen ist, auf 10 vergrößern – mittels:

```
ALTER TABLE benutzer CHANGE anrede anrede VARCHAR(10);
```

Prüfen Sie wie immer das Ergebnis:

```
SHOW COLUMNS FROM benutzer;
```

```
+-----------------+------------+-...
| Field           | Type       | ...
+-----------------+------------+-...
| vorname         | varchar(40) | ...
| name            | varchar(40) | ...
| email           | varchar(50) | ...
| passwort        | varchar(20) | ...
| registriert_seit | date       | ...
| anrede          | varchar(10) | ...
+-----------------+------------+-...
6 rows in set (0.00 sec)
```

Sie haben *anrede* übrigens deswegen zweimal angegeben, weil `CHANGE` es Ihnen ermöglicht, die Spalte auch gleich umzubennen. Der zweite Name ist dabei der neue. Da Sie den gleichen Spaltennamen *anrede* nochmals angegeben haben, erfolgt keine Umbenennung.

Hier ist noch die BNF:

```
ALTER TABLE tbl_name alter_specification [,alter_specification] ...

alter_specification: CHANGE old_col_name new_col_name column_definition
```

Listing 13.3 BNF zum Ändern/Umbenennen von Spalten

`column_definition` bietet hier die gleichen Möglichkeiten wie beim `CREATE TABLE`, d. h. Sie können z. B. auch Schlüssel angeben.

13.5 Tabellen umbenennen mit RENAME

Falls Sie die Tabelle selbst umbenennen möchten, ist das ebenfalls mit `ALTER` möglich. Um z. B. *seminare* in *kurse* umzubenennen, geben Sie einfach ein:

```
ALTER TABLE seminare RENAME TO kurse;
```

Prüfen Sie das Ergebnis mit `SHOW tables;`

```
+----------------------------+
| Tables_in_seminarverwaltung |
+----------------------------+
| benutzer                   |
| kurse                      |
+----------------------------+
2 rows in set (0.00 sec)
```

Benennen Sie sie bitte wieder in *seminare* zurück, bevor Sie weiterarbeiten:

```
ALTER TABLE kurse RENAME TO seminare;
```

Hier noch, wie üblich, die BNF:

```
ALTER TABLE tbl_name alter_specification [,alter_specification] ...

alter_specification: RENAME TO new_tbl_name
```

Listing 13.4 BNF zum Umbenennen von Tabellen

13.6 BNF von ALTER

Die gezeigten ALTER -Varianten lassen sich zu folgender BNF zusammenfassen[1]:

1 Es gibt noch weitere Möglichkeiten, die in der Praxis aber eher selten benötigt werden oder fortgeschrittene Datenbankfunktionen betreffen, die dieses Buch nicht behandelt. Sie finden die Gesamt-BNF von ALTER TABLE unter dev.mysql.com/doc/refman/5.1/en/alter-table.html.

```
ALTER TABLE tbl_name alter_specification [, alter_specification] ...
alter_specification:
  ADD col_name column_definition
  | DROP col_name
  | CHANGE old_col_name new_col_name column_definition
  | RENAME TO new_tbl_name
```

Listing 13.5 BNF von ALTER

Zusammenfassung

- Mit ALTER -Anweisungen können Sie das Tabellenschema ändern, ohne Daten zu verlieren.

SQL: DDL

Zweck	BNF
Datenbank anlegen	`CREATE DATABASE db_name`
Datenbank löschen	`DROP DATABASE db_name`

Tabelle anlegen	`CREATE TABLE tbl_name (create_definition,...)`
	
	`create_definition:`
	`  col_name column_definition`
	`  | PRIMARY KEY (col_name,...)`
	`  | UNIQUE KEY (col_name,...)`
	
	`column_definition:`
	`  data_type [AUTO_INCREMENT] [UNIQUE KEY | PRIMARY KEY]`

Tabelle löschen	`DROP TABLE tbl_name`

Tabellenschema ändern	`ALTER TABLE tbl_name alter_specification [, alter_specification] ...`
	
	`alter_specification:`
	`  ADD col_name column_definition`
	`| DROP col_name`
	`| CHANGE old_col_name new_col_name column_definition`
	`| RENAME TO new_tbl_name`

Tabelle 13.1 DDL-Anweisungen

13.7 Aufgaben zur Selbstkontrolle

Testen Sie Ihr Wissen

1. Was unterscheidet `CHANGE` und `ALTER`?
2. Kann `ALTER` einen Datenverlust bewirken?

Übungen

Aufgabe 1: Spalte hinzufügen

Fügen Sie der Tabelle *seminare* das neue Attribut *kategorie* hinzu, das es erlaubt, ein Seminar einer Kategorie wie z. B. *Programmierung* oder *Webdesign* zuzuordnen.

Aufgabe 2: Korrekturen

Gegeben ist die Tabelle *buecher* mit folgender CREATE-Anweisung:

```
CREATE TABLE buecher (titel VARCHAR(30), autor VARCHAR(80), klappentext
TEXT, seiten INTEGER);
```

Korrigieren Sie die Tabelle mit Hilfe von ALTER-Anweisungen, bis sie folgendem Schema entspricht:

```
CREATE TABLE romane (id INTEGER PRIMARY KEY, titel VARCHAR(50), autor
VARCHAR(80), seitenzahl INTEGER);
```

Aufgabe 3: Korrekturen am Projekt »Filmverleih«

Ergänzen Sie die Tabelle *filme* um das Attribut *genre*. Erhöhen Sie die maximale Länge eines Titels auf 150 Zeichen.

Zusatzübungen

Aufgabe 4: Korrekturen am Projekt »Fluggesellschaft«

Führen Sie mit Hilfe von ALTER wenigstens zwei Korrekturen am Datenbankschema der Fluggesellschaft durch. Versuchen Sie das Schema dadurch zu verbessern. Falls Sie keine Ideen dazu haben, fügen Sie einfach ein neues Attribut hinzu und verkleinern Sie einen der VARCHARs.

Aufgabe 5: Korrekturen am Projekt »Partnervermittlung«

Verbessern Sie nun auch zwei Aspekte der Partnervermittlung.

14 Migrationen

In dieser Lektion lernen Sie:

- wie Sie SQL-Skripte verwenden.
- wie Sie Migrationen durchführen.
- wie Sie sich einen Satz Beispieldaten zum Testen halten können.

14.1 Versionieren des Datenbankschemas

In der Praxis hinterlegen Sie Ihre Anwendung immer auf mehreren Systemen. Beispielsweise programmieren Sie die Seminarverwaltung auf Ihrem eigenen Arbeitsplatzrechner, müssen sie aber später auf dem Server Ihres Kunden (der Akademie) installieren. Für die meisten Projekte empfiehlt sich außerdem noch ein Testsystem, auf dem Sie die jeweils neuesten Features unter möglichst realistischen Bedingungen testen können.

Dabei müssen Sie stets sicherstellen, dass das Datenbankschema zum aktuellen Stand der Anwendung passt. Betrachten Sie dazu folgendes Beispiel:

> *Auf dem Server der Akademie war zu Testzwecken schon der alte Stand der Anwendung installiert. Nun haben Sie die Spalte anrede in der benutzer-Tabelle hinzugefügt. Den PHP-Code Ihrer Anwendung haben Sie erweitert, so dass für jeden Benutzer die Anrede ausgewählt werden muss.*
>
> *Wenn Sie diese Änderung auf den Server der Akademie übertragen, so dürfen Sie nicht vergessen, die Spalte Anrede auch in der dortigen Datenbank anzulegen, d. h. Sie müssen die exakt gleiche `ALTER`-Anweisung auch auf dem Akademie-Server absetzen. Dabei könnten Sie sich aber vertippen und beispielsweise die Spalte falsch benennen. In diesem Fall würden Sie einen Fehler in die Anwendung einbringen, der sich nur auf diesem Server nachvollziehen lässt. In der Praxis ist es oft unklar, was den Fehler verursacht – das Datenbankschema, der Anwendungscode oder möglicherweise die Konfiguration des Servers? Solche Fehler sind schwer zu finden. Sobald die Anwendung produktiv ist, müssten Sie zur Fehlersuche womöglich sogar den Server vom Netz nehmen.*

Glücklicherweise können Sie das Problem von Anfang an vermeiden. Dazu benötigen Sie sogenannte *Migrationsskripte*.

Die Idee dabei ist, dass Sie alle DDL-Anweisungen nicht direkt ausführen, sondern sie stattdessen zunächst in einer Datei speichern. Dadurch können Sie die Anweisungen bei Ihrem Projekt speichern und sie auf jedem anderen Rechner wieder exakt gleich durchführen. Sie vermeiden damit Fehler und sparen sich außerdem den Aufwand, sich die Anweisungen neu zu überlegen (falls schon etwas Zeit vergangen ist) und mehrmals zu tippen.

Probieren Sie es am Beispiel der Seminarverwaltung:

1. Löschen Sie die Datenbank und erzeugen Sie sie erneut.
2. Legen Sie sich einen Ordner namens *Migrations* an.
3. Legen Sie eine neue Datei unter dem Namen `001_erzeuge_benutzer_und_seminare.sql` an.
4. Schreiben Sie das DML zum Erzeugen Ihrer Datenbankstruktur in diese Datei:

```
CREATE TABLE seminare (
id INTEGER PRIMARY KEY AUTO_INCREMENT,
titel VARCHAR(120),
```

```
beschreibung TEXT,
preis DECIMAL(6,2));

CREATE TABLE benutzer (
id INTEGER PRIMARY KEY AUTO_INCREMENT,
vorname VARCHAR(40),
name VARCHAR(40),
email VARCHAR(50) UNIQUE,
passwort VARCHAR(20),
registriert_seit DATE);
```

Listing 14.1 001_erzeuge_benutzer_und_seminare.sql

Da die Datei aus einer Reihe von SQL-Anweisungen besteht, wird sie als SQL-Skript bezeichnet. In diesem Fall handelt es sich außerdem um ein Migrationsskript.

5. Als Nächstes führen Sie dass Skript in der Datenbank `serminarverwaltung` aus. Wechseln Sie dazu in der Konsole in das Migration-Verzeichnis und geben Sie ein:

   ```
   mysql -u root seminarverwaltung < 001_erzeuge_benutzer_und_seminare.sql
   ```

   Durch das *kleiner-als* -Symbol »<« leiten Sie die Zeilen der Datei direkt an MySQL um. Das entspricht der Eingabe der Anweisungen direkt im MySQL-Client.

6. Überprüfen Sie mit Hilfe von SHOW-Anweisungen, ob MySQL die Tabellen korrekt erzeugt hat.

   Dieses Vorgehen mag Ihnen zunächst vielleicht etwas umständlich erscheinen. Es hat aber den Vorteil, dass Sie damit sicherstellen, dass alle an Ihrem Projekt beteiligten Rechner über exakt das gleiche Datenbankschema verfügen.

   Wenn Sie nun Änderungen durchführen wollen, schreiben Sie die Anweisungen einfach in ein weiteres Migrationsskript.

7. Legen Sie dazu im Migrations-Verzeichnis eine Datei unter dem Namen `002_ergaenze_anrede_und_kategorie.sql` an. Hinterlegen Sie dort folgende Zeilen:

   ```
   ALTER TABLE benutzer ADD anrede VARCHAR(10);
   ALTER TABLE seminare ADD kategorie VARCHAR(10);
   ```

   Listing 14.2 002_ergaenze_anrede_und_kategorie.sql

8. Geben Sie auf der Konsole ein:

   ```
   mysql -u root seminarverwaltung < 002_ergaenze_anrede_und_kategorie.sql
   ```

9. Prüfen Sie wieder, ob die Änderung erfolgreich war.

Wenn Sie Ihre zusammengehörigen Änderungen in Migrationsskripten hinterlegen, können Sie genau die gleichen Änderungen auch auf jedem anderen Rechner durchführen. Sie müssen dazu lediglich wissen, welchen Versionsstand die Datenbank der jeweilige Rechner hat – daher die spezielle Benennung der Dateien. Die Zahl am Anfang des Dateinamens ist die Versionsnummer[1]. Sie erhöhen sie bei jedem Skript einfach um eins. Jetzt müssen Sie lediglich noch dokumentieren, welcher Rechner welche Version Ihres Datenbankschemas verwendet. Sie wissen dadurch immer, welche Migrationsskripte Sie auf dem jeweiligen Rechner noch ausführen müssen. Für den zweiten Teil des Dateinamens überlegen Sie sich einen sinnvollen Kommentar, mit dem Sie auf den ersten Blick erkennen können, was das Skript tut.

1 Alternativ können Sie auch statt Versionsnummern Datum und Uhrzeit verwenden, z. B. `200904141145_ergaenze_benutzer_um_anrede.sql` für den 14.04.2009 um 11:45.

Neben den reinen Strukturdaten können Sie durchaus auch Datensätze mit Hilfe von Migrationsskripten einpflegen. Sie sollten aber nur Daten einpflegen, die in jeder Installation der Anwendung benötigt werden. Beispielsweise wäre es nicht sinnvoll, Kursdaten einzupflegen, falls die Anwendung später von mehreren Niederlassungen der Akademie genutzt wird und diese jeweils andere Kurse anbieten. Sinnvollerweise könnten Sie aber z. B. schon mal einen Administrator-Benutzer anlegen – also einen Benutzer, der später über alle Zugriffsrechte in der Anwendung verfügt. Das Passwort müsste der jeweils verantwortliche Administrator natürlich später ändern.

14.2 Beispieldaten

Wenn Sie Ihre Anwendung testen und weiterentwickeln, ist es hilfreich, einen Satz an Beispieldaten parat zu haben. Es ist jedoch nicht ratsam, Beispieldaten direkt in die Datenbank einzupflegen. Sie werden vielleicht später zu Testzwecken Daten löschen oder unsinnige Daten eintragen. Haben Sie die Daten manuell eingepflegt, so müssen Sie Ihren Datenbestand nach dem Testen wieder mit viel Aufwand reparieren.

Abhilfe schaffen auch hier SQL-Skripte. Legen Sie pro Tabelle je ein Skript an:

- `seminare.sql`
- `benutzer.sql`

In jedem Skript halten Sie die dazugehörigen Beispieldatensätze in Form von INSERT-Anweisungen fest. Falls Sie Ihren Datenbestand beim Testen beschädigen sollten, können Sie die Skripte jederzeit wieder einspielen.

1. Legen Sie nun die Skripte an[1]:

1 Sie finden das Skript selbstverständlich im Begleitmaterial zu diesem Buch.

```
INSERT INTO seminare (id, titel, beschreibung, preis, kategorie)
VALUES (1, 'Datenbanken & SQL', 'Nahezu alle modernen…', 975.00, 'Datenbanken');
INSERT INTO seminare (id, titel, beschreibung, preis, kategorie)
VALUES (2, 'Ruby on Rails', 'Ruby on Rails ist das neue, sensation…', 2500.00, 'Programmierung'
INSERT INTO seminare (id, titel, beschreibung, preis, kategorie)
VALUES (3, 'Ajax & DOM', 'Ajax ist längst dem Hype-Stadium…', 1699.99, 'Programmierung');
INSERT INTO seminare (id, titel, beschreibung, preis, kategorie)
VALUES (4, 'JavaScript', '...gilt als DIE Programmiersprache…', 2500.00, 'Programmierung');
INSERT INTO seminare (id, titel, beschreibung, preis, kategorie)
VALUES (5, 'Adobe Flash (Grundlagen)', 'Adobe Flash bringt voll animierte…', 1500.00, 'Webdesig
INSERT INTO seminare (id, titel, beschreibung, preis, kategorie)
VALUES (6, 'Adobe Flash (ActionScript)', 'Für anspruchsvolle Flash-Präsentationen und…', 1500.0
INSERT INTO seminare (id, titel, beschreibung, preis, kategorie)
VALUES (7, 'Bildbearbeitung mit Adobe PS', 'In diesem Seminar lernen Sie die Grundlagen…', 1500
```

Listing 14.3 seminare.sql

```
INSERT INTO benutzer (id, anrede, name, vorname, registriert_seit, email, passwort)
VALUES (1, 'Herr', 'Reich', 'Frank', '2008-04-12', 'f.reich@example.com', 'kochtopf');
INSERT INTO benutzer (id, anrede, name, vorname, registriert_seit, email, passwort)
VALUES (2, 'Frau', 'Huana', 'Marie', '2009-02-03', 'huana@example.com', 'reibekuche');
INSERT INTO benutzer (id, anrede, name, vorname, registriert_seit, email, passwort)
VALUES (3, 'Herr', 'Meisenbär', 'Andreas', '2008-07-15', 'a.meisenbär@example.com', 'schüssel')
```

Listing 14.4 benutzer.sql

2. Führen Sie sie aus:

```
mysql -u root seminarverwaltung < seminare.sql
mysql -u root seminarverwaltung < benutzer.sql
```

Mit Hilfe der SQL-Skripte können Sie nun jederzeit die Datenbank löschen und mit aktuellen Testdaten wiederherstellen. Außerdem können Sie die Skripte bei Ihrem Projekt speichern und haben somit alle relevanten Daten an einem Ort.

Migrationen in objekt-relationalen Mappern

Manche OR-Mapper wie z. B. Doctrine (PHP) oder ActiveRecord (Ruby) bieten von Haus aus Unterstützung für Migrationsskripte an. Wenn Sie einen solchen Mapper

einsetzen, können Sie viele Arbeitsvorgänge mit Hilfe des Mappers automatisieren, z. B. das Verwalten der Versionsnummern. Sie sparen somit wertvolle Entwicklungszeit.

Zusammenfassung

- SQL-Skripte enthalten Anweisungen, die Sie sonst auf der Konsole eingeben würden.
- Mit Hilfe von Migrationsskripten können Sie Aufwände einsparen und Fehler vermeiden.
- Ein SQL-Skript mit Beispieldaten ist nützlich zum Testen Ihrer Anwendung.

14.3 Aufgaben zur Selbstkontrolle

Übungen

Aufgabe 1: SQL-Skripte für das Projekt »Filmverleih«

1. Entwickeln Sie ein Migrationsskript, das die Tabellen des Projekts anlegt.
2. Entwickeln Sie je ein SQL-Skript, das Beispieldaten für die Tabellen *filme* und *regisseure* einfügt. Verwenden Sie dazu die Beispieldaten aus Lektion 10.
3. Löschen Sie die Datenbank *filmverleih* und erzeugen Sie sie erneut.
4. Führen Sie die Skripte aus.
5. Prüfen Sie die erzeugten Tabellen und Daten.

Zusatzübungen

Aufgabe 2: SQL-Skripte für das Projekt »Fluggesellschaft«

1. Entwickeln Sie ein Migrationsskript, das die Tabellen des Projekts anlegt.
2. Entwickeln Sie je ein SQL-Skript, das Beispieldaten in alle Tabellen einfügt. Verwenden Sie auch hier Ihre Beispieldaten aus Lektion 10.

Aufgabe 3: SQL-Skripte für das Projekt »Partnervermittlung«

Entwickeln Sie die Skripte nach dem gleichen Muster wie in den vorherigen Übungen.

15 SQL: Datensätze gezielt auslesen

In dieser Lektion lernen Sie:

- wie Sie Spalten einzeln selektieren.
- wie Sie doppelte Datensätze entfernen.
- wie Sie bestimmte Datensätze gezielt abfragen.

15.1 SELECT mit Spaltenangabe

Oftmals benötigen Sie in Ihrer Anwendung verschiedene Sichten auf Ihre Daten. Betrachten Sie dazu folgende Anforderung:

Wir möchten neben der Adminsicht auch eine öffentliche Benutzerliste integrieren. Diese soll einfach alle Benutzer tabellarisch untereinander auflisten. E-Mail-Adresse und Passwort dürfen aber nicht angezeigt werden.

Diese Anforderung ist leicht zu realisieren. Sie benötigen dazu lediglich die `SELECT` - Anweisung. Statt wie bisher

```
SELECT * FROM benutzer;
```

zu verwenden, bestimmen Sie nun genau, welche Spalten Sie benötigen. Geben Sie statt »*« einfach die benötigten Spalten an:

```
SELECT vorname, name, registriert_seit FROM benutzer;
```

Sie erhalten folgende Ergebnistabelle:

```
+---------+------------+------------------+
| vorname | name       | registriert_seit |
+---------+------------+------------------+
| Frank   | Reich      | 2008-04-12       |
| Marie   | Huana      | 2009-02-03       |
| Andreas | Meisenbär  | 2008-07-15       |
+---------+------------+------------------+
3 rows in set (0.00 sec)
```

Der Stern »*«, den Sie bisher verwendet haben, steht für alle Spalten der Ursprungstabelle. Sie erhalten mit dem Stern die gleiche Ergebnistabelle, als ob Sie alle Spalten direkt angäben.

```
SELECT vorname, name, email, passwort, registriert_seit FROM benutzer;
```

Falls Sie tatsächlich alle Spalten benötigen, ist der Stern aber einfach praktischer.

15.2 Doppelte Datensätze entfernen

Damit sich die Kunden über unser Seminarangebot informieren können, benötigen wir noch eine Übersichtsseite der Seminare für die Kunden. Eine Liste mit Titel und Preis genügt. Die Beschreibung lässt sich auf eine Detailseite auslagern.

Allerdings sind es ziemlich viele Seminare. Deswegen soll der Besucher der Website die Möglichkeit haben, eine Kategorie zu wählen. Es gibt also zunächst eine Übersichtsseite, die alle Seminare zeigt und eine Navigation für die verschiedenen Kategorien. Wenn der Besucher eine Kategorie anklickt, kommt er auf eine Seite, die nur die Seminare der jeweiligen Kategorie listet.

Realisieren Sie zunächst die Übersichtsseite. Für die Übersichtsseite benötigen Sie zwei SQL-Anweisungen. Die erste Anweisung listet die Seminare. Das dürfte mittlerweile kein Problem mehr für Sie sein. Überlegen Sie bitte erst einmal selbst, bevor Sie weiterlesen.

Die Anweisung lautet einfach:

```
SELECT titel, preis FROM seminare;
```

Die zweite Anweisung soll die Kategorien auflisten. Wie listen Sie nun alle Kategorien auf? Versuchen Sie folgendes:

```
SELECT kategorie FROM seminare;
```

Gefällt Ihnen das Ergebnis?

```
+----------------+
| kategorie      |
+----------------+
| Datenbanken    |
| Programmierung |
| Programmierung |
| Programmierung |
| Webdesign      |
| Programmierung |
| Webdesign      |
+----------------+
7 rows in set (0.00 sec)
```

Wenn die Navigation einigermaßen sinnvoll aussehen soll, darf sie jede Kategorie nur einmal auflisten. Aber Moment, die Ergebnistabelle besitzt doch nur eine Spalte. Dürfen hier überhaupt mehrfach gleiche Zeilen auftreten?

Christopher sagt…

> *Es gibt ein Vielzahl praktischer Argumente, die die Ansicht unterstützen, dass doppelte Tupel (kurz: Dublikate) zu vermeiden sind.*
>
> Date (2005)

Nun, im Sinne des relationalen Modells ist das eigentlich verboten. SQL ist eben leider nicht perfekt.

Sie können das Problem zum Glück leicht umgehen. Teilen Sie dem DBMS gezielt mit, dass Sie nur Zeilen in Ihrer Ergebnistabelle haben möchten, die sich voneinander unterscheiden. *Voneinander verschieden* heißt *distinct*. Verbessern Sie die SQL-Anweisung:

```
SELECT DISTINCT kategorie FROM seminare;
```

```
+----------------+
| kategorie      |
+----------------+
| Datenbanken    |
| Programmierung |
| Webdesign      |
+----------------+
3 rows in set (0.01 sec)
```

15.3 Zeilen filtern

Realisieren Sie nun die Seite für die Anzeige aller Seminare einer speziellen Kategorie, beispielsweise der Kategorie *Programmierung*. Dazu benötigen Sie eine Bedingung. Auf Deutsch könnten Sie sagen:

- *Liste alle Seminare, deren Kategorie Programmierung ist.*

oder anderes ausgedrückt,

- *Liste alle Seminare, für die gilt: die Kategorie ist Programmierung.*

Genauso können Sie es in SQL formulieren. *Für die gilt* drücken Sie in SQL durch `WHERE` aus. Schreiben Sie also:

```
SELECT titel, preis FROM seminare WHERE kategorie = 'Programmierung';
```

Da *Programmierung* ein String ist, müssen Sie ihn in Anführungszeichen setzen. Dabei ist es nicht erforderlich, dass Sie die Spalte *kategorie* schon im SELECT-Teil angeben. Hier das Ergebnis:

```
+---------------------------+---------+
| titel                     | preis   |
+---------------------------+---------+
| Ruby on Rails             | 2500.00 |
| Ajax & DOM                | 1699.99 |
| JavaScript                | 2500.00 |
| Adobe Flash (ActionScript) | 1500.00 |
+---------------------------+---------+
4 rows in set (0.00 sec)
```

Die Bedingung hinter dem WHERE wird auf jede Zeile angewendet. Nur die Zeilen, auf die die Bedingung zutrifft, werden in der Ergebnistabelle aufgenommen. In diesem Fall eben alle, deren Kategorie *Programmierung* ist.

Die Bedingung ist dabei immer ein boole'scher Ausdruck. *Eine Bedingung trifft zu* bedeutet, dass der Ausdruck `TRUE` zurück gibt.

Beachten Sie, dass der Operator »=« in SQL der Gleichheitsoperator ist und nicht der Zuweisungsoperator, wie in vielen Programmiersprachen.

15.4 Vergleichsoperatoren

Neben dem Gleichheitsoperator gibt es auch in SQL die bekannten Vergleichsoperatoren:

Operator	Name	Beispiele
<	kleiner als	3 < 4 ergibt TRUE, 4 < 3 ergibt FALSE
>	größer als	4 > 3 ergibt TRUE, 3 > 4 ergibt FALSE
<=	kleiner oder gleich	3 <= 4 ergibt TRUE, 3<= 3 ergibt TRUE
>=	größer oder gleich	3 >= 4 ergibt FALSE, 3 >= 3 ergibt TRUE
!=	ungleich	3 != 3 ergibt FALSE, 3 != 4 ergibt TRUE

Tabelle 15.1 Vergleichsoperatoren

Wenn Sie beispielsweise alle Seminare anzeigen möchten, die weniger als 2000 € kosten, können Sie folgende SQL-Anweisung verwenden:

```
SELECT titel, preis FROM seminare WHERE preis < 2000;
```

```
+------------------------------+---------+
| titel                        | preis   |
+------------------------------+---------+
| Datenbanken & SQL            | 975.00  |
| Ajax & DOM                   | 1699.99 |
| Adobe Flash (Grundlagen)     | 1500.00 |
| Adobe Flash (ActionScript)   | 1500.00 |
| Bildbearbeitung mit Adobe PS | 1500.00 |
+------------------------------+---------+
5 rows in set (0.01 sec)
```

Neben Zahlen können Sie aber auch Strings und Datumsangaben in Vergleichen verwenden:

```
SELECT name, vorname, registriert_seit FROM benutzer WHERE
registriert_seit<'2009-01-01';
```

Diese SQL-Anweisung gibt Ihnen alle Benutzer zurück, die sich vor dem 01.01.2009 registriert haben.

```
+------------+---------+-----------------+
| name       | vorname | registriert_seit |
+------------+---------+-----------------+
| Reich      | Frank   | 2008-04-12       |
| Meisenbär  | Andreas | 2008-07-15       |
+------------+---------+-----------------+
2 rows in set (0.00 sec)
```

15.5 Suche mit LIKE

Denken Sie auch noch an die Suchfunktion? Auf der Übersichtsseite der Seminare muss es ein Suchfeld geben. Wenn der Benutzer dort z. B. »Adobe« eingibt, werden anschließend alle Seminare aufgelistet, bei denen »Adobe« im Titel vorkommt, z. B. »Adobe Flash (Grundlagen)« oder »Bildbearbeitung mit Adobe PS«.

