

Hibernate Recipes:

A Problem-Solution Approach



SRINIVAS GURUZU
GARY MAK

Apress®

Hibernate Recipes: A Problem-Solution Approach

Copyright © 2010 by Srinivas Guruzu and Gary Mak

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and the publisher.

ISBN-13 (pbk): 978-1-4302-2796-0

ISBN-13 (electronic): 978-1-4302-2797-7

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Trademarked names, logos, and images may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, logo, or image we use the names, logos, and images only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

President and Publisher: Paul Manning

Lead Editor: Steve Anglin

Technical Reviewer: Sumit Pal and Dionysios G. Synodinos

Editorial Board: Clay Andres, Steve Anglin, Mark Beckner, Ewan Buckingham, Gary Cornell, Jonathan Gennick, Jonathan Hassell, Michelle Lowman, Matthew Moodie, Duncan Parkes, Jeffrey Pepper, Frank Pohlmann, Douglas Pundick, Ben Renow-Clarke, Dominic Shakeshaft, Matt Wade, Tom Welsh

Coordinating Editor: Anita Castro

Copy Editor: Tiffany Taylor

Compositor: Kimberly Burton

Indexer: BIM Indexing & Proofreading Services

Artist: April Milne

Cover Designer: Anna Ishchenko

Distributed to the book trade worldwide by Springer Science+Business Media, LLC., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com.

For information on translations, please e-mail rights@apress.com, or visit www.apress.com.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales—eBook Licensing web page at www.apress.com/info/bulksales.

The information in this book is distributed on an “as is” basis, without warranty. Although every precaution has been taken in the preparation of this work, neither the author(s) nor Apress shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in this work.

The source code for this book is available to readers at www.apress.com. You will need to answer questions pertaining to this book in order to successfully download the code.

To my lovely wife, Usha

— *Srinivas Guruzu*

Contents at a Glance

■ About the Authors.....	xix
■ About the Technical Reviewer	xx
■ Acknowledgements	xxi
■ Chapter 1: Starting with Hibernate.....	1
■ Chapter 2: Basic Mapping and Object Identity	33
■ Chapter 3: Component Mapping	49
■ Chapter 4: Inheritance and Custom Mapping	69
■ Chapter 5: Many-to-One and One-to-One Mapping	95
■ Chapter 6: Collection Mapping	115
■ Chapter 7: Many-Valued Associations.....	137
■ Chapter 8: HQL and JPA Query Language.....	155
■ Chapter 9: Querying with Criteria and Example	167
■ Chapter 10: Working with Objects.....	179
■ Chapter 11: Batch Processing and Native SQL.....	193
■ Chapter 12: Caching in Hibernate.....	203
■ Chapter 13: Transactions and Concurrency	219
■ Chapter 14: Web Applications	237
■ Index.....	265

Contents

■ About the Authors	xix
■ About the Technical Reviewer	xx
■ Acknowledgements	xxi
■ Chapter 1: Starting with Hibernate	1
1.1 Setting Up Hibernate	3
Problem	3
Solution	3
How It Works	4
Installing the JDK	4
Installing the Eclipse Web Tools Platform (WTP)	5
Installing Derby	5
1.2 Programming with Basic JDBC	7
Problem	7
Solution	7
How It Works	7
Creating an Eclipse Project	7
JDBC Initialization and Cleanup	8
Using JDBC to Query a Database	8
Using JDBC to Update a Database	8
Creating the Domain Model.....	9
Retrieving Object Graphs.....	10
Persisting Object Graphs.....	11
Problems with Using JDBC.....	12
1.3 Configuring Hibernate	12

Problem	12
Solution	12
How It Works	12
Getting the Required Jars	13
Creating Mapping Definitions	13
Configuration	14
Programmatic Configuration	14
XML Configuration	15
Opening and Closing Sessions	16
Retrieving Objects	16
1.4 Configuring a JPA Project	17
Problem	17
Solution	17
How It Works	17
Opening a Session	22
1.5 Using Entity Manager	23
Problem	23
Solution	23
How It Works	23
1.6 Enabling Logging in Hibernate	27
Problem	27
Solution	27
How It Works	28
Inspecting the SQL Statements Issued by Hibernate	28
Configuring Log4j	28
Enabling Live Statistics	28
1.7 Generating a Database Schema Using Hibernate	29
Problem	29
Solution	29
How It Works	29