Zur Entwicklung der Suchfunktion benötigen Sie einen weiteren Vergleichsoperator – den Operator `LIKE`. Mit seiner Hilfe können Sie einen Vergleich mit einem Muster durchführen. Ein Muster ist ein String, der sogenannte ***Wildcards*** (Platzhalter, Jokerzeichen) enthält. Davon gibt es zwei verschiedene. Das Zeichen »_« ist ein Wildcard, der für **ein** beliebiges Zeichen steht. Das Zeichen »%« steht sogar für mehrere beliebige Zeichen.

Beispiel

- `'Marco' LIKE 'Marc_'` ergibt TRUE
- `'Marci' LIKE 'Marc_'` ergibt TRUE
- `'Marc' LIKE 'Marc_'` ergibt FALSE
- `'Marc' LIKE 'Marc%'` ergibt TRUE
- `'Adobe Flash (Grundlagen)' LIKE 'Adobe%'` ergibt TRUE
- `'Bildbearbeitung mit Adobe PS'' LIKE 'Adobe%'` ergibt FALSE
- `'Bildbearbeitung mit Adobe PS' LIKE '%Adobe%'` ergibt TRUE
- `'Datenbanken & SQL' LIKE '%Adobe%'` ergibt FALSE

Wenn Sie alle Seminare auflisten möchten, die *Adobe* enthalten, schreiben Sie:

```
SELECT titel, preis, kategorie FROM seminare WHERE titel LIKE '%Adobe%';
```

```
+-----------------------------+---------+----------------+
| titel                       | preis   | kategorie      |
+-----------------------------+---------+----------------+
| Adobe Flash (Grundlagen)    | 1500.00 | Webdesign      |
| Adobe Flash (ActionScript)  | 1500.00 | Programmierung |
| Bildbearbeitung mit Adobe PS | 1500.00 | Webdesign      |
+-----------------------------+---------+----------------+
3 rows in set (0.00 sec)
```

`'%Adobe%'` bedeutet, der *titel* darf vor und nach `Adobe` beliebig viele beliebige Zeichen enthalten – oder anders ausgedrückt: Der *titel* enthält den String `Adobe`. Sie können das `Adobe` in `'%Adobe%'` später in der Programmierung durch eine Variable ersetzen, die die Suchanfrage des Benutzers enthält.

15.6 Logische Operatoren

Die Suchfunktion macht ja schon einen netten Eindruck. Wir würden dem Benutzer aber auch gerne die Möglichkeit geben, innerhalb einer Kategorie zu suchen. Wenn er z. B. in der Kategorie »Programmierung« den Suchbegriff »Adobe« eingibt, sollen nur Programmier-Seminare gelistet werden, die »Adobe« im Titel haben, z. B. »Adobe Flash (ActionScript)«. Können Sie das bitte noch ergänzen?

Um diese kleine Zusatzanforderung zu erfüllen und auch jede Kategorieunterseite mit einem entsprechenden Suchfeld zu versehen, müssen Sie in der SQL-Anweisung zwei Bedingungen angeben. Die Ergebnistabelle enthält ein Seminar, wenn es

- in der richtigen Kategorie ist und
- sein Titel zum Suchmuster passt.

Genauso wie ich die beiden obigen Sätze im Deutschen mit *und* verbunden habe, müssen Sie in Ihrer SQL-Anweisung die zwei Bedingungen mit `AND` verbinden. Geben Sie also ein:

```
SELECT titel, preis FROM seminare WHERE kategorie = 'Programmierung' AND
titel LIKE '%Adobe%';
```

```
+---------------------------+---------+
| titel                     | preis   |
+---------------------------+---------+
| Adobe Flash (ActionScript) | 1500.00 |
+---------------------------+---------+
1 row in set (0.00 sec)
```

`AND` ist der übliche logische Operator, der nur dann TRUE zurück gibt, wenn seine beiden Operanden TRUE sind – d. h. nur Seminare, auf die tatsächlich beide Bedingungen zutreffen, werden in der Ergebnistabelle gelistet. Hier treffen beide Bedingungen nur auf das Seminer *Adobe Flash (ActionScript)* zu.

Der `AND` -Operator lässt sich auch als sogenannte ***Wahrheitstabelle***[1] darstellen. Dabei entsprechen die Variablen `a` und `b` beliebigen logischen Aussagen, wie z. B. `kategorie = 'Programmierung'`.

---

1 siehe de.wikipedia.org/wiki/Wahrheitstabelle

a	b	a AND b
FALSE	FALSE	FALSE
FALSE	TRUE	FALSE
TRUE	FALSE	FALSE

TRUE	TRUE	TRUE

Tabelle 15.2 Wahrheitstabelle des AND-Operators

Der nächste logische Operator, den Sie benötigen, ist `OR`. Er gibt TRUE zurück, sobald einer seiner beiden Operatoren TRUE ist – d. h., der eine **oder** der andere. Sie können `OR` z. B. verwenden, um gleichzeitig in Titel und Beschreibung der Seminare zu suchen:

```
SELECT titel, beschreibung FROM seminare WHERE titel LIKE '%JavaScript%' OR
beschreibung LIKE '%JavaScript%';
```

```
+------------+-------------------------------------+
| titel      | beschreibung                        |
+------------+-------------------------------------+
| Ajax & DOM | Ajax ist längst…JavaScript ist…  |
| JavaScript | ...gilt als DIE Programmiersprach… |
+------------+-------------------------------------+
2 rows in set (0.01 sec)
```

Die Ergebnistabelle zeigt alle Seminare, die *JavaScript* im Titel oder in der Beschreibung haben – oder auch in beiden Spalten.

Hier sehen Sie zur Verdeutlichung die Wahrheitstabelle von `OR`.

a	b	a OR b
FALSE	FALSE	FALSE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	TRUE

Tabelle 15.3 Wahrheitstabelle des OR-Operators

Schließlich gibt es noch den logischen Operator `NOT`. Mit `NOT` können Sie einen Wahrheitswert invertieren. Beispielsweise könnten Sie alle Seminare ausgeben, die **nicht** zur Kategorie Programmierung gehören:

```
SELECT titel, kategorie, preis FROM seminare WHERE NOT kategorie =
'Programmierung';
```

```
+-----------------------------+-------------+---------+
| titel                       | kategorie   | preis   |
+-----------------------------+-------------+---------+
| Datenbanken & SQL           | Datenbanken | 975.00  |
| Adobe Flash (Grundlagen)    | Webdesign   | 1500.00 |
| Bildbearbeitung mit Adobe PS | Webdesign   | 1500.00 |
+-----------------------------+-------------+---------+
3 rows in set (0.00 sec)
```

Der NOT-Operator hat nur einen Operanden, wie Sie der Wahrheitstabelle leicht entnehmen können:

a	NOT a
FALSE	TRUE
TRUE	FALSE

Tabelle 15.4 Wahrheitstabelle des NOT-Operators

15.7 BNF von SELECT

```
SELECT [DISTINCT] * | col_name [,col_name…]
FROM table_name
[WHERE where_condition]
```

Listing 15.1 BNF von SELECT

Die `where_condition` ist ein beliebiger Ausdruck, der einen boole'schen Wert zurückliefert. Ein Ausdruck ist dabei eine beliebige Kombination aus Literalen, Spaltennamen, Operatoren und Funktionen. Später lernen Sie noch Funktionen und weitere Operatoren kennen. Die BNF von SELECT ist auch noch nicht ganz vollständig. Wir werden die Möglichkeiten von SELECT bald ausweiten.

15.8 SELECT-FROM-WHERE-Block

Die Kombination aus `SELECT`, `FROM` und `WHERE` wird oft auch als *SELECT-FROM-WHERE-Block* bezeichnet, weil die meisten SELECT-Anweisungen in der Praxis aus diesen drei Teilen bestehen. Als kleine Hilfestellung können Sie sich merken, dass Sie nach dem `SELECT` die Spalten und nach dem `WHERE` die Zeilen Ihrer Ergebnistabelle auswählen. Nach dem `FROM` geben Sie die Tabelle an, aus der Sie Daten entnehmen möchten.

Zusammenfassung

- Hinter `SELECT` können Sie Spalten auswählen, hinter `WHERE` Zeilen filtern.
- Doppelte Datensätze entfernen Sie mit `DISTINCT`.
- Eine Bedingung ist ein boole'scher Ausdruck.
- Sie können in SQL Ausdrücke mit logischen Operatoren und Vergleichsoperatoren verwenden.

SQL: DML

Zweck	BNF

Datensätze auslesen	`SELECT [DISTINCT] * | col_name [,col_name…]`
	`FROM table_name`
	`[WHERE where_condition]`

Datensatz einfügen	`INSERT INTO tbl_name (col_name,...) VALUES (value,...)`

Tabelle 15.5 BNF von DML-Anweisungen

15.9 Aufgaben zur Selbstkontrolle

Übungen

Aufgabe 1: Anfragen an den Filmverleih

Um aussagekräftigere Ausgaben zu erhalten, importieren Sie bitte die Daten aus `mehr_filme.sql`.

1. Geben Sie alle Filme des Filmverleihs mit allen Spalten aus.
2. Geben Sie alle Filme mit *titel* und *erscheinungsjahr* aus.
3. Geben Sie alle Jahre aus, aus denen der Filmverleih Filme besitzt (ohne Doppelnennungen).
4. Geben Sie alle Filme aus, die 2002 oder nach 2002 erschienen sind.
5. Geben Sie alle Filme aus, die den Text *Ring* im Titel enthalten.
6. Geben Sie alle Filme aus, die den Text *Ring* im Titel enthalten und nicht vor 2002 erschienen sind.
7. Geben Sie alle Filme aus, die den Text *Ring* im Titel oder der Kurzbeschreibung enthalten.

Aufgabe 2: Passwortprüfung

Gegeben ist folgende Benutzertabelle:

```
CREATE TABLE benutzer (id INTEGER PRIMARY KEY AUTO_INCREMENT, login
VARCHAR(30), passwort VARCHAR(30));
```

Entwickeln Sie für diese Tabelle eine SQL-Anweisung, die einen Benutzernamen (login) und ein Passwort entgegen nimmt. Für das Beispiel können Sie als Login *mhuana* und als Passwort *geheim* verwenden. Existiert der Benutzer und ist sein Passwort richtig, so soll die Anweisung die id zurückliefern. Existiert der Benutzer nicht oder ist das Passwort falsch, so soll die Ergebnistabelle leer sein.

16 Ausdrücke in SQL

In dieser Lektion lernen Sie:

- wie Sie Ausdrücke in SQL angeben.
- wie Sie Berechnungen durchführen.
- wie Sie die Auswertungsreihenfolge beeinflussen.
- wie Sie Spalten der Ergebnistabelle umbenennen.

16.1 Ausdrücke im SELECT

In der letzten Lektion haben Sie erfahren, dass Sie hinter dem Schlüsselwort SELECT die Spalten angeben können, die in Ihrer Ergebnistabelle erscheinen sollen. Tatsächlich können Sie aber noch wesentlich mehr tun. Die Angabe hinter dem SELECT bestimmt – ganz allgemein – das Format der Ergebnistabelle. Sie können dabei aber nicht nur bestehende Spalten aus einer Quelltabelle übernehmen, sorn auch völlig neue Spalten erzeugen. Betrachten Sie beispielsweise folgende Anforderung:

Wir benötigen noch dringend eine Auflistung der Seminare, die Netto- und Bruttopreis getrennt ausweist.

Diese kleine Anforderung Ihrer Auftraggeberin können Sie mit SQL problemlos umsetzen. Geben Sie hinter dem `SELECT` die geforderten Spalten an. Für den Bruttopreis, der in der Quelltabelle *seminare* nicht vorhanden ist, erzeugen Sie einfach eine neue Spalte. Die Werte der Spalte berechnen Sie nach Bedarf. Der Bruttopreis ergibt sich dabei aus dem jeweiligen Preis des Seminars mal 1,19.

```
SELECT titel, preis, preis * 1.19 FROM seminare;
```

```
+-----------------------------+---------+--------------+
| titel                       | preis   | preis * 1.19 |
+-----------------------------+---------+--------------+
| Datenbanken & SQL           | 975.00  | 1160.2500    |
| Ruby on Rails               | 2500.00 | 2975.0000    |
| Ajax & DOM                  | 1699.99 | 2022.9881    |
| JavaScript                  | 2500.00 | 2975.0000    |
| Adobe Flash (Grundlagen)    | 1500.00 | 1785.0000    |
| Adobe Flash (ActionScript)  | 1500.00 | 1785.0000    |
| Bildbearbeitung mit Adobe PS | 1500.00 | 1785.0000    |
+-----------------------------+---------+--------------+
7 rows in set (0.02 sec)
```

Prinzipiell können Sie hinter dem `SELECT` mehrere Ausdrücke angeben, deren Wert pro Zeile berechnet wird. Ein ***Ausdruck*** ist dabei eine Kombination von Literalen, Spaltennamen, Operatoren und Funktionen. Eine Spalte der Quelltabelle können Sie dabei ähnlich einsetzen wie eine Variable in einer Programmiersprache. MySQL ersetzt für jede Zeile den Namen der Spalte durch den Wert des jeweiligen Datensatzes.

Beispiel

```
SELECT 3 FROM seminare;
```

```
+---+
| 3 |
+---+
| 3 |
| 3 |
| 3 |
| 3 |
| 3 |
| 3 |
```

```
| 3 |
+---+
7 rows in set (0.00 sec)
```

```
SELECT 3 * 4 FROM seminare;
```

```
+-------+
| 3 * 4 |
+-------+
| 12    |
..
| 12    |
+-------+
7 rows in set (0.00 sec)
```

```
SELECT 3 * preis FROM seminare;
```

```
+-----------+
| 3 * preis |
+-----------+
| 2925.00   |
| 7500.00   |
| 5099.97   |
| 7500.00   |
| 4500.00   |
| 4500.00   |
| 4500.00   |
+-----------+
7 rows in set (0.00 sec)
```

16.2 Ausdruck ohne Quelltabelle

Es ist dabei nicht einmal zwingend erforderlich, eine Quelltabelle anzugeben. In dem Fall wird der Ausdruck nur einmal ausgewertet. Es gibt dann keine Zeilen, auf die er sich beziehen könnte.

Beispiel

```
SELECT 3;
```

```
+---+
| 3 |
+---+
| 3 |
+---+
1 row in set (0.00 sec)
```

```
SELECT 3 * 4;
```

```
+-------+
| 3 * 4 |
+-------+
| 12    |
+-------+
1 row in set (0.00 sec)
```

Diese Möglichkeit werden Sie vermutlich selten praktisch einsetzen, es ist aber sehr nützlich, um schnell und einfach Ausdrücke in MySQL auszuprobieren.

16.3 Ausdrücke im WHERE-Teil

Natürlich können Sie auch im WHERE-Teil beliebige Ausdrücke verwenden. Entscheidend ist dabei, dass SQL den Rückgabewert des Gesamtausdrucks als bool'schen Wert versteht. Für jede Zeile prüft das DBMS den Gesamtwert. Ist dieser `TRUE`, wird die Zeile für die Auswertung berücksichtigt.

Beispiel

```
SELECT titel, preis FROM seminare WHERE preis > 2 * 1000;
```

```
+---------------+---------+
| titel         | preis   |
+---------------+---------+
| Ruby on Rails | 2500.00 |
| JavaScript    | 2500.00 |
+---------------+---------+
2 rows in set (0.00 sec)
```

16.4 Operatoren

16.4.1 Arithmetische Operatoren

Zunächst gibt es in MySQL die klassischen arithmetischen Operatoren, wie folgende Tabelle zeigt.

Operator	Bedeutung	Beispiel	Ergebnis
+	Addition	8 + 3	11
-	Subtraktion	8 – 3	5
*	Multiplikation	8 * 3	24
/	Division	8 / 3	2.6667
DIV	Integerdivision	8 DIV 3	2

Tabelle 16.1 Arithmetische Operatoren

Beispiel

```
SELECT 3 * 4 DIV 5;
```

```
+-------------+
| 3 * 4 DIV 5 |
+-------------+
| 2           |
+-------------+
1 row in set (0.00 sec)
```

16.4.2 Logische und Vergleichsoperatoren

Neben den arithmetischen Operatoren gibt es noch die Vergleichsoperatoren und die logischen Operatoren, die Sie bereits kennen gelernt haben. Sie können Sie aber nicht nur im WHERE-, sondern auch im SELECT-Bereich verwenden.

Beispiel

```
SELECT 8 > 3;
```

```
+-------+
| 8 > 3 |
+-------+
| 1     |
+-------+
1 row in set (0.00 sec)
```

```
SELECT TRUE AND FALSE;
```

```
+----------------+
| TRUE AND FALSE |
+----------------+
| 0              |
+----------------+
1 row in set (0.00 sec)
```

```
SELECT 1 = 1 OR 2 != 3;
```

```
+-------------+
| 1=1 OR 2!=3 |
+-------------+
| 1           |
+-------------+
1 row in set (0.00 sec)
```

Wichtig ist hier, dass MySQL keinen echten bool'schen Datentyp besitzt, sondern `1` statt `TRUE` und `0` statt `FALSE` zurückgibt. Die Rückgabewerte sind vom Typ INTEGER und können sowohl als bool'sche Werte als auch als Integerwerte weiterverwendet werden. Letzteres macht allerdings selten Sinn.

Beispiel

```
SELECT (8 > 3) * 4;
```

```
+-------------+
| (8 > 3) * 4 |
+-------------+
| 4           |
+-------------+
1 row in set (0.00 sec)
```

16.4.3 Präzedenz und Klammerung

Wie in den meisten Programmiersprachen gibt es auch in SQL Präzedenzregeln. Durch Klammerung können Sie die Reihenfolge der Auswertung verändern.

Beispiel

```
SELECT 3 + 4 * 2;
```

```
+-----------+
| 3 + 4 * 2 |
+-----------+
| 11        |
+-----------+
1 row in set (0.00 sec)
```

```
SELECT (3 + 4) * 2;
```

```
+-------------+
| (3 + 4) * 2 |
+-------------+
| 14          |
+-------------+
1 row in set (0.00 sec)
```

16.5 Funktionen

Auch in SQL gibt es Funktionen, die Sie als Teil eines Ausdrucks einsetzen können. Es ist sicherlich nicht sinnvoll, alle Funktionen, die MySQL bietet, auswendig zu lernen. Deswegen zeige ich Ihnen nur einige wenige. Was zählt, ist das Prinzip. Bei Bedarf können Sie weitere Funktionen in der MySQL-Dokumentation[1] nachschlagen.

---

1 dev.mysql.com/doc/refman/5.4/en/functions.html

Funktionen haben Parameter und Rückgabewerte. Wie bei vielen Programmiersprachen befinden sich die Parameter in Klammern und werden durch Komma getrennt. Es gibt natürlich auch Funktionen ohne Parameter.

Beispiel

Die Funktion `ROUND` rundet eine Zahl auf eine angegebene Zahl von Nachkommastellen.

```
SELECT ROUND(2.715, 2);
```

```
+-----------------+
| ROUND(2.715, 2) |
+-----------------+
| 2.72            |
+-----------------+
1 row in set (0.00 sec)
```

Die Funktion `UPPER` konvertiert einen String in Großbuchstaben.

```
SELECT UPPER(titel) FROM seminare;
```

```
+------------------------------+
| UPPER(titel)                 |
+------------------------------+
| DATENBANKEN & SQL            |
| RUBY ON RAILS                |
| AJAX & DOM                   |
| JAVASCRIPT                   |
| ADOBE FLASH (GRUNDLAGEN)     |
| ADOBE FLASH (ACTIONSCRIPT)   |
| BILDBEARBEITUNG MIT ADOBE PS |
+------------------------------+
7 rows in set (0.00 sec)
```

Ganz ohne Parameter arbeitet `NOW`. Bei einem Aufruf von NOW() erhalten Sie das aktuelle Datum inklusive Uhrzeit zurück.

```
SELECT NOW();
```

```
+---------------------+
| NOW()               |
+---------------------+
| 2009-09-09 12:32:41 |
+---------------------+
1 row in set (0.02 sec)
```

16.5.1 Verschachteln von Funktionen

Die Rückgabewerte von Funktionen lassen sich problemlos von anderen Funktionen und Operatoren als Parameter weiterverwerten. Auf diese Weise können Sie Funktionen leicht verschachteln. Dazu ein kleines Beispiel aus der Praxis:

Unsere Administratoren hätten gerne die Möglichkeit, sich schnell einen Überblick über die Benutzer zu verschaffen. Dazu hätten Sie gerne eine Liste, die die Benutzer in folgendem Format anzeigt:

```
Nachname, erster Buchstabe des Vornamens.
```

Also z. B. Reich, F. statt Frank Reich.

Mit Hilfe einer Kombination von Stringfunktionen können Sie dieser Forderung leicht nachkommen. Zunächst benötigen Sie die Möglichkeit, den ersten Buchstaben des Vornamens zu extrahieren. Das ermöglicht die Funktion `SUBSTR`. Sie hat folgende Parameter:

1. den Eingabestring
2. die Position, ab der Sie den Substring aus dem Eingabestring extrahieren möchten
3. die Anzahl der Zeichen, die Sie benötigen

Da Sie genau ein Zeichen ab der ersten Position benötigen, funktioniert folgender Aufruf:

```
SELECT SUBSTR(vorname, 1, 1) FROM benutzer;
```

```
+----------------------+
| SUBSTR(vorname, 1, 1) |
+----------------------+
| F                    |
| M                    |
| A                    |
| K                    |
| M                    |
| B                    |
+----------------------+
6 rows in set (0.00 sec)
```

Anschließend können Sie mit Hilfe von `CONCAT` (verbinden) die Verbindung zwischen den einzelnen Stringbestandteilen herstellen:

```
SELECT CONCAT(name, ', ', SUBSTR(vorname, 1, 1), '.') FROM benutzer;
```

```
+-----------------------------------------------+
| CONCAT(name, ', ', SUBSTR(vorname, 1, 1), '.') |
+-----------------------------------------------+
| Reich, F.                                     |
| Huana, M.                                     |
| Meisenbär, A.                                 |
| Uhr, K.                                       |
| Rosoft, M.                                    |
| Lödmann, B.                                   |
+-----------------------------------------------+
6 rows in set (0.00 sec)
```

16.6 Umbenennen von Spaltenüberschriften

Schauen Sie sich nochmal das Beispiel vom Anfang der Lektion an:

```
SELECT titel, preis, preis * 1.19 FROM seminare;
```

```
+------------------+---------+--------------+
| titel            | preis   | preis * 1.19 |
+------------------+---------+--------------+
| Datenbanken & SQL | 975.00  | 1160.2500    |
...
+------------------+---------+--------------+
7 rows in set (0.02 sec)
```

Ein kleine Unschönheit stellen die Tabellenüberschriften dar. Zum einen ist unklar, dass es sich bei *preis* um den Nettopreis handelt. Zum anderen ist der Spaltenname *preis * 1.19* sehr unhandlich. Für solche Fälle hält SQL das Schlüsselwort AS bereit. Es dient zum Umbenennen von Spaltenüberschriften. Notieren Sie es einfach hinter einem Ausdruck. Das folgende Beispiel zeigt wie:

Beispiel

```
SELECT titel, preis AS netto, preis * 1.19 AS brutto FROM seminare;
```

```
+------------------------------+---------+-----------+
| titel                        | netto   | brutto    |
+------------------------------+---------+-----------+
| Datenbanken & SQL            | 975.00  | 1160.2500 |
| Ruby on Rails                | 2500.00 | 2975.0000 |
| Ajax & DOM                   | 1699.99 | 2022.9881 |
| JavaScript                   | 2500.00 | 2975.0000 |
| Adobe Flash (Grundlagen)     | 1500.00 | 1785.0000 |
| Adobe Flash (ActionScript)   | 1500.00 | 1785.0000 |
| Bildbearbeitung mit Adobe PS | 1500.00 | 1785.0000 |
+------------------------------+---------+-----------+
7 rows in set (0.00 sec)
```

16.7 BNF von SELECT

An dieser Stelle können Sie die BNF von SELECT ein wenig erweitern. SELECT lässt nicht nur Spaltennamen (`col_name`) zu, sondern auch beliebige Ausdrücke. Die MySQL-Dokumentation bezeichnet diese als SELECT-Expression (`select_expr`), da sie sich von Ausdrücken im WHERE-Bereich unterscheiden. Außerdem können Sie nun Spalten mit AS umbenennen[1] .

---

1 Beachten Sie, dass die Originaldokumentation von MySQL das AS nicht in der BNF aufnimmt, sondern lediglich im Zusatztext erläutert.

```
SELECT [DISTINCT] select_expr [AS alias_name] [,select_expr [AS alias_name] ...]
[FROM table_name]
[WHERE where_condition]
```

Listing 16.1 BNF von SELECT ergänzt um select_expr und AS

Falls Sie sich fragen, wo der Stern »`*` « geblieben ist: Keine Sorge, er ist nicht verloren gegangen. Der Stern ist lediglich eine mögliche Ausprägung eines SELECT-Ausdrucks.

Zusammenfassung

- Mit SELECT bestimmen Sie die Spalten der Ergebnistabelle.
- Ein Quelltabelle ist nicht zwingend erforderlich.
- Mit AS können Sie die Spaltennamen der Ergebnistabelle ändern.
- MySQL kennt Operatoren und Funktionen. Sie können sie in SELECT-Ausdrücken und WHERE- Bedingungen verwenden.
- Bei der Auswertung von Ausdrücken verwendet MySQL eine Präzedenzreihenfolge, die Sie mit Hilfe von Klammern beeinflussen können.

Operatorenübersicht in Rangreihenfolge

Präzedenz	Operator	Bedeutung	Beispiel	Ergebnis
1	*	Multiplikation	8 * 3	24
1	/	Division	8 / 3	2.6667
1	DIV	Integerdivision	8 DIV 3	2
1	%	Modulo	8 % 3	2
2	+	Addition	8 + 3	11
2	-	Subtraktion	8 – 3	5
3	=	gleich	8 = 3	0
3	!=	ungleich	8 != 3	1
3	>=	größer gleich	8 >= 3	1
3	<=	kleicher gleich	8 <= 3	0

3	>	größer	8 > 3	1
3	<	kleiner	8 < 3	0
3	LIKE	like	“Anna” LIKE “A%”	1
4	NOT	not	!TRUE	0
5	AND	and	1 && 0	1
6	OR	or	1 || 0	0

16.8 Aufgaben zur Selbstkontrolle

Übungen

Aufgabe 1: Arbeiten mit Dokumentation

Lösen Sie die folgenden Aufgaben mit Hilfe der Dokumentation von MySQL (www.mysql.com).

1. Geben Sie mit SQL die aktuelle Uhrzeit aus.
2. Geben Sie nur den aktuellen Monat aus.
3. Entwickeln Sie eine SQL-Anweisung, die einen Würfel simuliert. Bei jedem Aufruf soll eine zufällige Zahl zwischen 1 und 6 zurückgegeben werden.