Creating an Ant Build File	29
Generating Database Schema Using SchemaExport	30
Updating a Database Schema Using SchemaUpdate	30
Specifying the Details of a Database Schema	30
Summary	31
■ Chapter 2: Basic Mapping and Object Identity	33
2.1 Providing an ID for Persistence	33
Problem	33
Solution	33
How It Works	33
2.2 Creating a Composite Key in Hibernate	38
Problem	38
Solution	39
How It Works	39
2.3 SaveOrUpdate in Hibernate	42
Problem	42
Solution	42
How It Works	43
2.4 Dynamic SQL Generation in Hibernate	43
Problem	43
Solution	44
How It Works	44
2.5 Naming Entities in Hibernate	45
Problem	45
Solution	45
How It Works	46
Summary	48
■ Chapter 3: Component Mapping	49
3.1 Implementing a Value Type as a Component	49
Problem	49

Solution	49
How It Works	49
Using JPA Annotations	52
3.2 Nesting Components.....	55
Problem	55
Solution	55
How It Works	56
3.3 Adding References in Components.....	58
Problem	58
Solution	58
How It Works	58
3.4 Mapping a Collection of Components	61
Problem	61
Solution	61
How It Works	61
3.5 Using Components as Keys to a Map.....	66
Problem	66
Solution	66
How It Works	66
Summary	67
■ Chapter 4: Inheritance and Custom Mapping	69
4.1 Mapping Entities with Table per Class Hierarchy	70
Problem	70
Solution	70
How It Works	72
4.2 Mapping Entities with Table per Subclass.....	74
Problem	74
Solution	75

How It Works	75
4.3 Mapping Entities with Table per Concrete Class	78
Problem	78
Solution	78
How It Works	78
4.4 Custom Mappings	81
Problem	81
Solution	81
How It Works	81
4.5 CompositeUserType Mappings	87
Problem	87
Solution	87
How It Works	87
Summary	93
■ Chapter 5: Many-to-One and One-to-One Mapping	95
5.1 Using Many-To-One Associations	95
Problem	95
Solution	96
How It Works	96
5.2 Using a Many-to-One Association with a Join Table	99
Problem	99
Solution	100
How It Works	100
5.3 Using Lazy Initialization on Many-to-One Associations	102
Problem	102
Solution	102
How It Works	103
5.4 Sharing Primary Key Associations	104

Problem	104
Solution	104
How It Works	104
5.5 Creating a One-to-One Association Using a Foreign Key	107
Problem	107
Solution	107
How It Works	107
5.6 Creating a One-to-One Association Using a Join Table	109
Problem	109
Solution	109
How It Works	109
Summary	113
■ Chapter 6: Collection Mapping	115
6.1 Mapping a Set.....	115
Problem	115
Solution	115
How It Works	116
6.2 Mapping a Bag.....	118
Problem	118
Solution	118
How It Works	118
6.3 Mapping a List	122
Problem	122
Solution	122
How It Works	122
6.4 Mapping an Array	124
Problem	124
Solution	124

How It Works	124
6.5 Mapping a Map	126
Problem	126
Solution	126
How It Works	126
6.6 Sorting Collections.....	128
Problem	128
Solution	128
How It Works	129
Using the Natural Order.....	129
Writing Your Own Comparator.....	130
Sorting in the Database.....	132
6.6 Using Lazy Initialization	133
Problem	133
Solution	133
How It Works	134
Summary	135
■ Chapter 7: Many-Valued Associations.....	137
7.1 Mapping a One-to-Many Association with a Foreign Key	137
Problem	137
Solution	137
How It Works	138
7.2 Mapping a One-to-Many Bidirectional Association Using a Foreign Key.....	142
Problem	142
Solution	142
How It Works	143
7.3 Mapping a One-to-Many Bidirectional Association Using a Join Table	145
Problem	145