Aufgabe 2: Ausgaben im Filmverleih: Regisseure

Geben Sie alle Regisseure mit ungefährem Alter aus. Sie müssen nur das Geburtsjahr berücksichtigen.

Beispiel

```
+---------+-------------+
| name    | lebensalter |
+---------+-------------+
| Natali  | 40          |
| Jackson | 48          |
+---------+-------------+
2 rows in set (0.00 sec)
```

Aufgabe 3: Ausgaben im Filmverleih: Filme

Geben Sie alle Filme mit Beschreibung aus, aber kürzen Sie die Beschreibung auf 40 Zeichen und fügen Sie '…' an.

Beispiel

```
+----------------+-------------------------+
| titel          | beschreibung            |
+----------------+-------------------------+
| Cube           | 7 völlig Fremde mit… |
| Herr der Ringe | In einem kleinen Dor… |
+----------------+-------------------------+
2 rows in set (0.00 sec)
```

Zusatzübungen

Aufgabe 4: Ausgaben im Filmverleih

Geben Sie alle Filme mit ihrer Dauer in Stunden und Minuten aus (90 Minuten entsprechen 1 Stunde und 30 Minuten).

17 Aggregatsfunktionen

In dieser Lektion lernen Sie:

- wie Sie die Anzahl der Zeilen einer Tabelle ermitteln.
- wie Sie Minimum und Maximum einer Spalte bestimmen.
- wie Sie den Durchschnitt und die Summe aller Werte einer Spalte berechnen.

Wir haben mittlerweile sehr viele Benutzer in unserer Datenbank. Gibt es eine Möglichkeit, auf einen Blick zu sehen, wie viele genau?

Außerdem hat unsere Marketingabteilung ein paar Daten angefragt: Wie viel kostet das teuerste Seminar? Wie viel das günstigste? Wie viel kostet ein Seminar im Durchschnitt?

Programmieren Sie bitte eine Seite, auf der wir diese Daten immer im aktuellen Zustand einsehen können.

Um Ihre Kundin zufrieden zu stellen, müssen Sie der Datenbank einige Informationen abverlangen. Normalerweise sollten sich diese mit Hilfe von Funktionen zusammenstellen lassen. Mit einfachen Funktionen ist das aber nicht möglich, da diese zeilenweise arbeiten. Was Sie benötigen, sind Funktionen, deren Eingabe nicht aus einer Zeile, sondern aus einer ganzen Tabelle besteht.

Tatsächlich bietet SQL solche Funktionen an. Sie heißen ***Aggregationsfunktionen***, da Sie die Informationen aus mehreren Datensätzen aggregieren (zusammenstellen).

17.1 Zählen von Zeilen

Die einfachste Aggregationsfunktion ist `COUNT`. Mit COUNT können Sie Datensätze zählen und somit z.B. erfahren, wie viele Benutzer bereits registriert sind.

Beispiel

```
SELECT COUNT(*) FROM benutzer;
```

```
+----------+
| COUNT(*) |
+----------+
| 6        |
+----------+
1 row in set (0.00 sec)
```

Beachten Sie, dass es sich beim Ergebnis wieder um eine komplette Tabelle handelt, auch wenn diese selbstverständlich nur einen einzigen Wert enthält – d.h. eine Tabelle mit einer Zeile und einer Spalte. Das ist typisch für Aggregationsfunktionen, da sie eben nicht pro Datensatz arbeiten, sondern einen Gesamtwert ermitteln.

Genauso einfach können Sie die anderen Anforderungen umsetzen.

17.2 Minimum und Maximum

Das Minimum und Maximum eines Attributes können Sie mit Hilfe der Funktionen `MIN` und `MAX` in Erfahrung bringen. Wenn Sie also wissen möchten, wie viel das teuerste Seminar kostet, verwenden Sie die folgende SQL-Anweisung:

Beispiel

```
SELECT MAX(preis) FROM seminare;
```

```
+------------+
| MAX(preis) |
+------------+
| 2500.00    |
+------------+
1 row in set (0.01 sec)
```

17.3 Durchschnitt und Summe

Mit der Funktion `AVG` (average, dt. Durchschnitt) bestimmen Sie den Durchschnittswert aller Werte einer Spalte. Dazu müssen Sie die Spalte ebenso angeben, wie bei MIN oder MAX.

Beispiel

```
SELECT AVG(preis) FROM seminare;
```

```
+-------------+
| AVG(preis)  |
+-------------+
| 1739.284286 |
+-------------+
1 row in set (0.01 sec)
```

Schließlich gibt es noch die Funktion `SUM`, mit der Sie die Summe einer Spalte berechnen können. Für die Seminarpreise macht das wenig Sinn. Das folgende Beispiel soll deswegen einfach nur zur Verdeutlichung des Prinzips dienen.

Beispiel

```
SELECT SUM(preis) FROM seminare;
```

```
+------------+
| SUM(preis) |
+------------+
| 12174.99   |
+------------+
1 row in set (0.01 sec)
```

Zusammenfassung

- Mit Aggregationsfunktionen können Sie Informationen über ganze Spalten hinweg zusammensetzen (aggregieren).

Übersicht über die wichtigsten Aggregationsfunktionen

Name	Bedeutung
COUNT	Anzahl
MIN	Minimum
MAX	Maximum
SUM	Summe
AVG	Durchschnitt

17.4 Aufgaben zur Selbstkontrolle

Testen Sie Ihr Wissen

1. Wo innerhalb einer SQL-Anweisung können Sie Aggregationsfunktionen einsetzen, wo normale Funktionen?
2. Was ist die Besonderheit von Aggregationsfunktionen bezüglich ihrer Ein- und Ausgabe?

Übungen

Aufgabe 1: Aggregationsfunktionen im Filmverleih

Entwickeln Sie SQL-Anweisungen zum Beantworten folgender Fragen:

1. Wie viele Filme kennt die Datenbank?
2. Wie viele Minuten dauert der kürzeste Film seit dem Jahr 2002?
3. Aus wie vielen Zeichen besteht die längste Filmbeschreibung?
4. Wie alt ist der älteste Film?
5. Wie lange dauert ein Science-Fiction-Film im Durchschnitt?
6. Wie viele verschiedene Genres unterscheidet die Datenbank?

18 Datensätze ändern und löschen

In dieser Lektion lernen Sie:

- wie Sie bestehende Datensätze ändern können.
- wie Sie unliebsame Datensätze löschen.

Beim Anlegen der Seminare sind uns ein paar kleine Fehler unterlaufen. Außerdem haben wir noch Änderungswünsche. Das Seminar »JavaScript« z. B., soll nun nur noch 1999,99 € kosten.

Deswegen möchten wir möglichst bald vorhandene Seminare wieder ändern oder löschen können.

18.1 Ändern von Datensätzen

Zum Ändern bestehender Datensätze stellt Ihnen SQL den Befehl `UPDATE` (aktualisieren) zur Verfügung. Probieren Sie folgende SQL-Anweisung aus:

```
UPDATE seminare SET preis = 1999.99;
```

```
Query OK, 7 rows affected (0.00 sec)
Rows matched: 7 Changed: 7 Warnings: 0
```

Es sind sieben Zeilen betroffen? Das ist verdächtig! Schauen Sie sich geänderten Seminare an:

```
SELECT titel, preis FROM seminare;
```

```
+------------------------------+---------+
| titel                        | preis   |
+------------------------------+---------+
| Datenbanken & SQL            | 1999.99 |
| Ruby on Rails                | 1999.99 |
| Ajax & DOM                   | 1999.99 |
| JavaScript                   | 1999.99 |
| Adobe Flash (Grundlagen)     | 1999.99 |
| Adobe Flash (ActionScript)   | 1999.99 |
| Bildbearbeitung mit Adobe PS | 1999.99 |
+------------------------------+---------+
7 rows in set (0.00 sec)
```

Oh weh. Was ist denn hier passiert? Alle Seminare haben den gleichen Preis!

Das Problem ist, dass `UPDATE` alle Datensätze ändert. Wenn Sie nur einige Datensätze oder sogar – wie hier – nur einen ändern möchten, müssen Sie die Zeilen beschränken. Mit welchem SQL-Schlüsselwort war das nochmal möglich?

Richtig, mit `WHERE`. Restaurieren Sie nochmal die Beispieldaten mit Hilfe der Skripte:

```
mysql> DROP DATABASE seminarverwaltung;

mysql> CREATE DATABASE seminarverwaltung;

mysql -u root seminar_verwaltung < 001_erzeuge_benutzer_und_seminare.sql

mysql -u root seminar_verwaltung < 002_ergaenze_anrede_und_kategorie.sql

mysql -u root seminar_verwaltung < benutzer.sql

mysql -u root seminar_verwaltung < seminare.sql
```

Ergänzen Sie das UPDATE nun mit einem WHERE. Sie müssen sich auf das richtige Seminar beziehen! Am einfachsten lässt sich das mit Hilfe der `id` realisieren.

```
UPDATE seminare SET preis = 1999.99 WHERE id = 4;
```

```
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0
```

Überprüfen Sie wieder die Änderung:

```
SELECT titel, preis FROM seminare;
```

```
+------------------------------+---------+
| titel                        | preis   |
+------------------------------+---------+
| Datenbanken & SQL            | 975.00  |
| Ruby on Rails                | 2500.00 |
```

```
| Ajax & DOM                  | 1699.99 |
| JavaScript                  | 1999.99 |
| Adobe Flash (Grundlagen)    | 1500.00 |
| Adobe Flash (ActionScript)  | 1500.00 |
| Bildbearbeitung mit Adobe PS | 1500.00 |
+-----------------------------+---------+
7 rows in set (0.00 sec)
```

Diesmal haben Sie nur das richtige Seminar geändert.

Die BNF von UPDATE ist einfach:

```
UPDATE table_name SET col_name1 = value1 [, col_name2 = value2] [WHERE
where_condition]
```

18.2 Ausdrücke im UPDATE

Statt eines Wertes können Sie beim UPDATE auch ganze Ausdrücke einsetzen. Beispielsweise können Sie mit dem folgendem SQL eine Preiserhöhung aller Seminare um 10% veranlassen:

```
UPDATE seminare SET preis = preis * 1.1;
```

18.3 Löschen von Datensätzen

Der Befehl `DELETE` dient zum Löschen von Datensätzen und funktioniert ganz ähnlich wie der UPDATE-Befehl. Auch beim Löschen sollten Sie mittels `WHERE` angeben, welche Datensätze bzw. welchen Datensatz Sie löschen möchten. Geben Sie nichts weiter an, so löscht MySQL alle Datensätze der Tabelle.

Probieren Sie es dennoch – Sie können die Datensätze gleich wiederherstellen.

```
DELETE FROM seminare;
```

```
Query OK, 7 rows affected (0.00 sec)
```

Ein `SELECT id, titel FROM seminare;` zeigt Ihnen nun eine leere Tabelle:

```
Empty set (0.00 sec)
```

Stellen Sie die Datensätze mit Hilfe des Skriptes *seminare.sql* wieder her.

Versuchen Sie es nun erneut mit

```
DELETE FROM seminare WHERE id = 1;
```

```
Query OK, 1 row affected (0.00 sec)
```

Diesmal haben Sie nur das Seminar *Datenbanken & SQL* gelöscht, wie Sie mittels `SELECT id, titel FROM seminare;` wieder leicht überprüfen können:

```
+----+------------------------------+
| id | titel                        |
+----+------------------------------+
| 2  | Ruby on Rails                |
| 3  | Ajax & DOM                   |
| 4  | JavaScript                   |
| 5  | Adobe Flash (Grundlagen)     |
| 6  | Adobe Flash (ActionScript)   |
| 7  | Bildbearbeitung mit Adobe PS |
+----+------------------------------+
6 rows in set (0.00 sec)
```

Hier noch die BNF von `DELETE`:

```
DELETE FROM tabel_name [WHERE where_condition]
```

Zusammenfassung

- Mit UPDATE können Sie bestehende Datensätze ändern (aktualisieren).
- Mit DELETE können Sie Datensätze löschen.

SQL: DML

Zweck	BNF

Datensätze auslesen	`SELECT [DISTINCT] select_expr [AS alias_name] [,select_expr [AS alias_name] ...]`
	`[FROM table_name]`
	`[WHERE where_condition]`

Datensatz	

einfügen	`INSERT INTO tbl_name (col_name,...) VALUES (value,...)`
Datensätze ändern	`UPDATE table_name SET col_name1 = value1 [, col_name2 = value2]` `[WHERE where_condition]`
Datensätze löschen	`DELETE FROM table_name [WHERE where_condition]`

Tabelle 18.1 DML-Anweisungen

18.4 Aufgaben zur Selbstkontrolle

Testen Sie Ihr Wissen

1. Was passiert, wenn Sie ein UPDATE- oder DELETE-Anweisung ohne WHERE verwenden?

Übungen

Aufgabe 1: Fehlerkorrektur

Die Spieldauer des Films *The Dark Knight* ist mit 180 Minuten falsch angegeben. Ändern Sie sie auf 152 Minuten.

Aufgabe 2: Löschen alter Filme

Alle Filme, die von 1998 und älter sind, werden aus dem Programm genommen. Löschen Sie sie!

Zusatzübungen

Aufgabe 3: Namensmigration

Entwickeln Sie ein Migrationsskript für die Tabelle *regisseure*. Die Spalte *vorname* entfällt. Stattdessen soll der jeweilige *vorname* mit bei *name* gespeichert werden. Achten Sie darauf, dass die Migration verlustfrei geschieht.

Beispiel

Name: Jackson, **Vorname:** Peter => **Name:** Peter Jackson

19 Sortierung und LIMIT

In dieser Lektion lernen Sie:

- wie Sie Datensätze mit SQL sortieren.
- wie Sie mit Hilfe von SQL eine Blättern-Funktion realisieren.

Das Mitarbeiterinterface zur Seminarverwaltung ist wirklich recht praktisch – vielen Dank für die hervorragende Arbeit. Leider geht uns bei den vielen Seminaren immer noch hin und wieder die Übersicht verloren. Wir haben uns beraten und sind auf folgende Idee gekommen:

Die Überschriften Titel, Preis und Kategorie sollten anklickbar sein. Durch das Klicken müssen die Seminare dann entsprechend sortiert werden – und am besten jeweils wahlweise auf- oder absteigend.

19.1 Sortierung

Einmal mehr überrascht Sie Ihre Kundin mit neuen Ideen. Zum Glück ist das kein Problem für Sie – denn wie immer hat SQL die richtige Lösung parat. Sortieren können Sie in SQL mit der `ORDER BY` -Anweisung, die Sie zusätzlich noch hinter dem WHERE notieren. Sie können ORDER BY aber natürlich auch ohne WHERE verwenden. Es genügt dabei, das Attribut anzugeben, nach dem Sie sortieren möchten.

Beispiele

```
SELECT titel, preis, kategorie FROM seminare ORDER BY titel;
```

```
+------------------------------+---------+----------------+
| titel                        | preis   | kategorie      |
+------------------------------+---------+----------------+
| Adobe Flash (ActionScript)   | 1500.00 | Programmierung |
| Adobe Flash (Grundlagen)     | 1500.00 | Webdesign      |
| Ajax & DOM                   | 1699.99 | Programmierung |
| Bildbearbeitung mit Adobe PS | 1500.00 | Webdesign      |
| JavaScript                   | 2500.00 | Programmierung |
| Datenbanken & SQL            | 975.00  | Datenbanken    |
| Ruby on Rails                | 2500.00 | Programmierung |
+------------------------------+---------+----------------+
7 rows in set (0.01 sec)
```

```
SELECT titel, preis, kategorie FROM seminare ORDER BY preis;
```

```
+------------------------------+---------+----------------+
| titel                        | preis   | kategorie      |
+------------------------------+---------+----------------+
| Datenbanken & SQL            | 975.00  | Datenbanken    |
| Adobe Flash (Grundlagen)     | 1500.00 | Webdesign      |
| Adobe Flash (ActionScript)   | 1500.00 | Programmierung |
| Bildbearbeitung mit Adobe PS | 1500.00 | Webdesign      |
| Ajax & DOM                   | 1699.99 | Programmierung |
| Ruby on Rails                | 2500.00 | Programmierung |
| JavaScript                   | 2500.00 | Programmierung |
+------------------------------+---------+----------------+
7 rows in set (0.00 sec)
```

19.2 Sortierung nach mehreren Attributen

```
SELECT titel, preis, kategorie FROM seminare ORDER BY kategorie;
```

```
+-----------------------------+---------+----------------+
| titel                       | preis   | kategorie      |
+-----------------------------+---------+----------------+
| Datenbanken & SQL           | 975.00  | Datenbanken    |
| Ruby on Rails               | 2500.00 | Programmierung |
| Ajax & DOM                  | 1699.99 | Programmierung |
| JavaScript                  | 2500.00 | Programmierung |
| Adobe Flash (ActionScript)  | 1500.00 | Programmierung |
| Adobe Flash (Grundlagen)    | 1500.00 | Webdesign      |
| Bildbearbeitung mit Adobe PS | 1500.00 | Webdesign      |
+-----------------------------+---------+----------------+
7 rows in set (0.00 sec)
```

Es ist auch möglich, mehrere Attribute zur Sortierung heranzuziehen. Beispielsweise können Sie erst nach Kategorie sortieren und dann noch mal innerhalb der Kategorie alphabetisch nach dem Titel. Geben Sie dazu einfach beide Attribute an – durch Komma getrennt. Die Reihenfolge ist dabei entscheidend. Das erste Attribut wirkt auch als wichtigstes Sortierkriterium.

Beispiel

```
SELECT titel, preis, kategorie FROM seminare ORDER BY kategorie, titel;
```

```
+-----------------------------+---------+----------------+
| titel                       | preis   | kategorie      |
+-----------------------------+---------+----------------+
| Datenbanken & SQL           | 975.00  | Datenbanken    |
| Adobe Flash (ActionScript)  | 1500.00 | Programmierung |
| Ajax & DOM                  | 1699.99 | Programmierung |
| JavaScript                  | 2500.00 | Programmierung |
| Ruby on Rails               | 2500.00 | Programmierung |
| Adobe Flash (Grundlagen)    | 1500.00 | Webdesign      |
| Bildbearbeitung mit Adobe PS | 1500.00 | Webdesign      |
+-----------------------------+---------+----------------+
7 rows in set (0.00 sec)
```

```
SELECT titel, preis, kategorie FROM seminare ORDER BY kategorie, preis;
```

```
+-----------------------------+---------+----------------+
| titel                       | preis   | kategorie      |
+-----------------------------+---------+----------------+
| Datenbanken & SQL           | 975.00  | Datenbanken    |
| Adobe Flash (ActionScript)  | 1500.00 | Programmierung |
| Ajax & DOM                  | 1699.99 | Programmierung |
| Ruby on Rails               | 2500.00 | Programmierung |
| JavaScript                  | 2500.00 | Programmierung |
| Adobe Flash (Grundlagen)    | 1500.00 | Webdesign      |
| Bildbearbeitung mit Adobe PS | 1500.00 | Webdesign      |
+-----------------------------+---------+----------------+
7 rows in set (0.00 sec)
```

```
SELECT titel, preis, kategorie FROM seminare ORDER BY preis, kategorie;
```

```
+-----------------------------+---------+----------------+
| titel                       | preis   | kategorie      |
+-----------------------------+---------+----------------+
| Datenbanken & SQL           | 975.00  | Datenbanken    |
| Adobe Flash (ActionScript)  | 1500.00 | Programmierung |
| Adobe Flash (Grundlagen)    | 1500.00 | Webdesign      |
| Bildbearbeitung mit Adobe PS | 1500.00 | Webdesign      |
| Ajax & DOM                  | 1699.99 | Programmierung |
| Ruby on Rails               | 2500.00 | Programmierung |
| JavaScript                  | 2500.00 | Programmierung |
+-----------------------------+---------+----------------+
7 rows in set (0.00 sec)
```

19.3 Auf- und absteigende Sortierung

Ihre Auftraggeberin hat sich insbesondere noch gewünscht, dass zwischen auf- und absteigender Sortierung gewechselt werden kann. Auch das ist in SQL vorgesehen – in Form der Schlüsselwörter `ASC` und `DESC`.

ASC steht für *ascending*, d.h. aufsteigend. Eine aufsteigende Sortierung bedeutet, dass die Ausgabe mit dem kleinsten Wert beginnt und dann immer größere listet. Im Falle von Zeichentypen gibt es eine vordefinierte Reihenfolge, bei der A kleiner als Z gilt. Dementsprechend ist mit *aufsteigend* hier eine normale alphabetische Sortierung gemeint. Sie könnten also beispielsweise schreiben:

```
SELECT titel, preis, kategorie FROM seminare ORDER BY title ASC;
```

Das ASC müssen Sie aber nicht angeben, da es sich sowieso schon um das Standardverhalten von ORDER BY handelt.

Wollen Sie dagegen absteigend sortieren, ist die Angabe von DESC (descending – absteigend) verpflichtend.

Beispiel

```
SELECT titel, preis, kategorie FROM seminare ORDER BY titel DESC;
```

```
+------------------------------+---------+----------------+
| titel                        | preis   | kategorie      |
+------------------------------+---------+----------------+
| Ruby on Rails                | 2500.00 | Programmierung |
| Datenbanken & SQL            | 975.00  | Datenbanken    |
| JavaScript                   | 2500.00 | Programmierung |
| Bildbearbeitung mit Adobe PS | 1500.00 | Webdesign      |
| Ajax & DOM                   | 1699.99 | Programmierung |
| Adobe Flash (Grundlagen)     | 1500.00 | Webdesign      |
| Adobe Flash (ActionScript)   | 1500.00 | Programmierung |
+------------------------------+---------+----------------+
7 rows in set (0.00 sec)
```

Wie Sie sehen, steht nun das Seminar *Ruby on Rails* (das letzte nach der alphabetischen Reihenfolge) an erster Stelle, während *Adobe Flash (ActionScript)* die letzte Stelle belegt.

19.4 Die BNF von ORDER BY

Die BNF von ORDER BY ist nicht weiter kompliziert. ORDER BY erweitert wieder einmal die BNF von SELECT.

```
SELECT [DISTINCT] * | select_expr [AS alias_name][,select_expr [AS alias_name] ...]
FROM table_name
[WHERE where_condition]
[ORDER BY col_name [ASC | DESC], ...]
```

Listing 19.1 BNF von SELECT ergänzt um ORDER BY

19.5 Limit

Eine weitere Verbesserung, die zur Übersichtlichkeit beitragen könnte, wäre, wenn die Katalogseite nicht immer alle Seminare zeigt. Ich denke, die Seite wird bedienbarer und die Ladezeiten sicherlich ein wenig kürzer, wenn die Seite nur die ersten 20 Seminare anzeigt. Um zu den nächsten 20 zu kommen, wäre eine Blätterfunktion angebracht.

Tatsächlich bietet MySQL eine solche Funktion bereits out-of-the-box. Die Rede ist vom Schlüsselwort `LIMIT`. Mit Hilfe von LIMIT können Sie die Suchergebnisse auf eine fixe Anzahl beschränken. Im einfachsten Fall geben Sie die Anzahl von Zeilen an, die die Ergebnistabelle zeigen soll.

Beispiel

Das folgende Beispiel zeigt die ersten drei Seminare. Natürlich ließen sich statt 3 auch 20 Seminare anzeigen. Um den Effekt zu sehen, bräuchten Sie aber viel mehr Testdatensätze.

```
SELECT id, titel FROM seminare LIMIT 3;
```

```
+----+------------------+
| id | titel            |
+----+------------------+
| 1  | Datenbanken & SQL |
| 2  | Ruby on Rails    |
| 3  | Ajax & DOM       |
+----+------------------+
3 rows in set (0.00 sec)
```

Wenn der Kunde die zweite Seite sehen möchte, müssen Sie die Seminare 4 bis 6 ausgeben. Geben Sie LIMIT dazu einfach zwei Parameter. Der erste Parameter ist in diesem Fall die Verschiebung – der sogenannte ***Offset***. Der zweite Parameter gibt erneut die Anzahl an.

Beispiel

```
SELECT id, titel FROM seminare LIMIT 3, 3;
```

```
+----+--------------------------+
| id | titel                    |
+----+--------------------------+
| 4  | JavaScript               |
| 5  | Adobe Flash (Grundlagen)   |
| 6  | Adobe Flash (ActionScript) |
+----+--------------------------+
3 rows in set (0.00 sec)
```

Genauso geben Sie dann ein dritte Seite aus. Sie beginnen bei Seminar 7 (Verschiebung um 6) und geben wiederum 3 Seminare aus.

```
SELECT id, titel FROM seminare LIMIT 6, 3;
```

```
+----+------------------------------+
| id | titel                        |
+----+------------------------------+
| 7  | Bildbearbeitung mit Adobe PS |
+----+------------------------------+
1 row in set (0.00 sec)
```

Da es in den Testdaten nur insgesamt sieben Seminare gibt, wird nur noch das letzte ausgegeben.

19.6 BNF von LIMIT

LIMIT ist die letzte Angabe in der SELECT-Anweisung. Damit ist die BNF von SELECT – für die Belange dieses Buches – fast komplett. Eine Erweiterung lernen Sie aber später noch kennen.

```
SELECT [DISTINCT] select_expr [AS alias_name][,select_expr [AS alias_name] ...]
[FROM table_name]
[WHERE where_condition]
[ORDER BY col_name [ASC | DESC], ...]
[LIMIT [offset,] row_count]
```

Listing 19.2 BNF von SELECT ergänzt um LIMIT

Zusammenfassung

- Mit ORDER BY können Sie die Ergebnistabelle sortieren.
- Mit LIMIT können Sie die Ausgabe auf eine fixe Anzahl von Zeile beschränken.
- LIMIT ist besonders nützlich, um eine Blättern-Funktion zu programmieren.

19.7 Aufgaben zur Selbstkontrolle

Testen Sie Ihr Wissen

1. Nennen Sie zwei Varianten, um die Tabelle *seminar* nach dem Attribut *preis* aufsteigend zu sortieren.

Übungen

Aufgabe 1: Filme sortiert ausgeben

1. Geben Sie alle Filme nach Titel sortiert aus.
2. Geben Sie alle Filme nach Spieldauer sortiert aus. Dabei soll der längste Film zuerst erscheinen und der kürzeste zuletzt.
3. Geben Sie alle Filme in der Reihenfolge ihres Erscheinens aus. Falls in einem Jahr mehrere Filme vorhanden sind, so müssen diese innerhalb des Jahres noch alphabetisch sortiert werden.

Aufgabe 2: Filme ausgeben mit Limit

1. Geben Sie zwei Filme aus, die nach dem Jahr 2000 erschienen sind.
2. Geben Sie die nächsten zwei Filme aus, die nach dem Jahr 2000 erschienen sind.

20 Die Null-Markierung

In dieser Lektion lernen Sie:

- wie Sie mit unbekannten Daten umgehen.
- wie Sie Pflichtfelder anlegen.
- unbekannte Daten wiederzufinden.

20.1 NULL-Marker

Wir haben einige Seminare einzutragen, für die noch kein Preis feststeht. Kann ich das Feld leer lassen, oder soll ich 0 rein schreiben?

Ein Seminarpreis von *0* würde bedeuten, dass das Seminar nichts kostet. Ein solches Seminar würde aber später auf der Seite als buchbar gelistet und ein Kunde könnte es kostenlos belegen – das ist sicherlich nicht im Sinne der Akademie.

Deswegen müssen Sie in der Datenbank gezielt vermerken, dass der Preis noch nicht feststeht, d. h. unbekannt ist. Für diesen Fall hat SQL den Wert `NULL` vorgesehen. `NULL` ist nicht wirklich ein Wert, sondern eher eine Art Markierung. Sie können in ein Tabellenfeld statt eines Wertes NULL eintragen. Das hat die Bedeutung, dass das Feld zwar einen Wert besitzt, dieser jedoch unbekannt ist. Wenn also z. B. der Preis für das Seminar *Web Usability* noch nicht feststeht, lässt sich das folgendermaßen darstellen:

id	titel	beschreibung	preis
1	Ruby on Rails	Ruby on Rails ist das neue, sensationelle OpenSource-Framework, das auf der modernen…	2500,00
8	Web Usability	Webanwendungen wie etwa Online-Shops zeichnen sich durch eine Vielzahl …	**NULL**

Tabelle 20.1 Seminare mit NULLs

NULL ist dabei nicht etwa der String ‘NULL’, sondern ein SQL-Schlüsselwort, das eben für einen unbekannten Wert steht. Deswegen können Sie NULL bei jedem beliebigen Datentyp verwenden.

Fügen Sie das neue Seminar mit folgender Anweisung ein:

```
INSERT INTO seminare (titel, beschreibung, preis, kategorie) VALUES ('Web Usability', 'Webanwendungen wie etwa Online-Shops zeichnen sich durch eine Vielzahl…', NULL , 'Webdesign');
```

Die Tabellenzelle markiert den Preis für das neue Seminar mit `NULL`:

```
SELECT id, titel, preis FROM seminare;
```

```
+----+-----------------------------+---------+
| id | titel                       | preis   |
+----+-----------------------------+---------+
| 2 | Ruby on Rails                | 2500.00 |
| 3 | Ajax & DOM                   | 1699.99 |
| 4 | JavaScript                   | 2500.00 |
| 5 | Adobe Flash (Grundlagen)     | 1500.00 |
| 6 | Adobe Flash (ActionScript)   | 1500.00 |
| 7 | Bildbearbeitung mit Adobe PS | 1500.00 |
| 8 | Web Usability                | NULL    |
+----+-----------------------------+---------+
```

20.2 NULL-Operatoren

Dass NULL kein Wert ist, sehen Sie daran, wie SQL NULL in Ausdrücken behandelt. Versuchen Sie alle Seminare auszugeben, deren Preis NULL ist.

Ein erster Versuch könnte lauten:

```
SELECT id, titel, preis FROM seminare WHERE preis = NULL;
```

```
Empty set (0.00 sec)
```

Sie erhalten eine leere Ergebnismenge, da NULL kein Wert ist. Es gibt nichts, was gleich NULL sein kann. Das Ergebnis von `NULL = NULL` ist nicht `TRUE`, sondern `NULL`. Sie können es leicht ausprobieren mit

```
SELECT NULL = NULL;
```

```
+-----------+
| NULL=NULL |
+-----------+
| NULL      |
+-----------+
1 row in set (0.00 sec)
```

Woran liegt das?

NULL ist kein Wert, sondern steht für einen unbekannten Wert. Da auf beiden Seiten der Gleichung NULL steht, bedeutet das, dass es sich um zwei unbekannte Werte handelt. Die beiden Werte könnten gleich sein, sie könnten aber auch verschieden sein – wir wissen es nicht. Deswegen lässt sich die Frage nach der Gleichheit weder mit `TRUE` (`1`) noch mit `FALSE` (`0`) beantworten. Sie erhalten wiederum `NULL`, d. h. *unbekannt* als Ergebnis.

Dennoch kann es interessant sein, diese Anfrage zu stellen:

- *Gib alle Seminare zurück, deren Preis noch nicht feststeht!*

SQL bietet für solche Fälle speziellen Operatoren an:

- `IS NULL`
- `IS NOT NULL`

Mittels

```
SELECT id, titel, preis FROM seminare WHERE preis IS NULL ;
```

erhalten Sie alle Seminare, deren Preis unbekannt ist bzw. noch nicht feststeht.

```
+----+---------------+-------+
| id | titel         | preis |
+----+---------------+-------+
| 8  | Web Usability | NULL  |
+----+---------------+-------+
1 row in set (0.00 sec)
```

Die Anweisung

```
SELECT id, titel, preis FROM seminare WHERE preis IS NOT NULL ;
```

zeigt dagegen alle, für die ein Preis angegeben ist:

```
+----+----------------------------+---------+
```

```
| id | titel                         | preis   |
+----+-------------------------------+---------+
| 2  | Ruby on Rails                 | 2500.00 |
| 3  | Ajax & DOM                    | 1699.99 |
| 4  | JavaScript                    | 2500.00 |
| 5  | Adobe Flash (Grundlagen)      | 1500.00 |
| 6  | Adobe Flash (ActionScript)    | 1500.00 |
| 7  | Bildbearbeitung mit Adobe PS | 1500.00 |
+----+-------------------------------+---------+
6 rows in set (0.00 sec)
```

20.3 NULL-Marker für »nicht existent«

Neben der Bedeutung *unbekannt* setzen viele Entwickler NULL-Marker auch mit der Bedeutung *nicht existent* ein. Sollte beispielsweise ein Benutzer (d. h. Kunde) tatsächlich keine E-Mail-Adresse besitzen, so könnten Sie auch NULL in das entsprechende Feld eintragen.

Nun kann die Anwendung aber nicht mehr unterscheiden, zwischen Benutzern, deren E-Mail unbekannt ist, und solchen, die gar keine haben. Deswegen empfiehlt es sich, wenn möglich, stattdessen einen speziellen Wert zu verwenden. Sie könnten z. B.

- den String *n/a* (*not available*, engl. für *nicht verfügbar*),
- den String *keine E-Mail vorhanden*
- oder einfach einen leeren String

verwenden. Das ist aber auch nicht immer möglich – und schon gar nicht mit jedem Datentyp[1] . Außerdem birgt diese Technik ihre eigenen Gefahren, wie folgendes Beispiel zeigt.

---

1 Bei INTEGER z. B. verwenden Datenbankentwickler oft -1. Das ist natürlich nur möglich, falls die regulären Werte alle positiv sind.

Der NONE-Fahrer

Nicht bei allen Fahrzeugen, denen ein Strafzettel zugeordnet werden soll, kann das Kennzeichen festgestellt werden. Deswegen speicherte die Datenbank einer amerikanischen Polizeistation stattdessen in solchen Fällen den String »NONE« (keins) als Kennzeichen ab.

Als dann später ein Autofahrer tatsächlich ein Kennzeichen mit der Buchstabenkombination »NONE« anmeldete, sendete ihm die Polizeistation mehrere tausend Strafzettel zu[1].

---

1 gefunden bei Celko (2005)

In der Praxis ist es deswegen auch nicht unüblich, NULL in der Bedeutung *nicht existent* zu verwenden. Sie müssen es von Fall zu Fall abwägen. Außerdem kann es je nach Situation neben *unbekannt* und *nicht existent* weitere Zustände geben. Die Lösungen sind dementsprechend auch situationsabhängig. Sie sehen wieder einmal: Datenbankdesign ist ein kreativer Prozess!

20.4 NULL im relationalen Modell

Codd führte die NULL-Marker in der ersten Version des relationalen Modells ein. 1990 hat er eine neue Version des Modells vorgestellt, die zwei verschiedene Arten von NULL-Markern unterscheidet. Diese Version hat sich aber in der Praxis nie durchgesetzt, da sie logische Berechnungen und das Behandeln von NULL-Markern weiter verkompliziert[1].

---

1 de.wikipedia.org/wiki/Nullwert#NULL_in_SQL-Datenbanken

Aus der neusten Version des relationalen Modells haben Date und Darwen (2006) die NULL-Marker sogar gänzlich entfernt. Zuvor haben sie gezeigt, dass NULL-Marker auch zu erheblichen Problemen führen können.

Christopher sagt…

NULL-marker haben keinen Platz im relationalen Modell.

Leider hat bisher noch kein Datenbankhersteller ein praxistaugliches DBMS entwickelt, das die neuste Version des Modells vollständig umsetzt. In der Praxis sind Sie deswegen in vielen Situationen auf NULL-Marker angewiesen.

20.5 Not-Null-Spalten

Einer unserer Mitarbeiter hat versehentlich ein Seminar ohne Titel eingetragen. Das macht natürlich keinen Sinn. Können Sie so etwas wie Pflichtfelder in der Datenbank hinterlegen, damit niemand mehr unvollständige Datensätze eingibt?

Natürlich können Sie! Um eine Art Pflichtfeld zu erhalten, teilen Sie der Datenbank mit, dass die entsprechende Spalte kein NULL annehmen darf. Markieren Sie dazu die entsprechende Spalte in einer DDL-Anweisung als `NOT NULL`.

Würde die Tabelle *seminare* noch nicht existieren, könnten Sie folgende Anweisung verwenden:

```
CREATE TABLE seminare (id INTEGER PRIMARY KEY AUTO_INCREMENT, titel
VARCHAR(120) NOT NULL, beschreibung TEXT, preis DECIMAL(6,2));
```

Da die Tabelle schon besteht und Sie keine Daten verlieren möchten, verwenden Sie stattdessen folgende ALTER-Anweisung – sinnvollerweise in einem Migrationsskript.

```
ALTER TABLE seminare CHANGE titel titel VARCHAR(120) NOT NULL;
```

Listing 20.1 003_markiere_seminare_titel_als_nn.sql

Wenn Sie nun noch mal versuchen, ein Seminar ohne Titel einzutragen, erhalten Sie eine Fehlermeldung:

```
INSERT INTO seminare (titel, preis) VALUES (NULL, 2500);
```

```
ERROR 1048 (23000): Column 'titel' cannot be null
```

Außerdem können Sie sich jederzeit anschauen, welche Spalten NULL-Marker enthalten dürfen und welche nicht:

```
SHOW COLUMNS FROM seminare;
```

```
+--------------+--------------+------+-----+-...
| Field        | Type         | Null | Key | ...
+--------------+--------------+------+-----+-...
| id           | int(11)      | NO   | PRI | ...
| titel        | varchar(120) | NO   | UNI | ...
| beschreibung | text         | YES  |     | ...
| preis        | decimal(6,2) | YES  |     | ...
| kategorie    | varchar(20)  | YES  |     | ...
+--------------+--------------+------+-----+-...
5 rows in set (0.00 sec)
```

Die Spalten, die bei NULL mit *NO* gekennzeichnet sind, sind NOT-NULL-Spalten und dürfen keine NULL-Markierungen aufnehmen. Die *id* ist übrigens auch eine NOT-NULL-Spalte, da für den Primärschlüssel NULL-Markierungen grundsätzlich nicht erlaubt sind.

20.6 NOT NULL in UML

Scott Ambler (2003) schlägt vor, `NOT NULL` im physischen Datenbankmodell mit Hilfe der ***Object Constraint Language (OCL)*** zu modellieren. Dazu müssen Sie `not null` jeweils hinter dem betreffenden Attribut in geschweiften Klammern angeben. Beachten Sie, dass `not null` in UML klein geschrieben wird.

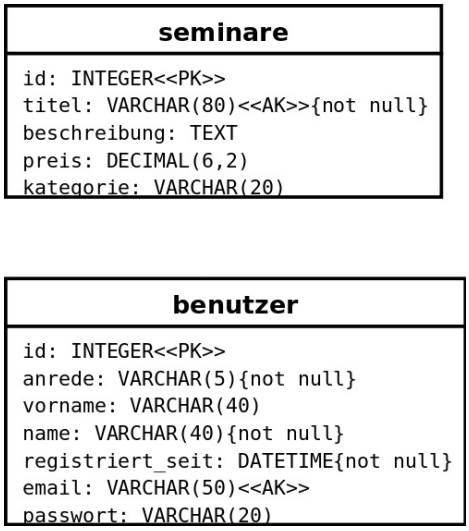


Abb. 20.1 Physisches Datenmodell mit NOT-NULL-Spalten

20.7 Vorgeschlagene Vorgehensweise

Für das Design Ihrer Datenbank empfehle ich Ihnen, nach folgender Reihenfolge vorzugehen:

- Falls es möglich und sinnvoll ist, versuchen Sie, Attribute als *not null* zu markieren. Das vereinfacht die Handhabung und bringt zudem Performancevorteile[1].
- Falls es nötig ist, setzen Sie `NULL` mit der Bedeutung *unbekannt* ein.
- Falls es nötig ist, führen Sie für *nicht existent*, *nicht anwendbar* usw. spezielle Werte ein, wie z. B. den leeren String, *n/a* usw[1] .
- Erwägen Sie NULL mit einer anderen Bedeutung als *unbekannt* einzusetzen.
- Finden Sie eine angemessene Lösung unter Abwägung der Vor- und Nachteile.

---

1 siehe Schwartz et al. (2008)
1 vergleiche 5.3.3 »Use Explicit Missing Values to Avoid NULLs« bei Joe Celko (2005).

Zusammenfassung

- NULL-Marker stehen für unbekannte Werte.
- Sie können Sie aber auch in anderer Bedeutung verwenden, z.B. als *nicht existent*.
- Spalten, die Sie als `NOT NULL` markiert haben, sind Pflichtfelder.
- Zum Abfragen von NULL-Werten benötigen Sie die Operatoren `IS NULL` und `IS NOT NULL`.

SQL: DDL

Zweck	BNF
Datenbank anlegen	`CREATE DATABASE db_name`
Datenbank löschen	`DROP DATABASE db_name`

Tabelle anlegen	`CREATE TABLE tbl_name (create_definition,...)`
	`create_definition:`
	`  col_name column_definition`
	`  | PRIMARY KEY (col_name,...)`
	`  | UNIQUE KEY (col_name,...)`
	`column_definition:`
	`  data_type [NOT NULL] [AUTO_INCREMENT] [UNIQUE KEY | PRIMARY KEY]`

Tabelle löschen	`DROP TABLE tbl_name`

Tabellenschema ändern	`ALTER TABLE tbl_name alter_specification [, alter_specification] ...`
	`alter_specification:`
	`  ADD col_name column_definition`
	`  | DROP col_name`
	`  | CHANGE old_col_name new_col_name column_definition`
	`  | RENAME TO new_tbl_name`

20.8 Aufgaben zur Selbstkontrolle

Testen Sie Ihr Wissen

1. Überlegen Sie sich mindestens zwei weitere Bedeutungen, für die Sie den NULL-Marker einsetzen könnten.
2. Welches UML-Mittel benötigen Sie, um Attribute in einem Klassendiagramm als NOT NULL zu kennzeichnen? Wie lautet die Kennzeichnung genau?
3. Welche Änderung ergibt sich bezüglich NULLs in der neusten Version des relationalen Modells?
4. Was ergibt NULL = NULL in SQL?

Übungen

Aufgabe 1: NOT NULL Spalten

Verwenden Sie für die beiden folgenden Änderungen ein Migrationsskript:

1. Sorgen Sie dafür, dass keine Filme mehr ohne Erscheinungsjahr eingetragen werden können.
2. Machen Sie die Spalten *vorname* und *nachname* der Tabelle *regisseure* zu Pflichtfeldern.

Aufgabe 2: Abfrage mit NULL-Werten

1. Geben Sie alle Filme aus, bei denen keine Spieldauer angegeben ist.
2. Geben Sie alle Filme nach Spieldauer sortiert aus. Filme, die keine Spieldauer haben, sollen in dieser Auflistung nicht erscheinen.

Zusatzübungen

Aufgabe 3: Pflichtfelder im Projekt »Fluggesellschaft«

Überlegen Sie sich sinnvolle Pflichtfelder für die Tabellen der Fluggesellschaft und schreiben Sie ein Migrationsskript, das die ensprechenden Spalten als NOT NULL markiert.

Aufgabe 4: Pflichtfelder im Projekt »Partnervermittlung«

Schreiben Sie ein Migrationsskript, um nun auch die Spalten der Partnervermittlung als NOT NULL auszuweisen, die Ihrer Meinung nach Pflichtfelder sein sollten.

21 Beziehungen im Domänenmodell

In dieser Lektion lernen Sie:

- wie Sie Beziehungen zwischen Entities herstellen.
- verschiedene Arten von Beziehungen zu unterscheiden.
- wie Sie Beziehungen in UML ausdrücken.

Jehnna: »Conan! Da sind sechs von ihnen gegen sie!«

Conan: »Eins, zwei, drei, … Ich denke Du hast recht.«

Arnold Schwarzenegger
& Grace Jones
in »Conan der Zerstörer«

Damit wir besser planen können, müssen wir für die Seminare noch Termine eintragen. Für jeden Seminartermin muss festgehalten werden, an welchem Tag das Seminar beginnt und an welchem es endet. Außerdem möchten wir wissen, in welchem Raum das Seminar zu diesem Termin stattfindet.

In der Anwendung wollen wir ein Seminar wie z. B. »Datenbanken & SQL« auswählen. Die Anwendung soll alle dazugehörigen Termine mit den entsprechenden Daten auflisten.

21.1 Beziehungen modellieren und benennen

Pflegen Sie diese neuen Anforderungen zunächst wieder im Domänenmodell ein. Das Resultat könnte folgendermaßen aussehen:

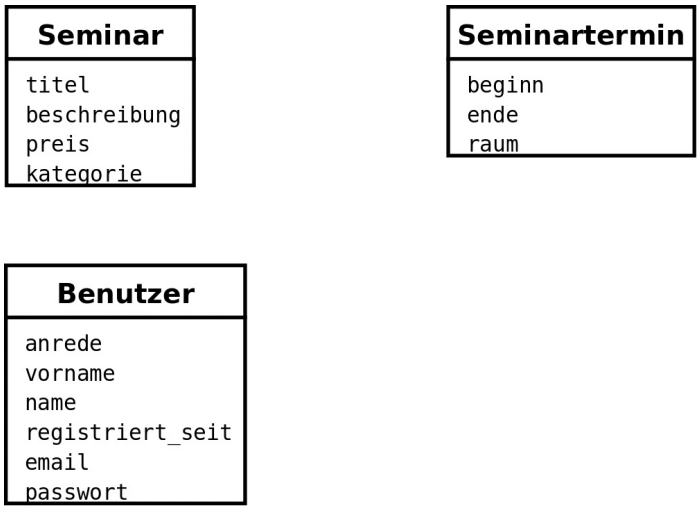


Abb. 21.1 Domänenmodell mit der neuen Entity *Seminartermin*

Damit Sie die Termine den jeweiligen Seminaren zuordnen können, benötigt das Modell noch eine Verbindung zwischen Seminaren und Seminarterminen. Dadurch können Sie auch die letzte Anforderung Ihrer Kundin umsetzen und zu jedem Seminar aller Termine auflisten. In der Datenbankentwicklung wird eine solche Verbindung zwischen zwei Entities ***Beziehung*** genannt (engl.: ***Relationship***).

In UML wird sie durch eine einfache Linie dargestellt. Sie können die Beziehung außerdem noch benennen. Für die Benennungen bieten sich Verben an, die in Verbindung mit den Entities einen Satz ergeben. Sie können also z. B. sagen:

- Ein Seminar **findet statt** zu einem Seminartermin.

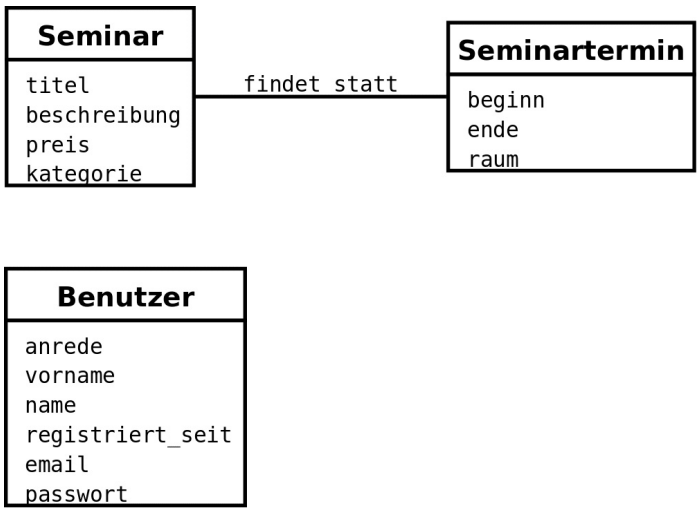


Abb. 21.2 Domänenmodell mit der Beziehung *findet statt*

Die Beziehung heißt entsprechend *findet statt.* Sie müssen übrigens nicht zusätzlich festhalten, wie das Seminar zum jeweiligen Termin heißt, welche Beschreibung es hat, was es kostet usw. Das ergibt sich jeweils aus der Beziehung.

Nachdem Sie Ihrer Kundin das Diagramm gezeigt haben, fällt ihr noch etwas ein.

Ach ja, wir möchten auch noch festhalten, welche Teilnehmer welche Seminare zu welchem Termin besuchen. Alle Teilnehmer registrieren sich entweder selbst oder werden von uns eingetragen. Deswegen sind alle Teilnehmer auch Benutzer. Es sollte doch kein Problem sein, diese zuzuordnen, oder?

Auch hier benötigen Sie wieder eine Beziehung. Diesmal zwischen Benutzer und Seminartermin.

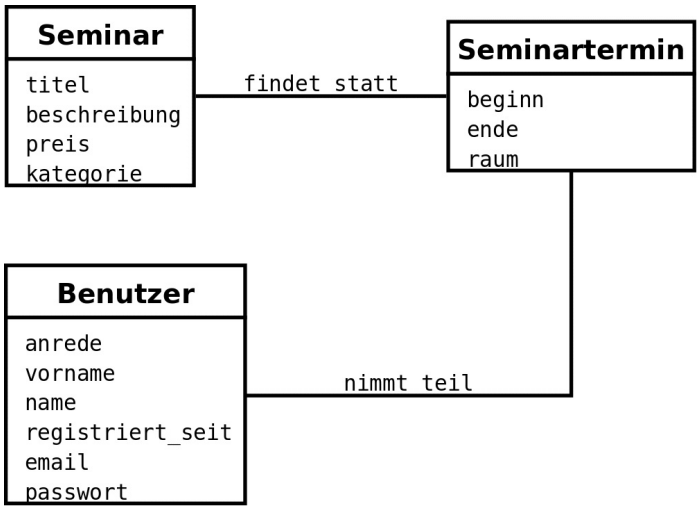


Abb. 21.3 Domänenmodell mit den Beziehungen *findet statt* und *nimmt teil*

21.2 Kardinalität

Für jede Beziehung müssen Sie noch eine sogenannte ***Kardinalität*** ermitteln. Die Kardinalität gibt an, in welcher Anzahl Einträge einander zugeordnet werden. Zu einem Seminar z. B. gehören mehrere Seminartermine, da das gleiche Seminar immer wieder mit neuen Gruppen zu anderen Terminen stattfindet. Aber zu jedem Seminartermin gehört nur ein Seminar. Ein Termin, wie z. B. der Datenbankenkurs vom 11.02.09 bis zum 14.02.09, kann nur zu einem Seminar gehören – dem Seminar *Datenbanken & SQL*. Der Termin kann nicht gleichzeitig zum Seminar *Grundlagen der PHP-Programmierung* gehören.

Für einen Datenbankentwickler ist es nicht wichtig, ob das Seminar 3 mal, 4 mal oder 150 mal stattfindet. Es ist lediglich wichtig zu wissen, dass es mehr als einmal stattfinden kann[1] . Eine solche Beziehung heißt ***1:n-Beziehung*** – gesprochen: *eins-zu-n-Beziehung*.

---

1 In dieser Hinsicht hat es ein Datenbankentwickler sogar einfacher als Conan, der Barbar. Conan kann nur bis drei zählen: »Eins, zwei, drei … ganz viele«. Ein Entwickler muss lediglich zwischen 1 oder mehreren unterscheiden.

Sie müssen jede Beziehung immer aus zwei Perspektiven betrachten – immer jeweils einmal aus der Sicht der beteiligten Entities. Deswegen ist es bei der 1:n-Beziehung wichtig, auf welcher Seite die *1* und auf welcher das *n* steht.

Betrachten Sie nun noch die Beziehung zwischen Benutzern und Seminarterminen. Ein Benutzer kann mehrere Termine besuchen. Ein Termin wird von mehr als einer Person besucht. In diesem Fall liegt eine sogenannte ***n:m-Beziehung*** vor. Beide Buchstaben stehen jeweils für eine Zahl unbekannter Größe.

Hier nochmal eine Übersicht über die Beziehungstypen:

Kardinalität der Beziehung	Beispiel
1:n	Zu einem Seminar finden mehrere Seminarveranstaltungen statt. In jeder Seminarveranstaltung wird nur ein Seminar unterrichtet.
n:m	Ein Benutzer kann an mehreren Seminarveranstaltungen teilnehmen. In einer Seminarveranstaltung sitzen mehrere Benutzer.

Tabelle 21.1 Beziehungen unterschiedlicher Kardinalität

In UML beschriften Sie die n-Seite einer Beziehung mit einem Stern »*«.

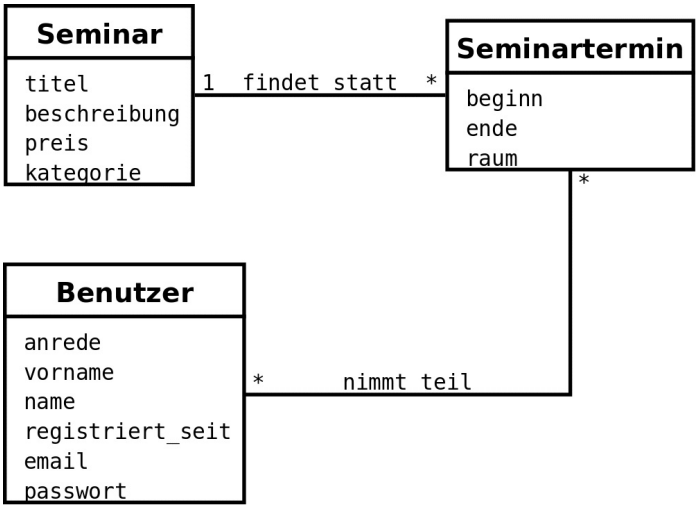


Abb. 21.4 Domänenmodell mit Kardinalität

Zusammenfassung

- Beziehungen stellen eine Verbindung zwischen Entities her.
- Die Namen von Beziehungen sind normalerweise Verben. Zusammen mit den dazugehörigen Entities einer Beziehung können Sie einen Satz bilden.
- Die Kardinalität gibt die Anzahl von Datensätzen an, die in einer Beziehung beteiligt sind.
- Beziehungen lassen sich anhand ihrer Kardinalität in 1:n und n:m unterteilen.