Solution	145
How It Works	145
7.4 Mapping a Many-to-Many Unidirectional Association with a Join Table	148
Problem	148
Solution	149
How It Works	149
7.5 Creating a Many-to-Many Bidirectional Association with a Join Table	150
Problem	150
Solution	150
How It Works	151
Summary	153
■ Chapter 8: HQL and JPA Query Language	155
8.1 Using the Query Object	155
Problem	155
Solution	155
How It Works	156
Creating a Query Object	156
The from Clause	156
The where Clause.....	157
Pagination	157
Parameter Binding	158
Named Queries.....	160
8.2 Using the Select Clause	161
Problem	161
Solution	161
How It Works	161
8.3 Joining	163
Problem	163
Solution	163

How It Works	163
Explicit Joins	163
Implicit Joins	164
Outer Joins	164
Matching Text.....	164
Fetching Associations	165
8.4 Creating Report Queries.....	165
Problem	165
Solution	165
How It Works	165
Projection with Aggregation Functions	165
Grouping Aggregated Results.....	166
Summary	166
Chapter 9: Querying with Criteria and Example	167
9.1 Using Criteria	168
Problem	168
Solution	168
How It Works	168
9.2 Using Restrictions.....	169
Problem	169
Solution	169
How It Works	169
Writing Subqueries.....	171
9.3 Using Criteria in Associations	172
Problem	172
Solution	172
How It Works	172
9.4 Using Projections	174
Problem	174

Solution	174
How It Works	174
Aggregate Functions and Groupings with Projections	175
9.5 Querying by Example	176
Problem	176
Solution	176
How It Works	176
Summary	177
■ Chapter 10: Working with Objects.....	179
10.1 Identifying Persistent Object States.....	179
Problem	179
Solution	179
How It Works	179
Transient Objects	179
Persistent Objects	180
Detached Objects	180
Removed Objects	181
10.2 Working with Persistent Objects.....	182
Problem	182
Solution	182
How It Works	182
Creating a Persistent Object.....	182
Retrieving a Persistent Object.....	184
Modifying a Persistent Object	185
Deleting a Persistent Object.....	185
10.3 Persisting Detached Objects	186
Problem	186
Solution	186
How It Works	186

Reattaching a Detached Object.....	186
Merging a Detached Object.....	186
10.4 Using Data Filters.....	187
Problem	187
Solution	187
How It Works	188
10.5 Using Interceptors.....	190
Problem	190
Solution	190
How It Works	190
Summary	192
Chapter 11: Batch Processing and Native SQL	193
11.1 Performing Batch Inserts.....	194
Problem	194
Solution	194
How It Works	194
11.2 Performing Batch Updates and Deletes	195
Problem	195
Solution	195
How It Works	195
11.3 Using Native SQL	197
Problem	197
Solution	197
How It Works	198
11.4 Using Named SQL Queries.....	199
Problem	199
Solution	199
How It Works	199

Summary	201
■ Chapter 12: Caching in Hibernate	203
Using the Second-Level Cache in Hibernate	204
Concurrency Strategies	205
Cache Providers	205
What Are Cache Regions?	207
Caching Query Results	207
12.1 Using the First-Level Cache	207
Problem	207
Solution	208
How It Works	208
12.2 Configuring the Second-Level Cache	209
Problem	209
Solution	209
How It Works	209
12.3 Caching Associations	212
Problem	212
Solution	212
How It Works	212
12.4 Caching Collections	213
Problem	213
Solution	213
How It Works	213
12.5 Caching Queries	215
Problem	215
Solution	215
How It Works	215
Summary	217