21.3 Aufgaben zur Selbstkontrolle

Testen Sie Ihr Wissen

1. Warum wird die Kardinalität nicht genauer bestimmt, z.B. 1:7 statt 1:n?

Übungen

Aufgabe 1: Beziehungen im Filmverleih

> *Wir benötigen nun die Möglichkeit, Filme anhand des Regisseurs zu suchen. Pro Film gibt es immer genau einen. Manche unserer Kunden möchten auch nach Schauspielern suchen. Die Schauspieler erfassen wir ebenfalls mit Vor-, Nachname und Geburtsdatum. Natürlich können in einem Film auch mehrere Schauspieler mitspielen.*
>
> *Außerdem muss für jeden Film festgelegt werden, in welcher Preiskategorie (günstig – 0,50 €, beliebt – 1 € oder neu – 2 €) er sich befindet, damit wir Rechnungen schreiben können.*

- Erweitern Sie das Domänenmodell gemäß der obigen Kundenanforderungen.
- Bestimmen Sie die Kardinalitäten.
- Überlegen Sie, ob es eventuell Möglichkeiten gibt, schon bei der Modellierung Redundanzen zu vermeiden.

Zusatzübungen

Aufgabe 2: Beziehungen bei der Fluggesellschaft

Die Datenbank muss erfassen, welcher Pilot welchen Flugzeugtyp fliegen kann. Passen Sie das Domänenmodell entsprechend an.

22 1:n-Beziehungen in SQL

In dieser Lektion lernen Sie:

- was Fremdschlüssel sind, und wie Sie sie in UML kennzeichnen.
- wie Sie 1:n-Beziehungen im relationalen Modell anlegen.
- wie Sie SELECTs über zwei Tabellen realisieren.

Beziehungen hat jeder, aber nicht jeder die richtigen.

Prof. Michael Marie Jung
*1940, deutscher Hochschullehrer

Nachdem Sie die Beziehungen im Domänenmodell erstellt haben, ist es nun Ihr Ziel, sie ins physische Datenmodell zu übertragen. Erinnern Sie sich noch an die Übertragung der Entities vom Domänenmodell ins relationale Datenbankmodell?[1] Nun ist es an der Zeit, auch die Beziehungen zu überführen. Bei den Beziehungen kommt es im Wesentlichen darauf an, welche Kardinalität vorliegt.

---

1 Falls nicht, werfen Sie doch noch mal einen Blick in Abschnitt 6.3.

Übertragen Sie zuerst die 1:n-Beziehungen. Im UML-Diagramm des physischen Datenbankmodells gibt es keine Verbindungslinien zwischen den Tabellen, da die Beziehungen hier nicht fest sind (siehe Abb. 22.1). Stattdessen stellen sie die gewünschten Verbindungen erst beim SELECT her. Sie benötigen dazu einen sogenannten ***Fremdschlüssel*** (***foreign key***).

22.1 Fremdschlüssel

Ein Fremdschlüssel ist ein Attribut[1] , das in einer anderen Tabelle Primärschlüssel ist. Sie erzeugen ihn aus dem Primärschlüssel der 1-Seite der Beziehung und fügen ihn der Tabelle der n-Seite hinzu. Für den Seminartermin bedeutet das, dass Sie die *id* der Seminare in der Tabelle *seminartermine* hinterlegen.

---

1 oder auch eine Menge von Attributen

Sie dürfen natürlich keine zwei Attribute in der Tabelle haben, die beide *id* heißen. Deswegen hat sich eine Benennungsrichtlinie eingebürgert. Üblicherweise benennen Sie einen Fremdschlüssel nach folgendem Schema:

```
<Name eines Elements der Ursprungstabelle>_id
```

Damit lautet der Fremdschlüssel hier `seminar_id`. In UML kennzeichnen Sie einen Fremdschlüssel durch den Zusatz `<<FK>>`.

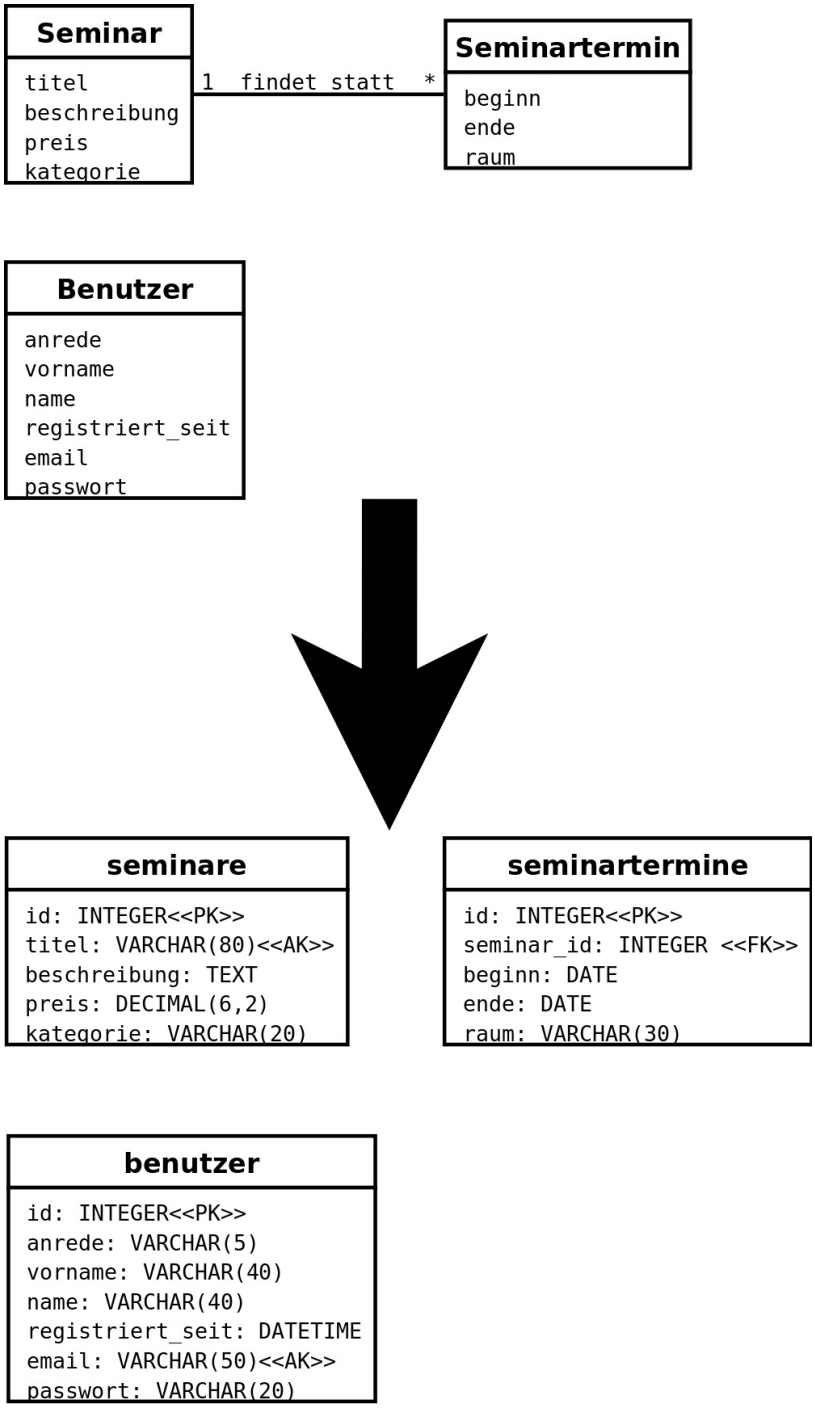


Abb. 22.1 Transformation einer 1:n-Beziehung

Wieso bildet der Fremdschlüssel die 1:n-Beziehung ab? Schauen Sie sich dazu ein paar Daten an:

Seminare

id	titel	beschreibung	preis
1	Relationale Datenbanken	Nahezu alle...	975,00
2	Ruby on Rails	Ruby on ...	2500,00
3	Ajax & DOM-Scripting	Ajax ist läng..	1699,99
4	JavaScript	JavaScript ist...	2500,00

Seminartermine

id	seminar_id	beginn	ende	raum
1	1	**2005-06-20**	2005-06-25	Schulungsraum 1
2	1	2005-11-07	2005-11-12	Schulungsraum 2
3	1	2006-03-20	2006-03-25	Schulungsraum 1
4	1	2006-12-04	2006-12-09	Besprechungsraum
5	4	**2005-01-17**	2005-01-24	Schulungsraum 1
6	4	2005-05-31	2005-06-07	Aula
7	4	2005-10-17	2005-10-24	Schulungsraum 2

Abb. 22.2 Daten in einer 1:n-Beziehung

Anhand des Fremdschlüssels *seminar_id* können Sie die Zusammenhänge erkennen. *seminar_id* dient jedem Termin als eine Referenz, die auf die *id* des jeweiligen Seminars verweist. So können Sie den Daten beispielsweise entnehmen, dass der Termin vom **20.06.2005** das Seminar *Datenbanken & SQL* zum Inhalt hat. Zum **17.01.2005** findet stattdessen ein *JavaScript* -Seminar statt. Umgekehrt sehen Sie auch, dass es für das Seminar *Datenbanken & SQL* 4 Termine gibt und für *JavaScript* 3.

Nun wird deutlich, warum der Fremdschlüssel auf der n-Seite eingeordnet ist. Ein Seminar kann mehrere Termine haben. Technisch bedeutet das, dass mehrere Termindatensätze in der Spalte *seminar_id* den gleichen Wert haben können – d. h. sich auf das gleiche Seminar beziehen. Umgekehrt kann aber jeder Termin immer nur zu einem Seminar gehören, da Sie als *seminar_id* immer nur **eine** Zahl eintragen können.

22.2 Die 1:n-Beziehung in SQL

Verwirklichen Sie das physische Datenmodell nun mit SQL. Erzeugen Sie dazu die Tabelle *seminartermine* – einschließlich des Fremdschlüssels.

```
CREATE TABLE seminartermine (
    id INTEGER PRIMARY KEY AUTO_INCREMENT,
    beginn DATE, ende DATE, raum VARCHAR(30),
    seminar_id INTEGER );
```

Listing 22.1 004_erzeuge_seminarveranstaltungen.sql

Fügen Sie ein paar Beispieldaten mit folgendem Skript ein:

```
INSERT INTO seminartermine (beginn, ende, raum, seminar_id)
VALUES ('2005-06-20', '2005-06-25', 'Schulungsraum 1', 1);
INSERT INTO seminartermine (beginn, ende, raum, seminar_id)
VALUES ('2005-11-07', '2005-11-12', 'Schulungsraum2', 1);
INSERT INTO seminartermine (beginn, ende, raum, seminar_id)
VALUES ('2006-03-20', '2006-03-25', 'Schulungsraum1', 1);
INSERT INTO seminartermine (beginn, ende, raum, seminar_id)
VALUES ('2006-12-04', '2006-12-09', 'Besprechungsraum', 1);
INSERT INTO seminartermine (beginn, ende, raum, seminar_id)
VALUES ('2005-01-17', '2005-01-24', 'Schulungsraum1', 4);
INSERT INTO seminartermine (beginn, ende, raum, seminar_id)
VALUES ('2005-05-31', '2005-06-07', 'Aula', 4);
INSERT INTO seminartermine (beginn, ende, raum, seminar_id)
VALUES ('2005-10-17', '2005-10-24', 'Schulungsraum2', 4);
```

Listing 22.2 seminartermine.sql

Durch die Beziehung, die Sie mit Hilfe des Fremdschlüssels hergestellt haben, können Sie die Daten der beiden Tabellen *seminare* und *seminartermine* nun auch in SQL verbinden (engl. ***join***). Dazu benötigen Sie einen sogenannten `JOIN`. SQL weiß allerdings nicht automatisch, dass der Fremdschlüssel und der Schlüssel zusammengehören. Sie müssen das bei Ihrer Anfrage angeben. Dazu dient das SQL-Schlüsselwort `ON`. Zum Auflisten der Termine mit den zugehörigen Seminardaten schreiben Sie:

```
SELECT seminare.titel, seminare.preis,
       seminartermine.beginn, seminartermine.ende,
       seminartermine.raum
FROM seminare JOIN seminartermine
ON seminare.id = seminartermine.seminar_id;
```

Hinter dem `FROM` haben Sie bisher immer nur eine Tabelle angegeben. In dieser SQL-Anweisung geben Sie nun zwei Tabellen an, die Sie mit `JOIN` verbinden. Die Bedingung hinter dem ON ist die sogenannte ***Join-Bedingung***. Sie ordnet Fremd- und Primärschlüssel einander zu. Da Sie zwei Tabellen in der Anfrage verwenden, müssen Sie außerdem für jede Spalte angeben, in welcher Tabelle sie sich befindet. Dazu dient die ***Punkt-Notation***. `seminare.id` meint die Spalte *id* aus der Tabelle *seminare*. Mit `seminartermine.seminar_id` ist entsprechend die Spalte *seminar_id* der Tabelle *seminartermine* gemeint. Genauso geben Sie hinter dem `SELECT` an, aus welcher Tabelle die Spalten kommen, die Sie in der Ergebnistabelle benötigen. Hier das Resultat:

```
+------------------+---------+------------+------------+--------+
| titel            | preis   | beginn     | ende       | raum   |
+------------------+---------+------------+------------+--------+
| Datenbanken & SQL | 975.00  | 2005-06-20 | 2005-06-25 | SR 1   |
| Datenbanken & SQL | 975.00  | 2005-11-07 | 2005-11-12 | SR 2   |
| Datenbanken & SQL | 975.00  | 2006-03-20 | 2006-03-25 | SR 1   |
| Datenbanken & SQL | 975.00  | 2006-12-04 | 2006-12-09 | Bespr. |
| JavaScript       | 2500.00 | 2005-01-17 | 2005-01-24 | SR 1   |
| JavaScript       | 2500.00 | 2005-05-31 | 2005-06-07 | Aula   |
| JavaScript       | 2500.00 | 2005-10-17 | 2005-10-24 | SR 2   |
+------------------+---------+------------+------------+--------+
```

```
7 rows in set (0.00 sec)
```

Wie Sie sehen, sind alle Termine gelistet – mit den benötigten Spalten (hier: *titel* und *preis*) des jeweiligen Seminars.

22.3 JOIN und WHERE

Natürlich lassen sich `JOIN` und `WHERE` kombinieren. Sie können somit z. B. folgende Suchanfrage stellen:

- Zeige mir alle Seminartermine, die in Schulungsraum 1 stattfinden bzw. stattfanden.

Ergänzen Sie einfach die WHERE-Bedingung:

```
SELECT seminare.titel, seminare.preis, beginn, ende, raum FROM seminare
JOIN seminartermine ON seminare.id = seminartermine.seminar_id WHERE raum =
'Schulungsraum 1';
```

```
+------------------+---------+------------+------------+------+
| titel            | preis   | beginn     | ende       | raum |
+------------------+---------+------------+------------+------+
| Datenbanken & SQL | 975.00  | 2005-06-20 | 2005-06-25 | SR 1 |
| Datenbanken & SQL | 975.00  | 2006-03-20 | 2006-03-25 | SR 1 |
| JavaScript       | 2500.00 | 2005-01-17 | 2005-01-24 | SR 1 |
+------------------+---------+------------+------------+------+
3 rows in set (0.00 sec)
```

22.4 BNF

Hier noch die erweiterte BNF von SELECT:

```
SELECT [DISTINCT] select_expr [AS alias_name] [,select_expr
[AS alias_name] ...]
[FROM table_reference]
[WHERE where_condition]
[ORDER BY col_name [ASC | DESC], ...]
[LIMIT [offset,] row_count]
table_reference: table_name | join_table
join_table: table_name JOIN table_name [join_condition]
join_condition: ON conditional_expr
```

Listing 22.3 BNF von SELECT mit JOIN

Damit ist die BNF für SELECT nun komplett – zumindest für dieses Buch. Insgesamt bietet SQL noch mehr Möglichkeiten. Es können aber nicht alle im Rahmen dieser Einführung behandelt werden. Außerdem sind viele eher exotischer Natur und finden in der Praxis kaum Verwendung.

Zusammenfassung

- 1:n-Beziehungen können Sie mit Hilfe von Fremdschlüsseln realisieren.
- Ein Fremdschlüssel entspricht dem Primärschlüssel einer anderen Tabelle.
- Mit einem JOIN können Sie Daten aus mehreren Tabellen abfragen.

22.5 Aufgaben zur Selbstkontrolle

Testen Sie Ihr Wissen

1. Kennt ein Datenbankschema Beziehungen?
2. Wie nennen Sie einen Fremdschlüssel, der auf eine *id* in der Tabelle *buecher* zeigt?

Übungen

Aufgabe 1: 1:n-Beziehungen im Filmverleih

- Erweitern Sie das physische Datenbankmodell des Filmverleihs um die 1:n-Beziehungen aus Ihrem Domänenmodell.
- Schreiben Sie ein neues Migrationsskript, das die 1:n-Beziehungen erzeugt.

Aufgabe 2: Joins

Hinweis: Für die folgende Übung können Sie die Beispieldaten aus dem Übungsmaterial importieren.

DROP DATABASE filmverleih;

CREATE DATABASE filmverleih;

mysql -u root filmverleih < filmverleih.sql

1. Geben Sie alle Filme mit ihrem Regisseur aus.
2. Geben Sie die Titel aller Filme aus, bei denen *Peter Jackson* Regie geführt hat.

23 n:m-Beziehungen in SQL

In dieser Lektion lernen Sie:

- wie Sie n:m-Beziehungen im relationalen Modell anlegen.
- wie Sie JOINs über beliebig viele Tabellen realisieren.

Auch in der Natur ist alles irgendwie geordnet, wenn auch in verschiedener Weise… Und es ist nicht so, dass eines beziehungslos neben dem anderen stünde, sondern überall gibt es Beziehungen. Auf ein Ziel hin ist alles in der Welt gerichtet.

Aristoteles

Nachdem Sie die 1:n-Beziehung erfolgreich übertragen haben, können Sie sich nun an die n:m-Beziehung wagen. Ein Fremdschlüssel reicht in diesem Fall aber leider nicht aus. Um eine n:m-Beziehung im relationalen Modell abzubilden, benötigen Sie eine ***Zwischentabelle***.

23.1 Die Zwischentabelle

In der Zwischentabelle sortieren Sie die Fremdschlüssel der beiden beteiligten Tabellen ein. Als Name für die Tabelle bietet sich der Name der Beziehung an: *nimmt_teil*. Die Zwischentabelle benötigt auch keine eigene id. Stattdessen verwenden Sie einen Primärschlüssel, der sich aus den beiden Fremdschlüsseln zusammensetzt.

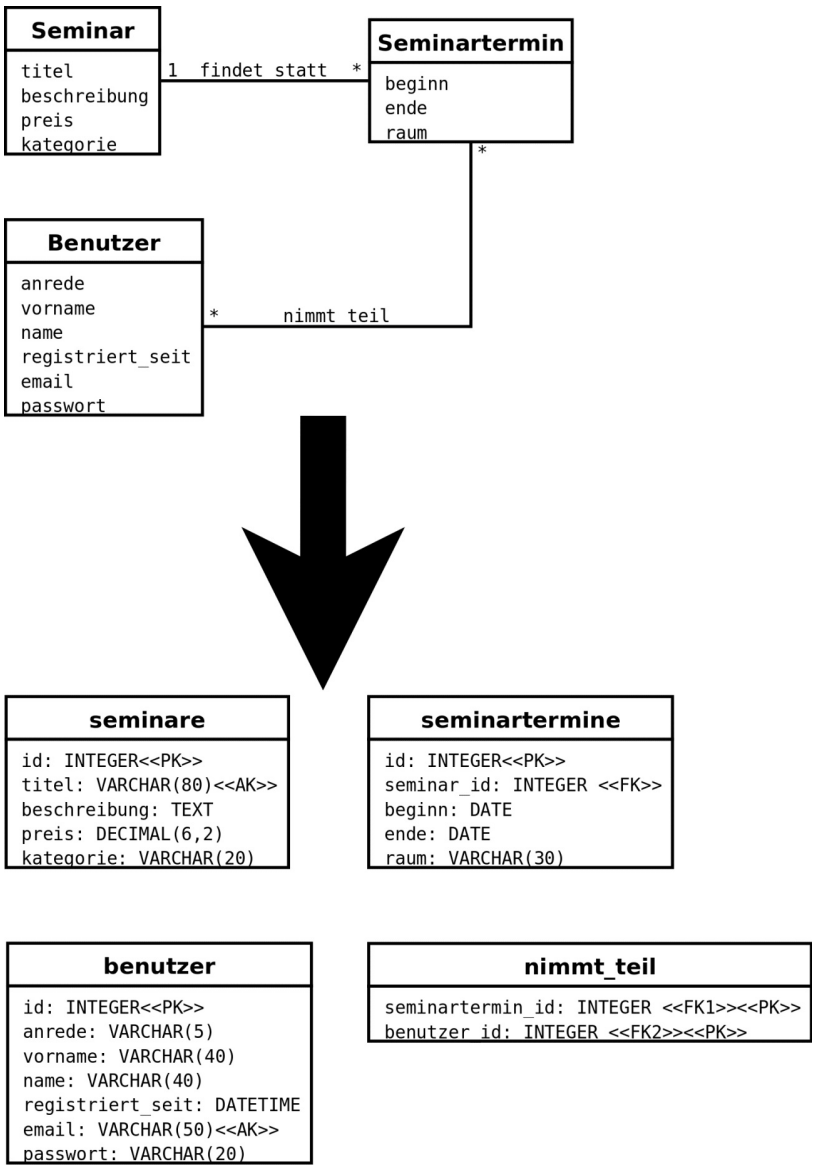
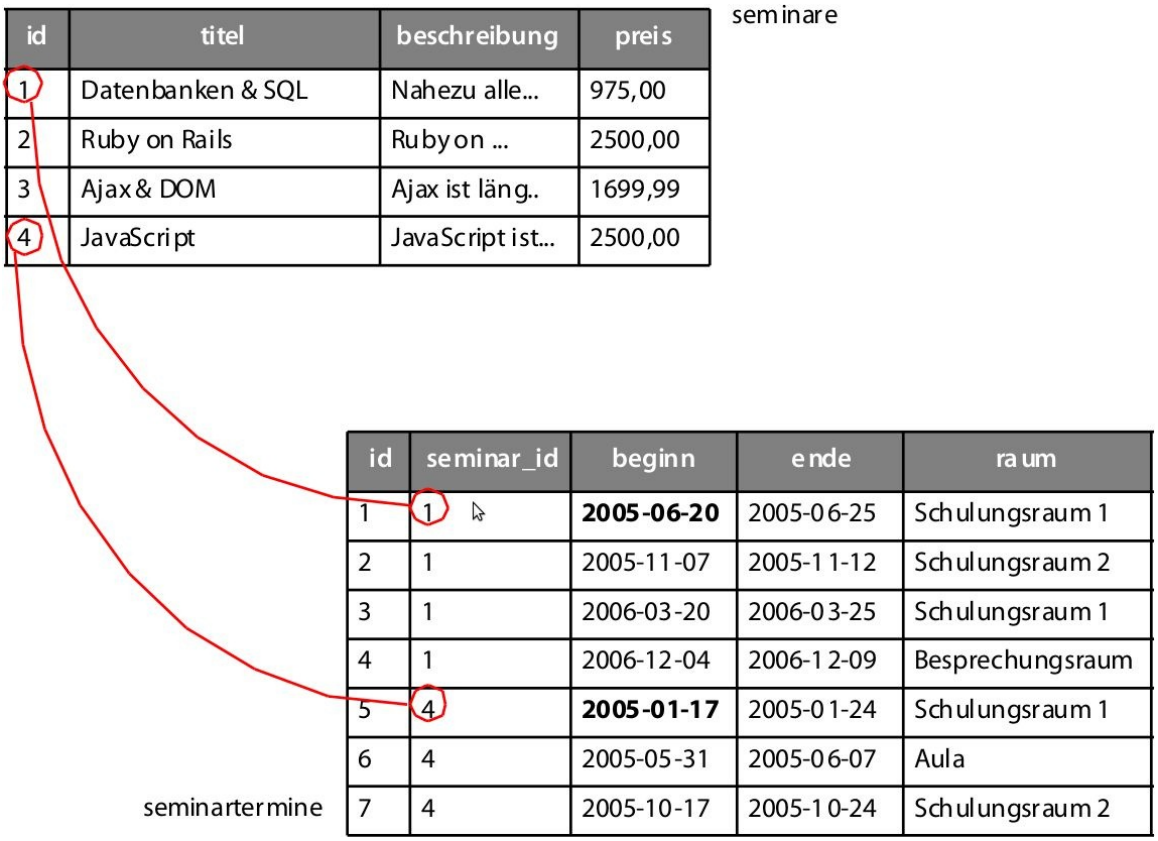


Abb. 23.1 Transformation einer n:m-Beziehung

Beachten Sie bitte, dass es zwei Fremdschlüssel sind (*{seminartermin_id}* und *{benutzer_id}*), aber nur ein Primärschlüssel (*{benutzer_id, seminartermin_id}*) – zusammengesetzt aus zwei Attributen.

Betrachten Sie folgendes Beispiel:



seminare

id	titel	beschreibung	preis
1	Datenbanken & SQL	Nahezu alle...	975,00
2	Ruby on Rails	Ruby on ...	2500,00
3	Ajax & DOM	Ajax ist läng..	1699,99
4	JavaScript	JavaScript ist...	2500,00

seminartermine

id	seminar_id	beginn	ende	raum
1	1	**2005-06-20**	2005-06-25	Schulungsraum 1
2	1	2005-11-07	2005-11-12	Schulungsraum 2
3	1	2006-03-20	2006-03-25	Schulungsraum 1
4	1	2006-12-04	2006-12-09	Besprechungsraum
5	4	**2005-01-17**	2005-01-24	Schulungsraum 1
6	4	2005-05-31	2005-06-07	Aula
7	4	2005-10-17	2005-10-24	Schulungsraum 2

Abb. 23.2 Daten in einer n:m-Beziehung

Die Tabellen *benutzer* und *seminartermine* sind durch die Zwischentabelle *nimmt_teil* und deren Fremdschlüssel miteinander verbunden. Sie sehen z. B. anhand der Zwischentabelle, dass Herr Reich die Seminartermine vom 07.11.2005 und vom 31.05.2005 belegt hatte. Genauso können Sie auch ablesen, dass zum Seminartermin vom 07.11.2005 neben Herrn Reich auch Frau Huana und Herr Meisenbär im Schulungsraum 2 saßen. Mit Hilfe der Zwischentabelle lassen sich auf diese Weise beliebige Beziehungen darstellen.

23.2 Die n:m-Beziehung in SQL

Verwirklichen Sie die neue Beziehung nun wiederum mit SQL. Erzeugen Sie dazu die Zwischentabelle *nimmt_teil*.

```
CREATE TABLE nimmt_teil (
    benutzer_id INTEGER, seminartermin_id INTEGER,
    PRIMARY KEY (benutzer_id, seminartermin_id));
```

Listing 23.1 005_erzeuge_nimmt_teil.sql

Fügen Sie ein paar Beispieldaten mit folgendem Skript ein:

```
INSERT INTO nimmt_teil (benutzer_id, seminartermin_id) VALUES (1, 2);
INSERT INTO nimmt_teil (benutzer_id, seminartermin_id) VALUES (1, 1);
INSERT INTO nimmt_teil (benutzer_id, seminartermin_id) VALUES (2, 2);
INSERT INTO nimmt_teil (benutzer_id, seminartermin_id) VALUES (3, 2);
```

Listing 23.2 nimmt_teil.sql

Um beim SELECT die Verbindung wieder herzustellen, müssen Sie alle drei Tabellen mit JOIN verbinden. Zudem benötigen Sie nun zwei Join-Bedingungen:

```
SELECT seminartermine.beginn, seminartermine.raum, benutzer.name,
benutzer.vorname FROM seminartermine JOIN nimmt_teil ON seminartermine.id =
nimmt_teil.seminartermin_id JOIN benutzer ON nimmt_teil.benutzer_id =
benutzer.id;
```

```
+------------+-----------------+------------+---------+
| beginn     | raum            | name       | vorname |
+------------+-----------------+------------+---------+
| 2005-06-20 | Schulungsraum 1 | Reich      | Frank   |
| 2005-11-07 | Schulungsraum 2 | Reich      | Frank   |
| 2005-11-07 | Schulungsraum 2 | Huana      | Marie   |
| 2005-11-07 | Schulungsraum 2 | Meisenbär  | Andreas |
+------------+-----------------+------------+---------+
4 rows in set (0.00 sec)
```

23.3 n:m-Beziehungen in Entities überführen

In einigen Fällen ist es notwendig, dass Beziehungen selbst wiederum Attribute führen. Betrachten Sie dazu folgende Anforderung:

Wir haben festgestellt, dass wir nun verpflichtet sind, für jeden Teilnehmer die Anwesenheit zu erfassen. In Zukunft müssen wir für jeden Teilnehmer (d.h. Benutzer) eintragen, wie viele Stunden er ein bestimmtes Seminar besucht hat.

Die Information, wie viele Stunden ein Teilnehmer ein Seminar besucht hat, lässt sich zweifellos in einem Attribut vom Typ INTEGER speichern. Aber in welcher Tabelle hinterlegen Sie das Attribut?

Würden Sie es beim Benutzer hinterlegen, so ließe sich nicht mehr feststellen, auf welchen Seminartermin es sich bezieht. Hinterlegen Sie es beim Seminartermin, ist unklar, welcher Benutzer gemeint ist.

Sie benötigen den Bezug zu beiden Tabellen. Die klassische Lösung ist deswegen, das Attribut der Beziehung zuzuordnen. Das würde bedeuten, die Zwischentabelle *nimmt_teil* erhält eine neue Spalte *anwesenheit_in_stunden*.

Leider lässt sich das jedoch im UML-Diagramm des Domänenmodells nur schwer darstellen. Außerdem sorgt dieses Konzept für Ärger beim objekt-relationalen Mapping. Deswegen ist es üblich, bei solchen oder ähnlichen Problemen, die Beziehung in ein Entity zu überführen. Statt der Beziehung *nimmt teil* entwerfen Sie das Entity *Teilnahme*. Diesem können Sie problemlos das neue Attribut zuordnen und haben später auch keine Schwierigkeiten beim Mapping. Zusätzlich benötigen Sie noch zwei 1:n-Beziehungen, die die Verbindung zu den anderen Entities herstellen. Die n-Seite dieser Beziehung liegt dann immer beim neu entstandenen Entity.

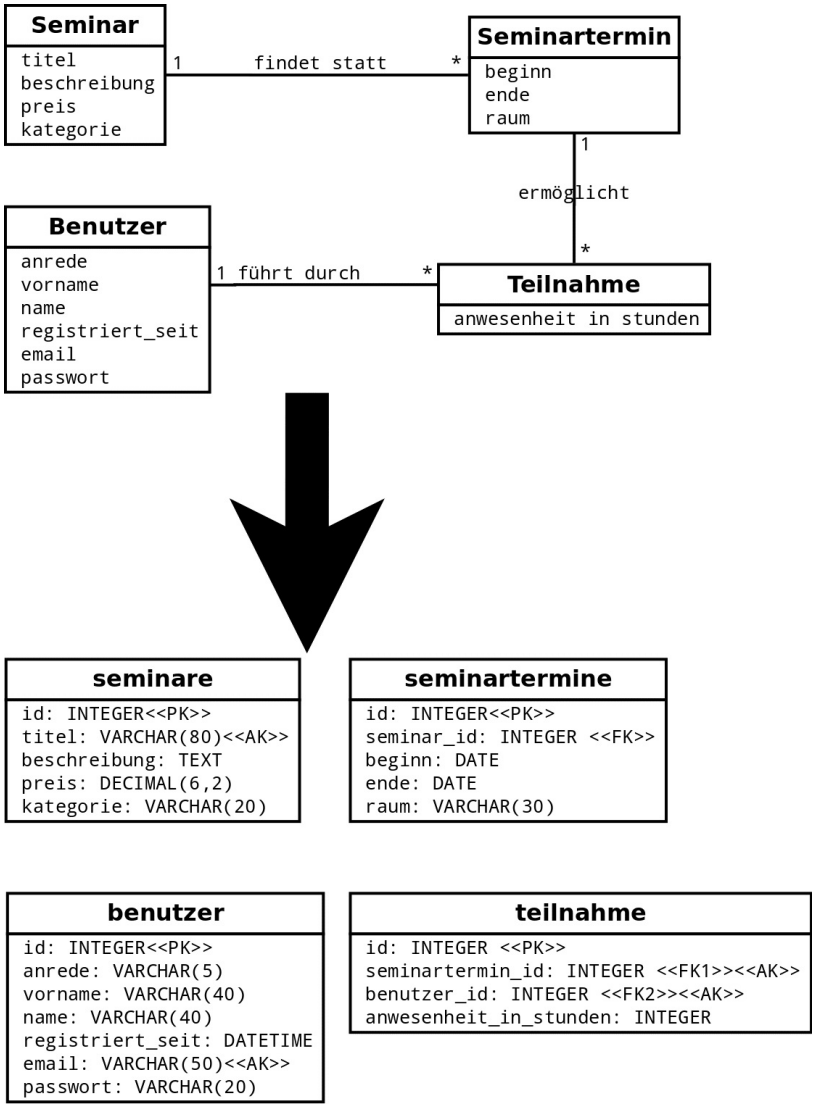


Abb. 23.3 Transformation einer Entity, die aus einer n:m-Beziehung entstandenen ist

Das resultierende physische Modell ist dem ursprünglichen recht ähnlich. Was im ursprünglichen Modell eine Zwischentabelle ist, die aus einer Beziehung entstanden ist, ist hier eine simple Entity-Tabelle. Beim OR-Mapping ergeben sich dann später aber handfeste Vorteile.

23.4 BNF

Die BNF für ein SELECT mit mehreren JOINs ist etwas komplexer, da Sie jetzt für jedes JOIN eine Tabellenreferenz einsetzen dürfen, die wiederum aus einem JOIN bestehen kann. Diese Konzept wird als ***rekursive*** Definition bezeichnet. Sie können somit JOINs beliebig oft verschachteln.

```
SELECT [DISTINCT] select_expr [AS alias_name] [,select_expr [AS alias_name] ...]
[FROM table_reference]
[WHERE where_condition]
[ORDER BY col_name [ASC | DESC], ...]
[LIMIT [offset,] row_count]
table_reference: table_name | join_table
join_table: table_reference JOIN table_name [join_condition]
join_condition: ON conditional_expr
```

Listing 23.3 BNF von SELECT mit mehreren JOINs

Zusammenfassung

- Eine n:m-Beziehung können Sie mit Hilfe einer Zwischentabelle abbilden.
- Der Primärschlüssel der Zwischentabelle setzt sich aus den beiden Fremdschlüsseln zu den beteiligten Tabellen zusammen.
- Beziehungen können selbst Attribute haben; aber besser ist es, solche Beziehungen als Entity darzustellen.

23.5 Aufgaben zur Selbstkontrolle

Testen Sie Ihr Wissen

1. Wie viele JOINs benötigen Sie, um eine einfache n:m-Beziehung abzufragen?
2. Wie viele JOINs sind in einer SELECT-Anweisung prinzipiell möglich?

Übungen

Aufgabe 1: n:m-Beziehungen im Filmverleih

- Erweitern Sie das physische Datenbankmodell des Filmverleihs um die n:m-Beziehungen aus Ihrem Domänenmodell.
- Schreiben Sie ein neues Migrationsskript, das die n:m-Beziehungen erzeugt.

Aufgabe 2: Joins

Hinweis: Für die folgende Übung können Sie die Beispieldaten aus dem Übungsmaterial importieren.

1. Geben Sie Filme mit hinterlegten Schauspielern aus.
2. Geben Sie alle Filme aus, in denen Christian Bale mitspielt.

Zusatzübungen

Aufgabe 3: Beziehungen bei der Fluggesellschaft

- Übertragen Sie die n:m-Beziehung aus Ihrem Domänen-Modell in ein physisches DB-Modell.
- Entwickeln Sie ein Migrationsskript, das die n:m-Beziehung aus dem physischen Modell in der Datenbank verwirklicht.

Aufgabe 4: Beziehungen bei der Fluggesellschaft: Flugstunden

Die Fluggesellschaft möchte nun noch genauer wissen, welcher Pilot wie viel Erfahrung mit welchem Flugzeugtyp hat. Dazu ist es notwendig, die Flugstunden zu erfassen, die ein Pilot auf einem bestimmten Flugzeugtyp gesammelt hat.

- Passen Sie Ihr Domänenmodell an, um das zu ermöglichen.
- Übernehmen Sie die Änderungen ins physische Modell.
- Schreiben Sie ein Migrationsskript, das die Änderungen durchführt.

Aufgabe 5: Beziehungen bei der Fluggesellschaft: Redundanzen

Haben Sie mehrere Arten von Flugstunden in Ihrer Datenbank? Falls ja, ist ihr DB-Design noch redundanzbehaftet. Entfernen Sie die Redundanz:

- Verbessern Sie Ihr Domänenmodell.
- Passen Sie Ihr physisches Datenbankmodell an.
- Entwickeln Sie ein entsprechendes Migrationsskript.
- Überlegen Sie sich die SELECT-Anweisungen, die nötig sind, um alle Informationen zu erhalten.

24 Anhang A: Befehlsübersicht

Konzept	Darstellung
Terminale	Wort in Großbuchstaben
Nichtterminale (Variablen)	Wort in Kleinbuchstaben
Wiederholung	…
Verschachtelte Definition	:
Optional	[]
Alternativ	|

Tabelle 24.1 BNF

Zweck	BNF
Datenbanken auflisten	`SHOW DATABASES`
Datenbank wechseln	`USE database_name`
Tabellen der aktuellen DB anzeigen	`SHOW TABLES`
Spalten einer Tabelle anzeigen	`SHOW COLUMNS FROM tbl_name`

Tabelle 24.2 Utility- und Administrations-Anweisungen

Zweck	BNF
Datenbank anlegen	`CREATE DATABASE db_name`
Datenbank löschen	`DROP DATABASE db_name`

Tabelle anlegen	`CREATE TABLE tbl_name (create_definition,...)`
	
	`create_definition:`
	`  col_name column_definition`
	`  | PRIMARY KEY (col_name,...)`
	`  | UNIQUE KEY (col_name,...)`
	
	`column_definition:`
	`  data_type [NOT NULL] [AUTO_INCREMENT] [UNIQUE KEY | PRIMARY KEY]`
	
	`data_type:`
	`  BOOLEAN | INTEGER | DECIMAL(length[,decimals]) | DATE`
	`  | TIME | DATETIME | YEAR | VARCHAR(length) | TEXT`

Tabelle löschen	`DROP TABLE tbl_name`

	`ALTER TABLE tbl_name alter_specification [, alter_specification] ...`
	
	`alter_specification:`

Tabellenschema ändern	``` ADD col_name column_definition | DROP col_name | DROP PRIMARY KEY | DROP KEY key_name | CHANGE old_col_name new_col_name column_definition | RENAME TO new_tbl_name ```

Tabelle 24.3 DDL-Anweisungen

Zweck	BNF
Datensätze auslesen	``` SELECT [DISTINCT] select_expr [AS alias_name] [,select_expr [AS alias_name] ...] [FROM table_reference] [WHERE where_condition] [ORDER BY col_name [ASC | DESC], ...] [LIMIT [offset,] row_count] table_reference: table_name | join_table join_table: table_reference JOIN table_name [join_condition] join_condition: ON conditional_expr ```
Datensatz einfügen	`INSERT INTO tbl_name (col_name,...) VALUES (value,...)`
Datensätze ändern	`UPDATE table_name SET col_name1 = value1 [, col_name2 = value2] [WHERE where_condition]`
Datensätze löschen	`DELETE FROM table_name [WHERE where_condition]`

Tabelle 24.4 DML-Anweisungen

25 Anhang B: Agilität in der Webentwicklung

In dieser Lektion lernen Sie:

- welche Rolle Agile Vorgehensmodelle in der Webentwicklung spielen.
- warum Sie auf Wasserfälle allergisch reagieren sollten.
- was iterativ-inkrementell bedeutet.

Agile Vorgehensmodelle

In der Softwareentwicklung gibt es zwei extreme Vorgehensweisen. Entweder Sie folgen einem bis ins kleinste Detail ausgearbeiteten Plan, oder Sie haben überhaupt keinen Plan und programmieren einfach »wild drauf los«. Bernd Oestereich (1998) nennt das die ***VHIT-Methode*** – Vom Hirn ins Terminal. Beide Extreme sind nicht gerade zu empfehlen.

Wenn Sie sich sofort auf den Code stürzen, verschwenden Sie meistens viel Zeit mit falschen Ansätzen, oder laufen gar in Sackgassen[1] . Oder Sie programmieren einfach tolle Features – die dann aber niemand wirklich braucht. Mit ein wenig Planung kann das alles vermieden werden.

---

1 d. h. Ihr Code lässt sich nicht mehr stabil weiterentwickeln

Das andere Extrem besteht darin, alles detailliert zu planen und genaue Methoden und Handlungsanleitungen vorzugeben. Das Ergebnis sind aber meist große Vorschriftenkataloge, Formalismen und Bürokratie. Ein Freund von mir hat bei einem großem Elektrokonzern Steuerungssoftware für medizinische Geräte programmiert. Er drückte das mal folgendermaßen aus:

> *Wir warten nur noch darauf, dass wir ein Formular ausfüllen müssen, wenn wir auf die Toilette gehen.*

Zu viele Formalismen vernichten massiv wertvolle Entwicklungszeit. Die ***Agile Softwareentwicklung*** bevorzugt dagegen einen Mittelweg, der nur so viel Plannung, Dokumentation und Formalismen verwendet, wie es für das Projektergebnis nützlich ist. Die wichtigsten Grundsätze beschreibt das *Manifesto for Agile Software Development*, das 17 bekannte Entwickler und Forscher[1] 2001 in Utah verfassten (Beck et al., 2001)[2]:

---

1 Martin Fowler z. B. ist einer davon.
2 agilemanifesto.org

- Menschen und Interaktionen sind wichtiger als Prozesse und Werkzeuge.
- Laufende Systeme sind wichtiger als umfassende Dokumentation.
- Die Zusammenarbeit mit dem Kunden ist wichtiger als Vertragsverhandlungen.
- Die Fähigkeit, auf Änderungen zur reagieren ist wichtiger als das Verfolgen eines

Plans.

> *Auch wenn die Punkte auf der rechten Seite wichtig sind, so erachten wir doch die Punkte auf der linken Seite als wichtiger.*

Ich möchte diese Sätze hier einfach mal so stehen lassen. Es gibt noch sehr viel mehr zum agilen Vorgehen zu sagen. Dieses Buch hat aber Datenbanken zum Thema und bietet nicht den Platz, um Agile Softwareentwicklung im Detail zu besprechen. Deswegen möchte ich Sie an dieser Stelle auf weiterführende Literatur verweisen:

- Hruschka P. , Rupp C., Starke G. (2003). Agility kompakt: Tipps für erfolgreiche Systementwicklung. Spektrum Akademischer Verlag.
- Hunt A., Subramaniam V. (2006). Practices of an Agile Developer. The Pragmatic Programmers.
- Highsmith J. (2002). Agile Software Development Ecosystems. Pearson Education.
- Wallace D., Ragget I., Aufgang J. (2003). Extreme Programming for Web Projects. XP Series. Addison-Wesley.
- Fowler M. (2005). The New Methology. Siehe www.martinfowler.com/articles/newMethodology.htm
- Der deutsche Wikipedia Artikel [1] über Agile Softwareentwicklung

---

1 de.wikipedia.org/wiki/Agile_Softwareentwicklung

Es gibt außerdem eine Reihe ***Agiler Vorgehensmodelle***, die jeweils einen Prozess für die Softwareentwicklung beschreiben. Hier eine kleine Auswahl:

- Crystal Clear
- Extreme Programming (XP)
- Feature Driven Development (FDD)
- ICONIX
- Scrum

Welcher Prozess sich für Ihr Projekt am besten eignet, ist vor allem vom Entwicklerteam, der Beziehung zum Kunden, den Umgebungsbedingungen und der Art des Projektes abhängig.

Generell bin ich der Meinung, dass es kaum Projekte gibt, die nicht von einem agilen Vorgehensmodell profitieren können[1]. Gerade Webprojekte sind ganz besonders für eine agile Vorgehensweise geeignet. Sowohl Anforderungen als auch verwendete Technologien entwickeln sich im Webbereich mit rasanter Geschwindigkeit. Wo für klassische Business-Software Entwicklungszyklen von 6 Monaten und mehr nichts Ungewöhnliches sind, werden im Web wesentlich kürze Zyklen verlangt. Die Anforderungen an eine Webanwendung ändern sich ständig, so wie sich der Markt und die Technologien ändern. Die meisten klassischen Vorgehensmodelle (z. B. V-Modell, Wasserfall) betrachten solche Änderungen als Störung. Sind bestimmte Plannungsphasen abgeschlossen, können Änderungswünsche nicht mehr (oder nur noch mit sehr hohem Aufwand) berücksichtigt werden. Für agile Vorgehensmodelle ist die Veränderung dagegen Normalität. Responding to change over following a plan[2], heißt es im Agile Manifesto (Beck et al. 2001). Embrace

Change[3] ist sogar der Leitspruch des Extreme Programmings (Beck & Andres 2004).

---

1 Die Ausnahme könnten Projekte mit sehr großen Entwicklungsteams oder extrem hohen Sicherheitsanforderungen sein (z. B. militärische Steuerungssoftware für Helikopter).
2 zu deutsch: »Auf Änderungen zu reagieren ist wichtiger als einem Plan zu folgen.«
3 zu deutsch: »Umarme die Veränderung«, oder bessser sinngemäß »Begrüße die Veränderung«

Auch wenn dieses Buch nicht genügend Raum bietet, um agile Vorgehensmodelle umfassend zu erläutern, so gibt es doch einen Aspekt, der für datenbankbasierte Webanwendungen eine besonders wichtige Rolle spielt. Die Rede ist vom sogenannten ***Iterativ-Inkrementellen Vorgehen***.

Das iterativ-inkrementelle Vorgehen

Das iterativ-inkrementelle Vorgehen zeichnet jedes agile Vorgehensmodell aus. Auch wenn sich die verschiedenen agilen Vorgehensmodelle in vielen Punkten unterscheiden, so ist das iterative-inkrementelle Vorgehen in jedem dieser Modelle[1] ein wichtiges Kernkonzept. Da dieses Vorgehen die klassische Entwicklung datenbankbasierter Webanwendungen geradezu auf den Kopf stellt, möchte ich es hier etwas ausführlicher erläutern. Es war einmal …

---

1 Zumindest kenne ich kein agiles Vorgehensmodell, auf das das nicht zutrifft.

Das Wasserfallmodell

Lange Zeit war das sogenannte ***Wasserfallmodell***[1] von Dr. Winston W. Royce (1970)[2] das meist-verbreitete Modell der Softwareentwicklung. Es geht auf das noch ältere Phasenmodell von Bennington (1956) zurück. Royce erkannte schon damals die Probleme des Modells und führte viele Verbesserungen an [3] – die die meisten Firmen dann jedoch völlig ignorierten.

---

1 Mehr zur Entstehung des Wasserfallmodells finden Sie bei Rerych M. (2002) unter cartoon.iguw.tuwien.ac.at/fit/fit01/wasserfall/entstehung.html.
2 siehe www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf
3 siehe de.wikipedia.org/wiki/Wasserfallmodell und en.wikipedia.org/wiki/Waterfall_model

Wie genau funktioniert das Wasserfallmodell?

Im Wasserfallmodell gibt es verschiedene aufeinander folgende Phasen. Eine Phase muss abgeschlossen sein, bevor die nächste beginnt. Jede Phase produziert ein Ergebnis, das als Ausgangspunkt der folgenden Phase gilt. Typischerweise erstellen die Entwickler in Zusammenarbeit[1] mit dem Kunden in der ersten Phase *Aufnahme der Anforderungen* ein Pflichten- und/oder Lastenheft. Dieses dient dann als Ausgangspunkt für das softwaretechnische Design. Ergebnisse des Designs sind meist Diagramme und Richtlinien. Danach erfolgt die eigentliche Implementierung, die das Design nun »nur noch« umsetzen soll. Die Phasen aus Royce' ursprünglichem Wasserfall waren wie folgt[2]:

---

1 Das ist schon der beste Fall. Leider kann es auch passieren, dass der Kunde das Heft selbst erstellt oder eine externe Firma beauftragt. Die Entwickler erhalten es dann – schlimmstenfalls kommentarlos – und müssen rätseln, wie die Anforderungen genau zu verstehen sind.

2 Entnommen aus Wikipedia[1]. Das *Standard Wasserfallmodell* der NASA verwendete einen alternativen Phasenplan. Sie

können den Plan im Webarchive unter web.archive.org/web/20040403211247/http://asd-www.larc.nasa.gov/barkstrom/public/The_Standard_Waterfall_Model_For_Systems_Development.htm einsehen.

1. Aufnahme der Anforderungen (Requirements)
2. Design
3. Implementierung/Programmierung (Implementation)
4. Integration
5. Testen und Fehlerbereinigung (Verification)
6. Installation
7. Wartung (Maintenance)

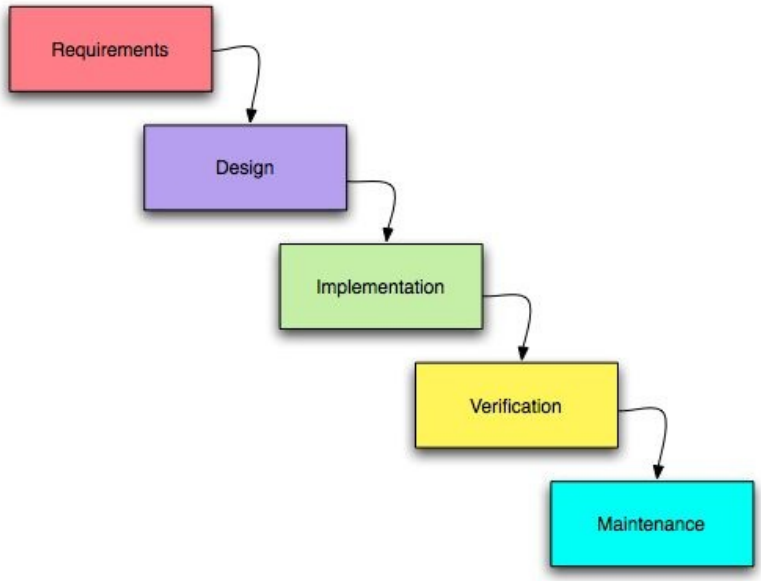


Abb. 25.1 Eine verkürzte Variante des Wasserfallmodells
entnommen aus Wikipedia[1]

Wenn Sie das Wasserfallmodell auf ein datenbankbasiertes Webprojekt anwenden, erhalten Sie beispielsweise folgende Phasen:

1. Aufnahme der Anforderungen (Ergebnis: Konzeptionelles Datenbank-Diagramm)
2. Logisches Datenbank-Design
3. Physisches Datenbank-Design
4. Website-Konzeption

In der Praxis sind außerdem Datenbankentwicklung und Anwendungsentwicklung oft getrennt. Es gibt Datenbankspezialisten auf der einen Seite und Softwareentwickler auf der anderen. Jede Gruppe befasst sich nur mit ihrem jeweiligen Teil der Anwendung. Oder schlimmer noch: Die Datenbank wurde gar nicht für diese Anwendung entwickelt, sondern enthält allgemeine Firmendaten. Die Anwendung muss dann »irgendwie« nachträglich auf diese zugreifen.

Unabhängig davon, bringt das Wasserfallmodell selbst auch einige erhebliche Nachteile mit sich.

Nachteile des Wasserfallmodells

- Änderungen sind kostspielig.

Änderungswünsche, die der Auftraggeber erst spät im Projektverlauf äußert, können sehr kostspielig sein. Für jede Änderung ist das Modell zurückzufahren. Die Änderung muss alle einzelnen Phasen von Anfang bis Ende durchlaufen. Durch die hohen Abhängigkeiten in der Softwareentwicklung müssen dann (je nach Umgang der Änderung) meist große Teile des Codes umgeschrieben werden.

- Fehlendes Feedback aus der Benutzung

Erst am Ende des Projekts liegt ein funktionsfähiges Produkt vor. Erst dann kann ein Benutzer das Produkt testen und Erfahrung sammeln. Diese Erkenntnisse kann das Wasserfallmodell aber nicht mehr berücksichtigen – oder nur in Form kostspieliger Änderungen einfließen lassen.

Dabei ist dem Auftraggeber aber meist zu Beginn des Projektes noch nicht bewusst, welche Features er tatsächlich benötigt, und wie sich diese am besten bedienen lassen.

- Fehlendes Feedback aus der Implementierung

Ob ein bestimmtes Softwaredesign funktioniert, stellt sich oft erst bei der Programmierung heraus. Dann ist es aber zu spät, diese Änderungen im Design zu berücksichtigen – das Design ist bereits abgeschlossen.

- Unrealistische Schätzungen

Aufwandsschätzungen, die auf einem Pflichtenheft basieren, sind meistens völlig unrealistisch. Sie müssten schon Hellseher sein, wenn Sie Aufwände über ein mehrmonatiges Projekt genau schätzen und planen wollen.

- Time to Market

Die fertige Webanwendung ist erst dann einsetzbar, wenn sie alle Phasen durchlaufen hat. D.h. bei einer Projektgröße von ca. sechs Monaten kann es ohne weiteres sein, dass Sie Ihrem Auftraggeber in den ersten fünf Monaten nur Diagramme und Powerpoint-Folien vorführen können.

Trotz der bekannten Nachteile und Probleme wenden traditionell veranlagte Firmen auch heute noch das Wasserfallmodell an. Oft wird es nicht als Wasserfallmodell bezeichnet oder versteckt sich hinter anderen, gut klingenden Bezeichnungen. Im Kern ist es aber dann dennoch ein Wasserfall. Sollten Sie das Pech haben, in einer solchen Firma zu arbeiten, kann ich Ihnen nur Scott Amblers Rat ans Herz legen.

Scott sagt…

> *Change your job or change your job (Ändern Sie Ihre Firma oder wechseln Sie Ihre Firma).*

Funktionsweise des iterativ-inkrementellen Vorgehens

Nachdem Sie nun wissen, was Sie besser unterlassen, sollen Sie natürlich auch erfahren, was eine vernünftige Vorgehensweise ausmacht.