Chapter 13: Transactions and Concurrency	219
13.1 Using Programmatic Transactions in a Standalone Java Application.....	220
Problem	220
Solution	220
How It Works	221
13.2 Using Programmatic Transactions with JTA.....	223
Problem	223
Solution	223
How It Works	224
13.3 Enabling Optimistic Concurrency Control	228
Problem	228
Solution	228
How It Works	231
13.4 Using Pessimistic Concurrency Control	234
Problem	234
Solution	234
How It Works	234
Summary	236
Chapter 14: Web Applications	237
14.1 Creating a Controller for the Bookshop Web Application.....	238
Problem	238
Solution	238
How It Works	238
Creating a Dynamic Web Project.....	238
Configuring the Connection Pool	241
Developing an Online Bookshop.....	242
Creating a Global Session Factory	242
Listing Persistent Objects.....	242
Updating Persistent Objects.....	244

Creating Persistent Objects	249
Deleting Persistent Objects	252
14.2 Creating a Data-Access Layer	254
Problem	254
Solution	254
How It Works	255
Organizing Data Access in Data-Access Objects	255
Using Generic Data-Access Objects	257
Using a Factory to Centralize DAO Retrieval	259
Navigating Lazy Associations	261
Using the Open Session in View Pattern	262
Summary	264
■ Index.....	265

About the Authors



■ **Srinivas Guruzu** is a Developer who has been coding for more than 8 years. After completing his MS in Mechanical Engineering, Srinivas worked on high traffic payment systems in the banking domain. He also has experience working in the insurance domain. Lately, he's been working with Spring and Hibernate building applications that integrate with other products as a part of large customer enrollment and file transmission system.



■ **Gary Mak**, founder and chief consultant of Meta-Archit Software Technology Limited, has been a technical architect and application developer on the enterprise Java platform for over seven years. He is the author of the Apress books *Spring Recipes: A Problem-Solution Approach* and *Pro SpringSource dm Server*. In his career, Gary has developed a number of Java-based software projects, most of which are application frameworks, system infrastructures, and software tools. He enjoys designing and implementing the complex parts of software projects. Gary has a master's degree in computer science. His research interests include object-oriented technology, aspect-oriented technology, design patterns, software reuse, and domain-driven development.

Gary specializes in building enterprise applications on technologies including Spring, Hibernate, JPA, JSF, Portlet, AJAX, and OSGi. He has been using the Spring Framework in his projects for five years, since Spring version 1.0. Gary has been an instructor of courses on enterprise Java, Spring, Hibernate, Web Services, and agile development. He has written a series of Spring and Hibernate tutorials as course materials, parts of which are open to the public, and they're gaining popularity in the Java community. In his spare time, he enjoys playing tennis and watching tennis competitions.

About the Technical Reviewer



■ Sumit Pal has about 16 years of experience with Software Architecture, Design & Development on a variety of platforms including Java, J2EE. Sumit has worked in SQLServer Replication group, while with Microsoft for 2 years & with Oracle's OLAP Server group, while with Oracle for 7 years.

Apart from Certifications like IEEE-CSDP and J2EE Architect, Sumit also has an MS in Computer Science from the Asian Institute of Technology, Thailand.

Sumit has keen interest in Database Internals, Algorithms and Search Engine Technology Data Mining and Machine Learning. He has invented both basic generalized algorithms to find divisibility between numbers, as well as divisibility rules for prime numbers less than 100.

Sumit loves badminton and swimming, is an amateur astrophysicist, and is inculcating green habits into his daily life.

Sumit works as the Architect at Leapfrogrx.com

Acknowledgments

Would I do it again? Writing a book, I mean. Of course, Yes! However, anyone who believes a book project is a simple effort does not have any idea what goes into it. It takes a lot of commitment, focus, and support from family and friends.

Josh Long, thank you, thank you, thank you very much for introducing me to Apress; for believing in me and guiding me through everything that a new author like me needed to know. Gary Mak, thank you, for trusting me and letting me be a co-author for this book. Thank you, Apress and Steve Anglin for giving me the opportunity to write this book and trusting Josh Long's instincts. Tom Welsh, Sumit Pal, and Dionysios G. Synodinos, thank you, for reviewing and providing excellent suggestions. Being a first time author, the suggestions provided were of immense help in developing the book. Anita Castro answered the book and non-book related questions very patiently. Tiffany Taylor helped with grammar, spelling and consistency throughout the book. I am very grateful for support and help provided by the Apress team.

To my lovely wife, Usha, without whom I would not have been able to complete this book.

Srinivas Guruzu
Scottsdale, AZ



Starting with Hibernate

An *object model* uses the principles of abstraction, encapsulation, modularity, hierarchy, typing, concurrency, polymorphism, and persistence. The object model enables you to create well-structured, complex systems. In an object model system, *objects* are the components of the system. Objects are instances of classes, and classes are related to other classes via inheritance relationships. An object has an identity, a state, and behavior. An object model helps you create reusable application frameworks and systems that can evolve over time. In addition, object-oriented systems are usually smaller than non-object-oriented implementations.

A *relational model* defines the structure of data, data manipulation, and data integrity. Data is organized in the form of tables, and different tables are associated by means of referential integrity (a foreign key). Integrity constraints such as a primary key, unique check constraints, and not null are used to maintain an entity's integrity in the relational model.

A relational data model isn't focused on supporting entity-type inheritance: entity-based polymorphic association from an object model can't be translated into similar entities in a relational model. In an object model, you use the state of the model to define equality between objects. But in a relational model, you use an entity's primary key to define equality of entities. Object references are used to associate different objects in an object model, whereas a foreign key is used to establish associations in a relational model. Object references in the object model facilitate easier navigation through the object graph.

Because these two models are distinctly different, you need a way to persist object entities (Java objects) into a relational database. Figures 1-1 and 1-2 provide a simple representation of the object model and the relational model.

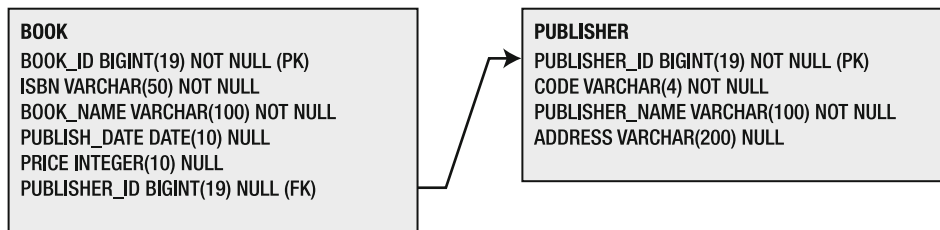


Figure 1-1. Entity-relationship (ER) diagram of Book and Publisher

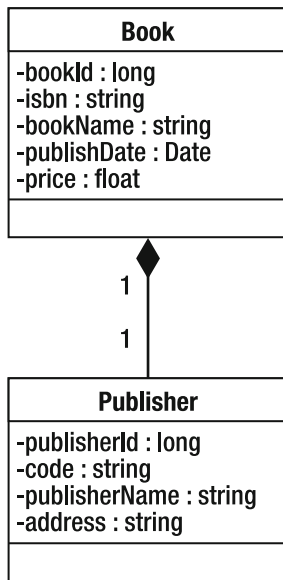


Figure 1-2. Class diagram of *Book* and *Publisher*

Object/relational mapping (ORM) frameworks help you take advantage of the features present in the object model (like Java) and the relational model (like database management systems [DBMS]). With the help of ORM frameworks, you can persist objects in Java to relational tables using metadata that describes the mapping between the objects and the database. The metadata shields the complexity of dealing directly with SQL and helps you develop solutions in terms of business objects.

An ORM solution can be implemented at various levels:

- *Pure relational:* An application is designed around the relational model.
- *Light object mapping:* Entities are represented as classes and are mapped manually to relational tables.
- *Medium object mapping:* An application is designed using an object model, and SQL is generated during build time using code-generation utilities.
- *Full object mapping:* This mapping supports sophisticated object modeling including composition, inheritance, polymorphism, and persistence by reachability.

The following are the benefits of using an ORM framework:

- *Productivity:* Because you use metadata to persist and query data, development time decreases and productivity increases.
- *Prototyping:* Using an ORM framework is extremely useful for quick prototyping.
- *Maintainability:* Because much of the work is done through configuration, your code has fewer lines and thus requires less maintenance.