Das Kernproblem des Wasserfallmodells besteht darin, dass es versucht, die ganze Komplexität eines Projektes auf einmal zu verdauen. So wird z. B. verlangt, dass alle Anforderungen bei Projektstart schon komplett verstanden und ausgearbeitet sind. In der Praxis moderner Webanwendungen sind aber bei Projektstart immer noch viele Fragen offen. Außerdem ist niemand perfekt. Auch Ihr Auftraggeber kann sich irren. Was sich heute wie eine sehr gute Idee anhört, kann in der Praxis sehr schlecht oder eventuell auch gar nicht funktionieren. Wenn Ihr Auftraggeber komplexe Features ausarbeitet, ist es hilfreich, wenn er die Anwendung schon einmal in der Praxis ausprobiert hat und sich nicht eine komplexe Anwendung vollständig im Kopf vorstellen muss.

Das Gleiche gilt auch für Architektur und Design Ihrer Webanwendung. Wenn Sie ein Softwaredesign für die ganze Anwendung erstellen, müssen Sie auch Bereiche, in den noch Unsicherheiten bezüglich der Anforderungen und der technischen Realisierung vorliegen, bereits vollständig ausarbeiten. Dieses Problem wird als ***BDUF*** (***Big Design Up Front-*** zu deutsch: Zuviel Design am Anfang) bezeichnet. Oftmals lernen Sie erst bei der Implementierung, wie gut bestimmte Aspekte Ihres Designs funktionieren. Beim iterativ-inkrementellen Vorgehen können Sie solche gewonnenen Erfahrungen einsetzen.

Arbeiten Sie in kleinen Schritten. Statt die komplette Anwendung auf einen Schlag zu konzipieren, konzipieren nur Sie einen kleinen Teil der Anwendung. Designen Sie dann auch nur einen kleinen Teil – und implementieren Sie schließlich nur diesen kleinen Teil. Sie können noch unklare Bereiche später bearbeiten, wenn Sie mehr wissen. Am wichtigsten ist jedoch, dass Sie Ihrem Auftraggeber schon nach kurzer Zeit einen kleinen – aber voll funktionstüchtigen! – Teil der Anwendung zeigen können. Er sieht, wie gut seine Ideen in der Praxis funktionieren. Deswegen kann er schon sehr früh Änderungswünsche äußern. Neue Features lassen sich auf Grund der gewonnenen Erfahrungen gezielter ausarbeiten.

Im Grunde genommen gibt es die Phasen des Wasserfallmodells immer noch. Das Projekt

durchläuft diese Phasen aber nicht nur einmalig und im Ganzen, sondern mehrmals in kleinen Schritten. Sie beginnen mit einigen Anforderungen, die Sie als Kernfeatures identifizieren. Nach der Anforderungsanalyse designen Sie den kleinen Teil Ihrer Anwendung, der diese Anforderungen abdeckt. Danach programmieren Sie diesen Teil und lieferen ihn an den Auftraggeber aus. Der kann den ersten Teil der Anwendung schon ausprobieren und mit mehr Erfahrung neue Features ausarbeiten. Der Zyklus beginnt erneut. Ein solcher Zyklus, in dem das Projekt alle Phasen einmal durchläuft, heißt ***Iteration***. Dadurch, dass in jeder Iteration dem Projekt etwas neues hinzugefügt wird, können Sie sagen: In jeder Iteration wächst das Projekt um ein ***Inkrement***. So setzt sich der Name der iterativ-inkrementellen-Vorgehensweise zusammen.

Das Beispielprojekt dieses Buchs wird nicht alle Phasen durchlaufen, die ein reales Projekt durchläuft. Insbesondere ist es von der Wahl des konkreten Vorgehensmodells (RUP, XP, Scrum etc.) abhängig, welche Phasen es gibt, und wie und wann Sie zwischen diesen wechseln. In vielen agilen Modellen ist keine bestimmte Reihenfolge vorgegeben. Im XP z. B. gibt es nicht einmal Phasen, es lässt sich dort nur zwischen der Aufnahme der Anforderungen (Planning Game) und der Implementierung unterscheiden. Unser Beispielprojekt setzt kein bestimmtes Vorgehensmodell ein. Sie werden es aber iterativ-inkrementell entwickeln.

Gehen Sie also bitte nicht davon aus, dass die in den Beispielen vorgestellten Aktivitäten und deren Reihenfolge zwingend sind. Sie stellen lediglich eine mögliche und außerdem unvollständige Vorgehensweise dar. Sie können sie aber als Einstiegspunkt adaptieren und später in ein Vorgehensmodell Ihrer Wahl einbinden.

Iterativ-inkrementelles Lernen im Spiralcurriculum

Ähnlich wie für das Entwickeln eines Projektes, ist es auch für das Lernen hilfreich, wenn Sie nicht einen riesigen Wissensbrocken auf einmal verdauen müssen. Sinnvoller ist es, kleine Teile zu lernen, diese auszuprobieren und schließlich zu meistern. Dann können Sie immer auf Ihrem bisherigen Stand aufbauen, um dann später Ihr Wissen und Ihre Fähigkeiten zu vertiefen und zu ergänzen. Das didaktische Konzept, das dem iterativ-inkrementellen Vorgehen am nächsten kommt, ist das ***Spiralcurriculum***[1] nach Bruner. Es dient als didaktischer Leitfaden dieses Buches und bietet mir die Möglichkeit, Ihnen die Webentwicklung in einer Reihenfolge zu vermitteln, die Sie auch in einem realen Projekt anwenden können. Im Sinne des Spiralcurriculums erkläre ich manche Themen oft erst einmal vereinfacht, um sie dann später zu vertiefen.

---

1 siehe de.wikipedia.org/wiki/Spiralcurriculum

26 Anhang C: Quellen & Literaturhinweise

26.1 APA-Style

Zum Aufzeigen von Referenzen, Quellen und weiterführender Literatur verwende ich den sogenannten ***APA-Style***. Der APA-Style ist – unter anderem – ein System zum Kennzeichnen von Referenzen. Entwickelt wurde dieses System von der *American Psychological Association* (APA).

Unter en.wikipedia.org/wiki/Apa_style können Sie sich noch genauer informieren. Eine APA-Style-Referenz besteht meist aus Nachname und Jahreszahl. Hier sehen Sie zwei Beispiele.

Beispiel

- *Generell lässt sich sagen, dass die meisten Webanwendungen Datenbanken grundsätzlich zur persistenten Speicherung verwenden (Fowler, 2003b).*
- *Ambler (2006) zeigt, dass es neben der technischen Kluft auch eine kulturelle gibt.*

In beiden Fällen beziehen sich die Angaben auf Quellen aus dem Referenzteil. Wenn Sie hinten im Referenzteil nachschlagen, finden Sie Folgendes:

Die erste Referenz ist eine Internetquelle, die zweite ein Buch. Bei Internetquellen gebe ich Ihnen im Text oder in einer Fußnote meist nochmal die URL an oder verzichte sogar auf den Apa-Style-Verweis. Dadurch können Sie die Seite direkt besuchen, ohne erst nachschlagen zu müssen. Außerdem lagere ich Quellenangaben oft vollständig in die Fußnote aus, um den Textfluss nicht zu stören.

26.2 Quellen

Ambler S. W. (2003) Agile Database Techniques: Effective Strategies for the Agile Software Developer. John Wiley & Sons

Ambler S. W., Sadalage P. (2006) Refactoring Databases: Evolutionary Database Design. Addison Wesley Professional

Ambler S. W. (Homepage) Scott W. Ambler's Home Page. Betrachtet am 28.08.2008 unter www.ambysoft.com/scottAmbler.html.

Ambler S.W. (August 2006) A UML Profile for Data Modeling. Agile Data Home Page. Betrachtet am 11.02.2009 unter www.agiledata.org/essays/umlDataModelingProfile.html.

Ainsworth S. (2009) Buchbesprechung im O'Reilly Online-Shop, siehe shop.oreilly.com/product/9780596523084.do

Aslett M. (2012) MySQL vs. NoSQL and NewSQL - survey results. Präsentation auf Slideshare. siehe www.slideshare.net/mattaslett/mysql-vs-nosql-and-newsql-survey-results-13073043

Beck K., Beedle M., van Bennekum A., Cockburn A., Cunningham W., Fowler M., Grenning J., Highs- mith J., Hunt A., Jeffries R., Kern J., Marick B., Robert C. M., Mellor S., Schwaber K., Sutherland J., Thomas D. (2001). Manifesto for Agile Software Development. Website, siehe agilemanifesto.org

Beck K., Andres C. (November 2004) Extreme Programming Explained: Embrace Change, Second Edition. Addison Wesley Professional

Bennington (Juni 1956) Production of Large Computer Programs. In: Proceedings of the ONR Symposium on Advanced Programming Methods for Digital Computers, Seite 15

Braun F. (Februar 2007) Grammar Types. Homepage von Frank Braun an der Universität Regensburg. Betrachtet am 10.09.2008 unter www-cgi.uni-regensburg.de/~brf09510/grammartypes.html

Campbell-Kelly M. (Mai 2003) Edgar Codd – Computer programmer who saw others profit from his inventing the relational database. Artikel in The Independent (online-Zeitschrift). Betrachtet am 09.09.2008 unter www.independent.co.uk/news/obituaries/edgar-codd-730256.html.

Celko Joe (2005) SQL Programming Style. Elsevier, Morgan Kaufmann Publishers.

Codd E. F. (1969) Derivability, Redundancy, and Consistency of Relations Stored in Large Data Banks. IBM Research Report RJ599

Codd E. F. (Juni 1970) A Relational Model of Data for Large Shared Data Banks. Communications of the ACM Volume 13, Issue 6: 377–387. Siehe www.seas.upenn.edu/~zives/03f/cis550/codd.pdf

Darrow B. (2004) Ted Codd, Chris Date. Online Artikel im CRN-Magazin. Veröffentlicht: 10.12.2004, siehe www.crn.com/news/channel-programs/55300688/ted-codd-chris-date.htm

Date C .J. (2004) An Introduction into Database Systems. 8th Edition, Pearson Education

Inc.

Date C. J. (2005) Database in Depth – Relational Theory for Practitioners. O'Reilly

Date C. J., Darwen H. (2007) Databases, Types and the Relational Model – The Third Manifesto. 3rd Edition, Addison Wesley, siehe auch thethirdmanifesto.com

Date C. J. (2007) Fachwörterbuch Relationale Datenbanken – kurz & gut. O'Reillys Taschenbibliothek. Deutsche Ausgabe von The Relational Database Dictonary (O'Reilly Media Inc. 2006)

Endlich S. (2012) The State of NoSQL. Fachartikel auf InfoQ.com. Betrachtet am 07.03.2013 unter www.infoq.com/articles/State-of-NoSQL.

Fowler M., Beck K., Brant J., Opdykeet. W. (1999) Refactoring, Improving the Design of Existing Code, Amsterdam: Addison-Wesley Longman

Fowler M. (2002) Patterns of Enterprise Application Architecture. Amsterdam: Addison-Wesley Longman

Fowler, M. (Februar 2003a) Domain Logic and SQL. Martin Fowlers persönliche Homepage. Betrachtet am 04.09.2008 unter www.martinfowler.com/articles/dblogic.html

Fowler M. (April 2003b) About Martin Fowler. Martin Fowlers persönliche Homepage. Betrachtet am 31.08.2008 unter martinfowler.com/aboutMe.html.

Fowler M. (2006) Keynote der RailsConf 2006. Betrachtet am 31.08.2008 unter www.scribemedia.org/2006/07/03/rails-martin-fowler

Gornik D. (Mai 2002) UML Data Modeling Profile. TP 162. Rational Software. IBM

Haigh T. (Juni 2007) Oral History of C. J. Date (Interview). Mountain View, California: Computer History Museum. X4090. 2007, siehe www.computerhistory.org/collections/accession/102658166

Horstmann J. (Februar 2006) Freie Datenbanken im Unternehmenseinsatz – Analyse und Vergleich der wichtigsten Open-Source-Datenbanken. Artikel auf OpenHeise, Heise Verlags-Website. Betrachtet am 09.09.2008 unter www.heise.de/open/Freie-Datenbanken-im-Unternehmenseinsatz-Ein-Vergleich—/artikel/70100/0

Kellerwessel H. (2002) Programmierrichtlinien in der Praxis. MITP

Muller R. J. (1999) Database Design for Smarties: Using UML for Data Modelling. Morgan Kaufmann Publishers

MySQL AB (2008) MySQL 5.1 Reference Manual. Erhältlich in der jeweils aktuellsten Version unter dev.mysql.com/doc

Oestereich B. (1998) Objektorientierte Softwareentwicklung: Analyse und Design mit der Unified Modeling Language. 4. aktualisierte Auflage. München ; Wien: Oldenburg. siehe www.oose.de/uml

Oshineye A. (März 2007) Bild von Martin Fowler auf der Flicker Website, Heruntergeladen am 31.08.2008 von www.flickr.com/photos/38234898@N00/423039025

Rerych M. (2002) Entstehungskontext. Studentischer Artikel in fit 2002 >

Softwareentwicklungsmodelle > Wasserfallmodell. Institut für Gestaltungs- und Wirkungsforschung. Technische Universität Wien. Betrachtet am 10.09.2008 unter cartoon.iguw.tuwien.ac.at/fit/fit01/wasserfall/entstehung.html. Hinweis: Dieses Bild unterliegt der Creative Commons 2.0 Attribution-Lizenz14 und darf im Rahmen dieser Lizenz frei verwendet werden.

Raymond E. S. (September 2000) The Cathedral and the Bazaar. Artikel auf Eric S. Raymond's Home Page unter www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar

Royce W. W. (August 1970) Managing the development of large software systems: concepts and techniques. International Conference on Software Engineering Proceedings of the 9th international conference on Software Engineering,1987, Monterey, California, United States. Neudruck der Pro- ceedings, IEEE WESCON, Seiten 1-9. Institute of Electrical and Electronics Engineers. Siehe www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf

SQL Industry News (Juni 2007) Gartner Says Worldwide Relational Database Market Increased 14 Percent in 2006. Nachrichtenartikel von SQL Industry News auf der SQL Manager.net-Website von EMS Database Management Solutions, Inc. Originalquelle: Gartner Dataquest (June 2007). Betrachtet am 09.09.2008 unter www.sqlmanager.net/en/news/sql/1189.

Schwartz B., Zaitsev P., Tkachenko V., Zawodny J.D., Lentz A., Balling D.J. (2008). High Performance MySQL 2nd Edition, O'REILLY.

Sparks G. (1999) Database Modelling in UML. Veröffentlicht in Methods & Tools e-newsletter. Siehe www.martinig.ch/mt/index.html und www.sparxsystems.com.au

Spicer J. (July/August 2003) Edgar (Ted) Codd, 1923-2003, Remembering the father of the relational database. Artikel auf der Oracle Hompage, betrachtet am 08.09.2008 auf www.oracle.com/technology/oramag/oracle/03-jul/o43edit.html

Standish Group International Inc. (1994) The Chaos Report Teorey T., Lightstone S., Nadeau T. (2006) Database Modelling and Design. 4th Edition, Morgan Kaufmann

26.3 Empfehlungen

Codd und Date sind sicherlich die beiden bekanntesten und einflussreichsten Forscher im Bereich der relationalen Datenbanken. Ambler und Fowler dagegen gelten als Pioniere des objekt-relationalen Mappings (ORM) und haben mit ihren Arbeiten auch die agile Softwareentwicklung stark mitgeprägt.

Das Lehrwissen dieses Skriptes stammt von diesen vier Wissenschaftlern und Praktikern und vielen weiteren, die sicherlich auch eine eigene Vorstellung verdient hätten. Aus Platzgründen kann ich aber leider nun eine kleine Auswahl treffen.

Einen Tipp kann ich Ihnen an dieser Stelle aber noch mit auf den Weg geben:

> Lesen Sie die Originale!

Viele Bücher – gerade im Datenbankenbereich – verfälschen Definitionen und Aussagen. Oftmals versuchen Autoren aus didaktischen Gründen Vereinfachungen einzuführen, die dann aber leider, manchmal schlicht falsch sind. Andere Autoren zitieren Codd oder Date und fehlinterpretieren deren Aussagen. Auch sehr geläufig ist folgendes: Ein Autor übernimmt eine Aussage eines anderen, der diese von einem anderen hat, der Codd oder Date gelesen hat. Kennen Sie das Kinderspiel *Stille Post*?

Ich kann Ihnen nur ans Herz legen: Verlassen Sie sich nicht nur auf mein Wort. Gehen Sie den Referenzen nach und lesen Sie wichtige Prinzipien nochmal im Original.

Sollten Sie in dieser Unterlage Fehler finden, bin ich für Hinweise und Korrekturvorschläge natürlich auch dankbar.

Hier nun die wichtigsten Originalquellen, die für diesen Kurs relevant sind:

- Codds ursprüngliches Paper *A Relational Model of Data for Large Shared Data Banks* (Codd, 1970). Es ist fast 40 Jahre alt und immer noch aktuell. Date und Darwen (2007) haben in der neuesten Version des Modells lediglich kleine Unstimmigkeiten bereinigt, wie sie immer wieder betonen.
- Das *Third Manifesto* (Date & Darwen, 2007). Es beschreibt die aktuellste Version[1] des relationalen Datenbankmodells. Die gleiche Beschreibung finden Sie auch in (Date, 2004) und (Date, 2005). Alle drei Bücher sind recht theoretischer Natur. Das lesbarste der drei ist *Database In Depth* (Date, 2005). Der Untertitel *Relational Theory for Practitoners* zeigt, dass es sich nicht an Forscher, sondern an praktizierende Entwickler wendet.
- Fowlers *Enterprise-Patterns* (Fowler, 2002). Fowler beschreibt in diesem Buch (unter anderem) verschiedene ORM-Strategien in Form von Patterns (Muster).
- Der *SQL-Standard*. Die SQL-Standard-Dokumente von ANSI und ISO sind prinzipiell zwar relevant – als Bettlektüre kann ich Sie Ihnen aber eher weniger empfehlen. Sie sind – wie für formale Spezifikationen leider üblich – recht schwer lesbar. Zudem richten Sie sich eher an DBMS-Entwickler und weniger an Anwendungs- und Webentwickler wie uns. Die originalen Standard-Dokumente sind auch nicht frei verfügbar. Sie sind kommerziell erhältlich unter iso.org und ansi.org. Auszüge und Drafts (Vorabversionen) können Sie aber mit etwas googlen normalerweise zu Tage fördern. Gute Anlaufstellen zu den Standards sind

beispielsweise auch www.wiscorp.com/SQLStandards.html und www.jcc.com/sql.htm.

- Das *MySQL-Handbuch* (MySQL AB, 2008). Dieses Handbuch ist ein hervorragendes Nachschlagewerk für alles, was Sie über MySQL für die Praxis wissen müssen. Dazu gehören beispielsweise die SQL-Befehle oder das Anlegen von Backups. Auf der MySQL-Website (www.mysql.com) können Sie es kostenlos herunterladen.

---

1 auch wenn Date selbst sich weigert von Versionen zu sprechen.

Lösungen

Lektion 2: Testen Sie Ihr Wissen

1. **Wozu werden Datenbanken benötigt?**

   Datenbanken werden benötigt, um Daten strukturiert zu speichern.

2. **Nennen Sie drei Arten von Webanwendungen, die Datenbanken verwenden!**

   Produktdatenbanken, Informationssysteme, Online-Spiele

3. **Geben Sie drei konkrete Beispiele für Webanwendungen, die Datenbanken verwenden!**

   z. B. www.amazon.com, www.ebay.de, www.webmasters-europe.org

4. **Nennen Sie drei Arten von Webkomponenten, die Datenbanken verwenden!**

   z. B. Forum, Blog, Wiki

5. **Unterscheiden Sie die Begriffe *Datenbank (DB)* und *Datenbank Management System (DBMS)*.**

   Die Datenbank ist die eigentliche Sammlung der Daten. Das DBMS ist das Software-Sys- tem, das die Daten verwaltet.

6. **Was ist *MySQL*?**

   MySQL ist ein relationales Datenbanksystem (DBMS).

7. **Nennen Sie mindestens 3 relationale Datenbank Management Systeme! Geben Sie jeweils an, ob es sich um Open Source oder kommerzielle Software handelt.**

   SQLite (open source), PostgresQL (open source), Oracle (kommerziell)

Lektion 3: Testen Sie Ihr Wissen

1. **Was ist SQL?**

   SQL ist eine Abfragesprache. Viele DBMS verwenden SQL, um mit einer Anwendung (d. h. einem Client) zu kommunizieren.

2. **Was ist eine Distribution?**

   Eine Distribution ist eine Bündelung von verschiedenen Software-Paketen. Eine Distribution hat meist einen gemeinsamen Installer und erleichtert somit die Einrichtung mehrerer aufeinander abgestimmter Programme.

3. **Aus welchen Teilen besteht XAMPP?**

   XAMPP besteht aus Apache, MySQL, Perl, PHP und vielen Bibliotheken.

4. **Was sind Terminale und Nichtterminale?**

   - Terminale in der BNF sind Symbole, die nicht weiter ersetzt werden können.

- Nichtterminale/Variablen in der BNF sind Symbole, die ersetzt werden müssen.

5. **Was hat MySQL mit SQL zu tun?**

   MySQL ist ein DBMS, das SQL als Abfragesprache verwendet.

Lektion 4: Testen Sie Ihr Wissen

1. **Was ist eine Entität?**

   Ein Entity im Sinne der Domänmodellierung entspricht einem Konzept aus der realen Welt. Es kann sich dabei genauso um ein materielles Objekt wie um ein abstraktes Konzept handeln. Sie können einem Entity stets Attribute zuordnen.

2. **Welches Problem löst die UML?**

   Die UML gibt einen einheitlichen Standard für Modellierungsfragen in der Softwareentwicklung vor – und setzt damit dem Wildwuchs von Diagrammtypen ein Ende. Allerdings gibt es in der Praxis neben der UML noch viele anderen Notationen, die aus verschiedenen Gründen Verwendung finden.

3. **Mit welchem Diagrammtyp begannen Datenbankentwickler die Modellierung, bevor die UML zur Verfügung stand?**

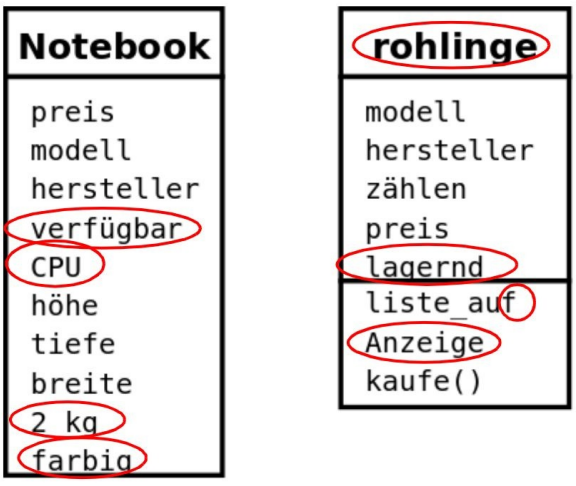
   Der klassiche Weg ist die Verwendung eines ER-Diagramms. ER-Diagramme sind auch heute noch verbreitet.

4. **Welche Hinweise können Sie verwenden, um Klassen, Attribute und Methoden des Domänenmodells aus einer Kundenanforderung zu extrahieren?**

   Ein erster Hinweis ist die Wortart. So sind Methoden grundsätzlich Verben, während Klassen und Attribute normalerweise Nomen sind.

Lektion 4: Übungen

Finden Sie die Fehler



- Statt *rohlinge* sollten Sie *Rohling* schreiben – groß und im Singular.

- Attribute werden üblicherweise klein geschrieben. Deswegen ist *cpu* besser als *CPU*.
- 2 kg ist ein Wert. Ein sinnvolles Attribut wäre *gewicht*.
- Besser als *farbig* wäre *farbe*.
- Statt *zählen*, sollten Sie *anzahl* verwenden.
- Bei *liste_auf* fehlen die Klammern »()«.
- *Anzeige* ist kein Verb – besser ist: *anzeigen* oder idealerweise der Imperativ *zeige_an*.
- Die Attribute *lagernd* und *verfügbar* sind nicht falsch. Sie könnten sie mit den logischen Werten TRUE/FALSE belegen. Besser eignet sich aber ein Attribut *lieferzustand*, das Sie mit Werten wie auf *lager*, *verfügbar*, *vorbestellt*, *nicht mehr lieferbar, usw.* belegen.

Begriffe

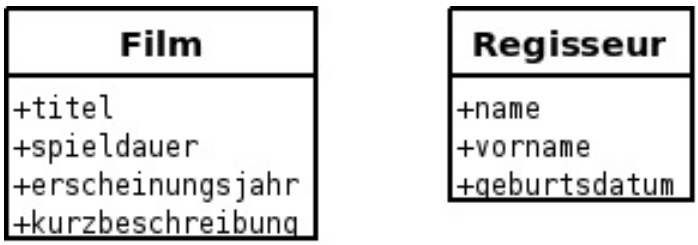
Begriff	Entity?	Begründung falls kein Entity
blau	nein	kein Nomen
Cache	nein	technischer Begriff
Seminar	ja	
JavaScript-Programmierung	nein	‘JavaScript-Programmierung’ ist ein Wert
Person	ja	
Entity	nein	zu allgemein
Unterrichten	nein	kein Nomen
Organisation	ja	
Teilnahme	ja	
Rechnungstabelle	nein	*Tabelle* ist ein Begriff aus der Implementierung, besser wäre hier *Rechnung*
Variable	nein	technischer Begriff
Veranstaltung	ja	
Dienstschnittstelle	nein	technischer Begriff
Mahnung	ja	
Cron-Job	nein	technischer Begriff

Lektion 5: Übungen

Projekt »Filmverleih«: Anforderungsanalyse
Film ist ein Entity, das sich aus den Attributen *titel*, *spieldauer*, *erscheinungsjahr* und *kurzbeschreibung* zusammensetzt. Die zweite Entity ist *Regisseur*, bestehend aus *name*,

vorname und *geburtsdatum*.

Projekt »Filmverleih«: Domänenmodell



Übrigens: Wenn Ihre Lösung nun nicht exakt mit der Musterlösung übereinstimmt, heißt das noch lange nicht, dass sie falsch ist. Gerade bei Analyse und Modellierung gibt es oft viele Möglichkeiten, die eine richtige Lösung darstellen.

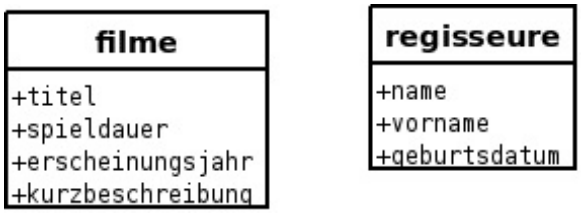
Lektion 6: Testen Sie Ihr Wissen

1. **Nennen Sie mindestens zwei Unterschiede zwischen dem Domänenmodell und dem physischen Datenmodell.**
   - Das Domänenmodell beschreibt die Konzepte einer Anwendung aus der Sicht eines Fachexperten. Das physische Datenmodell stellt dagegen eine direkte Abbildung der realen Datenbank dar.
   - Eine Klasse im Klassendiagramm eines Domänenmodells beschreibt ein Entity, während sie im phy. DB-Modell eine Tabelle darstellt.
2. **Was ist ein UML-Profil?**

   Ein UML-Profil ist eine Erweiterung zum UML-Standard. Der Standard beschreibt bereits, wie solche Profile definiert werden. Scott Ambler und andere haben UML-Profile speziell für die Datenbank-Modellierung entwickelt.
3. **Nennen Sie mindestens zwei SQL-Programmierrichtlinien.**
   - Geben Sie Tabellennamen für Tabellen, die aus Entities entstanden sind, im Plural an.
   - Schreiben Sie alle Bezeichner klein.
   - Trennen Sie Begriffe, die aus mehreren Wörtern bestehen, mit einem Underscore.
   - Vermeiden Sie Umlaute und Sonderzeichem.