- *Vendor independence:* An ORM abstracts an application from the underlying SQL database and SQL dialect. This gives you the portability to support multiple databases.

ORM frameworks also have some disadvantages:

- *Learning curve:* You may experience a steep learning curve as you learn how to map and, possibly, learn a new query language.
- *Overhead:* For simple applications that use a single database and data without many business requirements for complex querying, an ORM framework can be extra overhead.
- *Slower performance:* For large batch updates, performance is slower.

Hibernate is one of the most widely used ORM frameworks in the industry. It provides all the benefits of an ORM solution and implements the Java Persistence API (JPA) defined in the Enterprise JavaBeans (EJB) 3.0 specification.

Its main components are as follows:

- *Hibernate Core:* The Core generates SQL and relieves you from manually handling Java Database Connectivity (JDBC) result sets and object conversions. Metadata is defined in simple XML files. The Core offers various options for writing queries: plain SQL; Hibernate Query Language (HQL), which is specific to Hibernate; programmatic criteria, or Query by Example (QBE). It can optimize object loading with various fetching and caching options.
- *Hibernate Annotations:* With the introduction of Annotations in JDK 5.0, Hibernate provides the option of defining metadata using annotations. This reduces configuration using XML files and makes it simple to define required metadata directly in the Java source code.
- *Hibernate EntityManager:* The JPA specification defines programming interfaces, lifecycle rules for persistent objects, and query features. The Hibernate implementation for this part of the JPA is available as Hibernate EntityManager.

This book provides solutions using Hibernate Core and Annotations for each problem. The Hibernate version used is 3.3.2.

1.1 Setting Up Hibernate

Problem

What tools and libraries are required to set up Hibernate and get started?

Solution

You need JDK 1.5+, an IDE such as Eclipse, a database (this book uses Apache Derby), and SQL Squirrel to provide a GUI to use the database. You can also use Maven to configure your project. Maven is a software project-management and comprehension tool. Based on the concept of a project object model

(POM), Maven can manage a project's build, reporting, and documentation from a central piece of information. In Maven, the POM.XML is the central piece where all the information is stored.

The following libraries are required for the Hibernate 3.3.2 setup:

- `Hibernate3.jar`
- `Hibernate-commons-annotations.jar`
- `Hibernate-annotations.jar`
- `Hibernate-entitymanager.jar`
- `Antlr-2.7.6.jar`
- `Commons-collections-3.1.jar`
- `Dom4j-1.6.1.jar`
- `Javassist-3.9.0.GA.jar`
- `Jta-1.1.jar`
- `Slf4j-api-1.5.8.jar`
- `Ejb3-persistence.jar`
- `Slf4j-simple1.5.8.jar`

The following are required for the Derby setup :

- `Derby.jar`
- `Derbyclient.jar`
- `Derbynet.jar`
- `Derbytools.`

How It Works

The next few sections describe how to set up each of the required tools and then provide the solution to the problem. All the solutions are provided on a Windows platform. They can also be implemented on UNIX, provided you install and download the libraries and executables specific to the UNIX platform wherever applicable.

Installing the JDK

The JDK is an essential toolkit provided for Java application development. You can go to <http://java.sun.com/j2se/1.5.0/download.jsp> to download JDK 5.0. Install it into a folder such as `C:\jdk1.5.0`.

Installing the Eclipse Web Tools Platform (WTP)

Eclipse is an IDE for developing Java applications. The latest version is Galileo. You can install it from the following URL:

www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/galileo/SR1/eclipse-jee-galileo-SR1-win32.zip.

Installing Derby

Derby is an open source SQL relational database engine written in Java. You can go to http://db.apache.org/derby/derby_downloads.html and download the latest version. Derby also provides plug-ins for Eclipse. The plug-in gives you the required jar files for development and also provides a command prompt (ij) in Eclipse to execute Data Definition Language (DDL) and Data Manipulation Language (DML) statements.