Lektion 6: Übungen

Physisches Modell des Filmverleihs



Lektion 7: Testen Sie Ihr Wissen

1. **Wie werden Daten in einem relationalen DBMS organisiert?**

   Ein relationales DBMS (RDBMS) organisiert Daten in Relationen. Relationen lassen sich als Tabellen darstellen.

Lektion 7: Übungen

Fehlersuche

name	vorname	telefon_nr	telefon_nr
Meier	Michael	0911/12345	0171/12345
Schulze	Stefanie	0911/12345	0171/1234
Becker	Andreas	0911/12345	0171/1234
Meier	Michael	0911/12345	0171/12345

Folgende Fehler liegen vor:

- Das Attribut *telefon_nr* ist zweimal vorhanden.
- Der letzte Datensatz ist eine exakte Kopie des ersten.

Identische Headings

- Die Mengen 1 und 3 sind gleich, da die Reihenfolge der Attribute unerheblich ist.
- Die Mengen 6 und 7 und die Mengen 5 und 8 sind jeweils gleich.

Mengenschreibweise in Tabellen umwandeln

modell	hersteller	leistung_in_ps	von_0_auf_100_in_sec
Golf GT	VW	90	8,2
F40	Ferrari	720	4,6
E91	BMW	245	6,9

Tabelle 7.1 autos = {modell, hersteller, leistung_in_ps, von_0_auf_100_in_sec}

titel	autor	erscheinungsdatum	verlag	seitenzahl
Introduction to Database			Addison	

Systems	C.J. Date	01.09.2003	Wesley	1024
Agile Database Techniques	Scott Ambler	17.10.2003	Wiley & Sons	480
Patterns of Enterprise Application Architecture	Martin Fowler	15.11.2002	Addison Wesley	560

Tabelle 7.2 buecher = {titel, autor, erscheinungsdatum, verlag, seitenzahl}

Lektion 8: Testen Sie Ihr Wissen

1. **Nennen Sie 3 Datentypen und erklären Sie deren Einsatzzweck!**
   - INTEGER: wird zum Speichern ganzer Zahlen verwendet.
   - DATE: für Datumsangaben
   - TEXT: zum Speichern längerer Texte
2. **Welche Gruppen von Datentypen gibt es in MySQL?**

   Zeichenketten, Zahlen, Zeit- und Datumstypen und Wahrheitswerte
3. **Welchen Datentyp würden Sie für eine Telefonnummer verwenden?**

   Sie sollten normalerweise einen VARCHAR verwenden. Ein INTEGER ist völlig ungeeignet, da viele Telefonnummern mit einer

   `0`
   beginnen. Eine führende

   `0`
   wird bei einem INTEGER ignoriert, da sie keine Bedeutung für den Wert der Zahl hat.

Lektion 8: Übungen

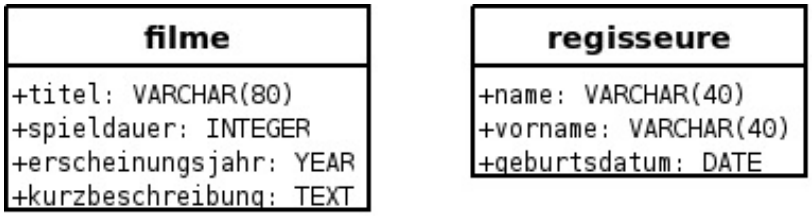
Datentypen für Werte

1. INTEGER
2. DECIMAL(3, 1)
3. DECIMAL(7, 3)
4. VARCHAR(20)
5. VARCHAR(20)

Datumsangaben

1. ‘1923-08-23’
2. ‘1975-12-02’
3. ‘2001-09-11 09:03’

Typen des Filmverleihs



Lektion 9: Testen Sie Ihr Wissen

1. **Welche Art von SQL-Anweisungen werden durch die DDL zusammengefasst?**

   SQL-Anweisungen, die die Definition der Daten betreffen.

2. **Welche durch die DML?**

   SQL-Anweisungen, mit denen Sie Daten manipulieren und auslesen können.

3. **Welche durch die DCL?**

   SQL-Anweisungen, mit denen Sie die Rechte der Datenbank verändern können.

4. **Gibt es neben DDL, DML und DCL noch weitere Arten von SQL-Anweisungen?**

   In MySQL gibt es z. B. noch die *Database Administration Statements* und die *MySQL Utility Statements*. Das kann sich aber von DBMS zu DBMS unterscheiden.

5. **Wie viele Tabellen kann eine Datenbank enthalten?**

   Manche DBMS haben eine künstliche Begrenzung, aber im Prinzip beliebig viel.

Lektion 9: Übungen

Fehler finden

```
CREATE TABLE kunden (kunden_nr INTEGER, name VARCHAR(40),
anzahl_bestellungen INTEGER, telefon_nr VARCHAR(20));
```

Korrektes SQL

1. ```
   CREATE TABLE tbl_name (create_definition,...)
   create_definition: col_name data_type
   ```

2. ```
   CREATE TABLE tbl_name (create_definition1, create_definition2, create_definition3, create_defin
   create_definition: col_name data_type
   ```

3. ```
   CREATE TABLE tbl_name (col_name1 data_type1, col_name2 data_type2, col_name3 data_type3, col_na
   ```

4. ```
   CREATE TABLE notebooks (preis DECIMAL(7, 2), modell VARCHAR(20), hersteller VARCHAR(20), liefer
   ```

BNF ausprägen

1. ICH GEHE AN DIE BAR UND MIXE MIR EINEN Vanilla Sky. DAZU

BENÖTIGE ICH Amaretto, Kahlúa, Vanillesirup UND Milch. ANSCHLIESSEND GARNIERE ICH DEN DRINK MIT einer Kirsche
2. ZU meinem Geburtstag WÜNSCHE ICH MIR ein rotes Fahrrad der Schnell & Wind AG AUS Hamburg.

Mengenschreibweise nach DDL

1. `CREATE TABLE autos (hersteller VARCHAR(40), modell VARCHAR(20), leistung_in_ps INTEGER, von_0_auf_100_in_sec DECIMAL(2, 1));`
2. `CREATE TABLE buecher (titel VARCHAR(50), autor VARCHAR(60), erscheinungsdatum DATE, klappentext TEXT, seitenzahl INTEGER));`

Tabellenschema des Filmverleihs

```
CREATE TABLE filme (titel VARCHAR(80), spieldauer INTEGER, erscheinungsjahr YEAR, filmgesellschaft V
CREATE TABLE regisseure (name VARCHAR(40), vorname VARCHAR(40), geburtsdatum DATE);
```

Lektion 11: Testen Sie Ihr Wissen

1. **Erläutern Sie die Begriffe: *eindeutig* und *irreduzibel*.**

   Anhand einer eindeutigen Attributmenge können Sie eine Zeile in einer Tabelle eindeutig identifizieren. Eine Attributmenge heißt irreduzibel, wenn Sie kein Attribut mehr herausnehmen können, ohne dass die Menge ihre Eindeutigkeit verliert.

2. **Geben Sie ein Beispiel für eine Teilmenge von *seminare = {titel, beschreibung, preis}*, die nicht eindeutig ist.**

   {preis}

3. **Geben Sie ein Beispiel für eine Teilmenge von *seminare = {titel, beschreibung, preis}*, die nicht irreduzibel ist.**

   {titel, preis}

4. **Erläutern Sie die Begriffe: *Primärschlüssel* und *Alternativschlüssel*.**

   Ein Primärschlüssel ist ein ausgewählter Schlüssel. Ein Alternativschlüssel ist ein weiterer Schlüssel, der nicht ausgewählt wurde.

5. **Was bedeutet die Unterstreichung bei folgender Menge?**
   uebungen = {<u>seminar_titel</u>, <u>kapitel_nr</u>, <u>uebungs_nr</u>, punkte}

   Die Menge {seminar_titel, kapitel_nr, uebungs_nr} ist der Primärschlüssel.

6. **Wozu wird ein Identity Field benötigt?**

   Das Muster Identity Field ermöglicht ein 1:1-Mapping zwischen Objekten und Datensätzen.

7. **Sind natürliche oder eher künstliche Schlüssel im Allgemeinen zu bevorzugen?**

   Diese Frage lässt sich so allgemein nicht beantworten. Die Meinungen gehen hier weit auseinander. Im speziellen Bereich der Webentwicklung (insbesondere in

Verbindung mit einem ORM) haben sich aber künstliche Schlüssel durchgesetzt.

8. **Kann eine Relation auch gar keine Schlüssel haben? Begründen Sie Ihre Antwort.**

   Nein. Im schlimmsten Fall werden alle Attribute zur eindeutigen Identifikation benötigt. Dann sind diese aber auch schon irreduzibel und damit ist die Menge dieser Attribute der Schlüssel der Relation.

Lektion 11: Übungen

Teilmengen bestimmen

- {model, hersteller}, {model, leistung_in_ps}, {hersteller, leistung_in_ps}, {modell}, {hersteller}, {leistung_in_ps}
- {wert}, {farbe}
- {a, b}, {a, c}, {b, c}, {a}, {b}, {c}

Schlüssel bestimmen

```
buch = {auto, erscheinungsdatum, isbn, titel}
```

Teilmengen von buch

{auto}, {erscheinungsdatum}, {isbn}, {titel},

{auto, erscheinungsdatum}, {auto, isbn}, {auto, titel}, {erscheinungsdatum, isbn}, {erscheinungsdatum, titel}, {isbn, titel},

{erscheinungsdatum, isbn, titel}, {auto, isbn, titel}, {auto, erscheinungsdatum, titel}, {autor- name, erscheinungsdatum, isbn},

{auto, erscheinungsdatum, isbn, titel}

identifizierende Teilmengen von buch

{isbn},

{auto, isbn}, {erscheinungsdatum, isbn}, {isbn, titel},

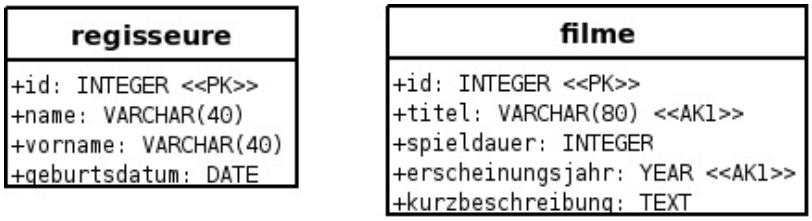
{erscheinungsdatum, isbn, titel}, {auto, isbn, titel}, {auto, erscheinungsdatum, titel}, {autor- name, erscheinungsdatum, isbn},

{auto, erscheinungsdatum, isbn, titel}

identifizierende, irreduzible Teilmengen (Schlüssel) von buch

{isbn}, {auto, erscheinungsdatum, titel}

Filmverleih mit Schlüsseln



Es lässt sich darüber streiten, ob {titel, erscheinungsjahr} als Schlüssel akzeptabel ist. {titel} alleine ist auf jeden Fall unzureichend, da es viele Filme mit gleichem Titel gibt. Für unseren Kunden, der eine kleine Videothek betreibt, schließen wir hier einfach mal aus, dass er mehrere Filme mit gleichem Titel und Erscheinungsjahr im Angebot hat.

Lektion 12: Testen Sie Ihr Wissen

1. **Wie viele Spalten in einer Tabelle können Sie als PRIMARY KEY deklarieren und wie viele als UNIQUE KEY?**

   Nur eine Spalte kann PRIMARY KEY sein, aber Sie können beliebig viele als UNIQUE KEY deklarieren.

2. **Welche Voraussetzungen muss eine Spalte erfüllen, damit Sie sie mit AUTO_INCREMENT versehen können?**

   Sie muss Schlüssel sein und einen Zahlentyp haben. Außerdem darf es keine weitere AUTO_INCREMENT-Spalte in der gleichen Tabelle geben.

Lektion 12: Übungen

BNF ausprägen

- ICH WAR GESTERN IM LOOP IN NÜRNBERG. DORT HABE ICH CHRISTIAN SCHMIDT AUS FÜRTH, EVA WEBER AUS HAMBURG UND NICOLE MÜLLER AUS MAINZ GETROFFEN.
- ICH WAR LETZTE WOCHE IN DER KULTURRUINE IN KARLSRUHE. DORT HABE ICH MICHAEL MEISTER UND BETTINA WEBER GETROFFEN.

Projekt »Filmverleih«: Schlüssel anlegen mit SQL

```
CREATE TABLE filme (id INTEGER PRIMARY KEY AUTO_INCREMENT, titel VARCHAR(80), spieldauer INTEGER,
erscheinungsjahr YEAR, kurzbeschreibung TEXT, UNIQUE KEY(titel, erscheinungsjahr));
CREATE TABLE regisseure (id INTEGER PRIMARY KEY AUTO_INCREMENT, name VARCHAR(40) UNIQUE KEY,
vorname VARCHAR(40), geburtsdatum DATE);
```

Lektion 13: Testen Sie Ihr Wissen

1. **Was unterscheidet `CHANGE` und `ALTER`?**

   ALTER ist der Grundbefehl zum Ändern in SQL. Mit CHANGE können Sie innerhalb einer ALTER-Anweisung ein Spalte umdefinieren.

2. **Kann `ALTER` einen Datenverlust bewirken?**

Ja. Allerdings versucht ALTER verlustfrei zu arbeiten, falls möglich.

Lektion 13: Übungen

Spalte hinzufügen

```
ALTER TABLE seminare ADD kategorie VARCHAR(20);
```

Korrekturen

```
ALTER TABLE RENAME buecher romane;
ALTER TABLE romane ADD id INTEGER PRIMARY KEY;
ALTER TABLE romane CHANGE titel titel VARCHAR(50);
ALTER TABLE romane DROP klappentext;
ALTER TABLE romane CHANGE seiten seitenzahl INTEGER;
```

Korrekturen am Projekt »Filmverleih«

```
ALTER TABLE filme ADD genre VARCHAR(20);
ALTER TABLE filme CHANGE titel titel VARCHAR(150);
```

Lektion 14: Übungen

SQL-Skripte für das Projekt »Filmverleih«

Sie finden die Skripte im Begleitmaterial zu diesem Buch.

Lektion 15: Übungen

Anfragen an den Filmverleih

1. `SELECT * FROM filme;`
2. `SELECT titel, erscheinungsjahr FROM filme;`
3. `SELECT DISTINCT erscheinungsjahr FROM filme;`
4. `SELECT titel, erscheinungsjahr FROM filme WHERE erscheinungsjahr < 2002;`
5. `SELECT titel, erscheinungsjahr FROM filme WHERE titel LIKE '%Ring%';`
6. `SELECT titel, erscheinungsjahr FROM filme WHERE titel LIKE '%Ring%' AND erscheinungsjahr >= 2002;`
7. `SELECT titel, kurzbeschreibung FROM filme WHERE titel LIKE '%Ring%' OR kurzbeschreibung LIKE '%Ring%';`

Passwortprüfung

```
SELECT id FROM benutzer WHERE login='mhuana' AND password='geheim'
```

Lektion 16: Übungen

Arbeiten mit Dokumentation

1. `SELECT CURTIME();`
2. `SELECT MONTH(NOW());`
3. `SELECT FLOOR(RAND() * 6) + 1;`

Ausgaben im Filmverleih: Regisseure

```
SELECT name, (YEAR(NOW()) - YEAR(geburtsdatum)) AS lebensalter FROM
regisseure;
```

Ausgaben im Filmverleih: Filme

```
SELECT titel, CONCAT(SUBSTR(kurzbeschreibung, 1, 20), '...') AS
beschreibung FROM filme;
```

Lektion 17: Testen Sie Ihr Wissen

1. **Wo innerhalb einer SQL-Anweisung können Sie Aggregationsfunktionen einsetzen, wo normale Funktionen?**

   Normale Funktionen können Sie sowohl innerhalb einer SELECT-Expression als auch in einer WHERE-Condition verwenden. Aggregationsfunktionen lassen sich dagegen nur in der SELECT-Expression einsetzen.

2. **Was ist die Besonderheit von Aggregationsfunktionen bezüglich ihrer Ein- und Ausgabe?**

   Aggregationsfunktionen verwenden als Eingabe eine Liste von Werten. Im einfachsten Fall sind das alle Werte einer Spalte. Als Ausgabe liefern sie nur einen einzigen Wert zurück.[1]

Lektion 17: Übungen

Aggregationsfunktionen im Filmverleih

1. `SELECT COUNT(*) FROM filme;`
2. `SELECT MIN(spieldauer) FROM filme WHERE erscheinungsjahr >= 2002;`
3. `SELECT MAX(LENGTH(kurzbeschreibung)) FROM filme;`
4. `SELECT MAX(YEAR(NOW())-erscheinungsjahr) FROM filme;`
5. `SELECT AVG(spieldauer) FROM filme WHERE genre="fantasy";`
6. `SELECT COUNT(DISTINCT genre) FROM filme;`

Lektion 18: Testen Sie Ihr Wissen

1. **Was passiert, wenn Sie ein UPDATE- oder DELETE-Anweisung ohne WHERE verwenden?**

   Es werden **alle** Datensätze geändert bzw. gelöscht.

Lektion 18: Übungen

Fehlerkorrektur

```
UPDATE filme SET spieldauer = 152 WHERE titel = 'The Dark Knight';
```

Löschen alter Filme

```
DELETE FROM FILME WHERE erscheinungsjahr <= 1998;
```

Lektion 19: Testen Sie Ihr Wissen

1. **Nennen Sie zwei Varianten, um die Tabelle *seminar* nach dem Attribut *preis* aufsteigend zu sortieren.**

   Sie können das ASC angeben oder weglassen, da aufsteigende Sortierung bereits das Standardverhalten ist:

   ```
   SELECT * FROM seminare ORDER BY preis;
   ```

   ```
   SELECT * FROM seminare ORDER BY preis ASC;
   ```

Lektion 19: Übungen

Filme sortiert ausgeben

1. `SELECT * FROM filme ORDER BY titel;`
2. `SELECT * FROM filme ORDER BY spieldauer DESC;`
3. `SELECT * FROM filme ORDER BY erscheinungsjahr, titel;`

Filme ausgeben mit Limit

1. `SELECT * FROM filme WHERE erscheinungsjahr > 2000 LIMIT 2;`
2. `SELECT * FROM filme WHERE erscheinungsjahr > 2000 LIMIT 2, 2;`

Lektion 20: Testen Sie Ihr Wissen

1. **Überlegen Sie sich mindestens zwei weitere Bedeutungen, für die Sie den NULL-Marker einsetzen könnten.**
   - *Nicht Anwendbar* – z.B. Ehemann einer unverheirateten Frau
   - *Information noch nicht verfügbar* – z.B. Todesdatum bei einer lebenden Person
2. **Welches UML-Mittel benötigen Sie, um Attribute in einem Klassendiagramm als NOT NULL zu kennzeichnen? Wie lautet die Kennzeichnung genau?**

   Sie kennzeichnen die Attribute mit Hilfe der OCL als **{not null}**.
3. **Welche Änderung ergibt sich bezüglich NULLs in der neusten Version des relationalen Modells?**

   Sie sind nicht länger erlaubt.
4. **Was ergibt NULL = NULL in SQL?**

NULL

Lektion 20: Übungen

NOT NULL Spalten

Sie finden die Skripte im Begleitmaterial zu diesem Buch.

Abfrage mit NULL-Werten

1. `SELECT * FROM filme WHERE spieldauer IS NULL;`
2. `SELECT * FROM filme WHERE spieldauer IS NOT NULL ORDER BY spieldauer;`

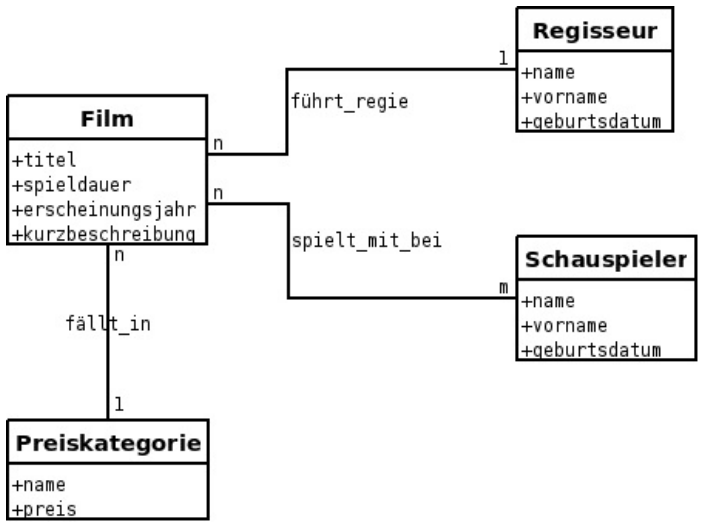
Lektion 21: Testen Sie Ihr Wissen

1. **Warum wird die Kardinalität nicht genauer bestimmt, z.B. 1:7 statt 1:n?**

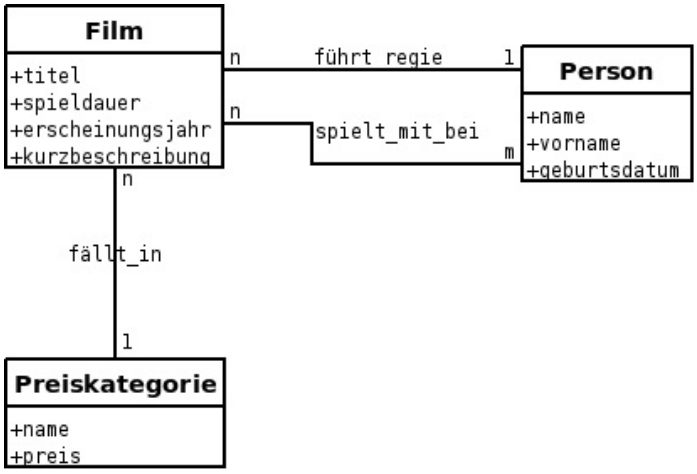
   Zum einen wäre das sehr unflexibel, da sich die genaue Kardinalität in der Praxis ständig ändert und zum anderen ist es für die Datenbankentwicklung nicht weiter wichtig.

Lektion 21: Übungen

Beziehungen im Filmverleih



Die obige Lösung lässt sich weiter optimieren. Vielleicht ist Ihnen aufgefallen, dass Schauspieler und Regisseure strukturell identisch sind. In diesem Fall ist es auch möglich, ein gemeinsame Klasse zu bestimmen (`Person`) und diese mit `Film` in Beziehung zu setzen.



Lektion 22: Testen Sie Ihr Wissen

1. **Kennt ein Datenbankschema Beziehungen?**

   Das Datenbankschema selbst kennt das Konzept der Beziehungen nicht[1]. Sie müssen sie mit Hilfe von JOIN bei einer SQL-Abfrage erst wieder herstellen. Hilfreich ist natürlich, wenn Sie sich an das Namensschema gehalten haben.

2. **Wie nennen Sie einen Fremdschlüssel, der auf eine *id* in der Tabelle *buecher* zeigt?**

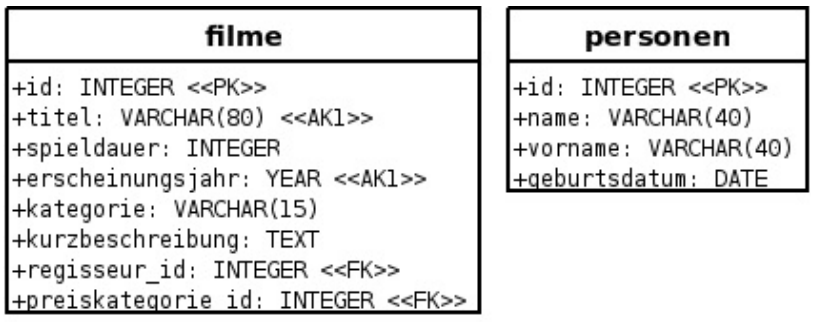
   Gemäß dem vorgestellten Namensschema heißt der Fremdschlüssel

   `buch_id`

   .

Lektion 22: Übungen

1:n-Beziehungen im Filmverleih



Das Migrationsskript finden Sie im Begleitmaterial zu diesem Buch.

Joins

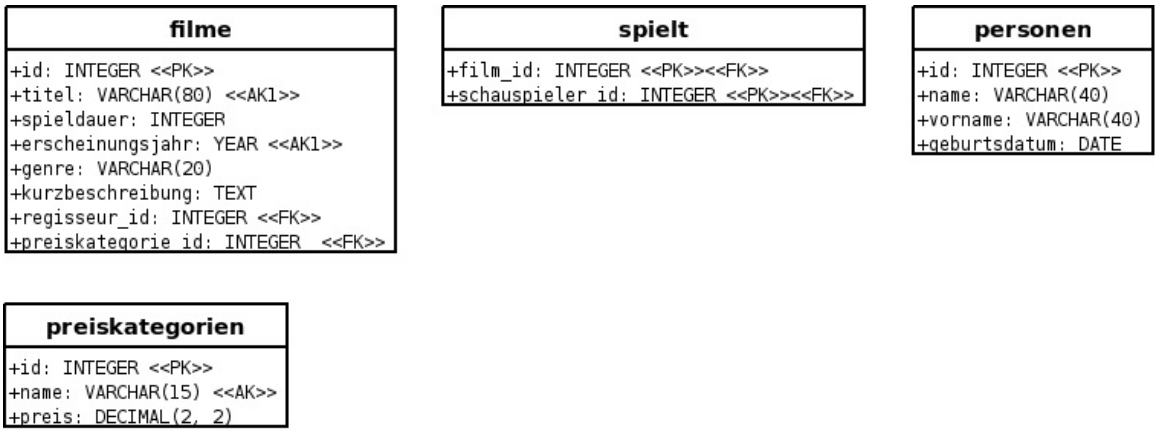
1. `SELECT filme.titel, regisseure.vorname, regisseure.name FROM filme JOIN regisseure ON filme.regisseur_id = regisseure.id;`
2. `SELECT filme.titel, regisseure.vorname, regisseure.name FROM filme JOIN regisseure ON filme.regisseur_id = regisseure.id WHERE CONCAT(regisseure.vorname, ' ', regisseure.name) = 'Peter Jackson';`

Lektion 23: Testen Sie Ihr Wissen

1. **Wie viele JOINs benötigen Sie, um eine einfache n:m-Beziehung abzufragen?**
   Zwei – jeweils einen von jeder Entity-Tabelle zur Zwischentabelle der Beziehung
2. **Wie viele JOINs sind in einer SELECT-Anweisung prinzipiell möglich?**
   beliebig viele

Lektion 23: Übungen

n:m-Beziehungen im Filmverleih



Das Migrationsskript finden Sie im Begleitmaterial zu diesem Buch.

Joins

1. `SELECT filme.titel, personen.vorname, personen.name FROM filme JOIN spielt ON spielt.film_id = filme.id JOIN personen ON spielt.schauspieler_id = personen.id;`
2. `SELECT filme.titel, personen.vorname, personen.name FROM filme JOIN spielt ON spielt.film_id = filme.id JOIN personen ON spielt.schauspieler_id = personen.id WHERE CONCAT(personen.vorname, ' ', personen.name) = 'Christian Bale';`