Creating a Derby Database Instance

To create a new Derby database called BookShopDB at the ij prompt, use the following command:

```
connect 'jdbc:derby://localhost:1527/BookShopDB;create=true;
user=book;password=book';
```

After the database is created, execute the SQL scripts in the next section to create the tables.

Creating the Tables (Relational Model)

These solutions use the example of a bookshop. Books are published by a publisher, and the contents of a book are defined by the chapters. The entities **Book**, **Publisher**, and **Chapter** are stored in the database; you can perform various operations such as reading, updating, and deleting.

Because an ORM is a mapping between an object model and a relational model, you first create the relational model by executing the DDL statements to create the tables/entities in the database. You later see the object model in Java and finally the mapping between the relational and the object models.

Create the tables for the online bookshop using the following SQL statements:

```
CREATE TABLE PUBLISHER (
    CODE VARCHAR(4) NOT NULL ,
    PUBLISHER_NAME VARCHAR(100) NOT NULL,          ADDRESS VARCHAR(200),  PRIMARY KEY
(CODE)
);
```

```
CREATE TABLE BOOK
(ISBN VARCHAR(50) NOT NULL,
BOOK_NAME VARCHAR(100) NOT NULL,
PUBLISHER_CODE VARCHAR(4),    PUBLISH_DATE DATE,
PRICE integer,
PRIMARY KEY (ISBN),    FOREIGN KEY (PUBLISHER_CODE)
REFERENCES PUBLISHER (CODE)
);
```

```

CREATE TABLE CHAPTER
(BOOK_ISBN VARCHAR(50) NOT NULL,
  IDX integer NOT NULL,
  TITLE VARCHAR(100) NOT NULL,
  NUM_OF_PAGES integer,
  PRIMARY KEY (BOOK_ISBN, IDX),
  FOREIGN KEY (BOOK_ISBN)
  REFERENCES BOOK (ISBN)
);

```

Figure 1-3 shows the entity model for the sample table structure.

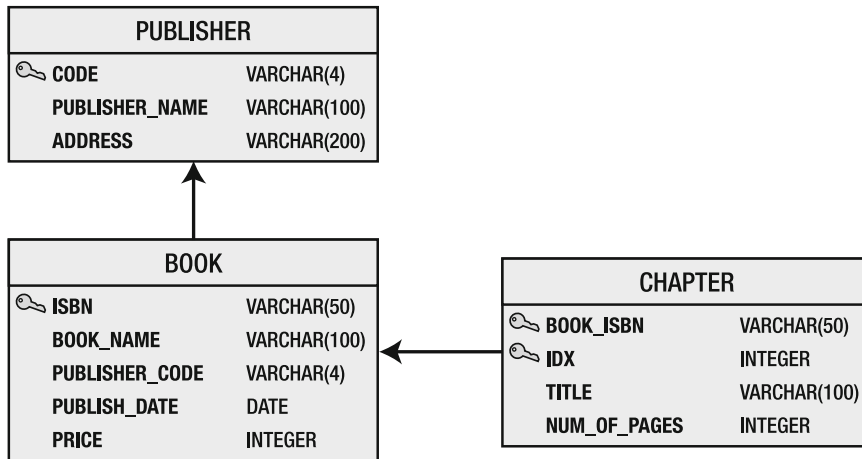


Figure 1-3. Relational model diagram for the bookshop

Next, let's input some data for these tables using the following SQL statements:

```

insert into PUBLISHER(code, publisher_name, address)
values ('001', 'Apress', 'New York ,New York');
insert into PUBLISHER(code, publisher_name, address)
values ('002', 'Manning', 'San Francisco', 'CA')
insert into book(isbn, book_name, publisher_code, publish_date, price)
values ('PBN123', 'Spring Recipes', '001', DATE('2008-02-02'), 30)
insert into book(isbn, book_name, publisher_code, publish_date, price)
values ('PBN456', 'Hibernate Recipes', '002', DATE('2008-11-02'), 40)

```

1.2 Programming with Basic JDBC

Problem

The traditional way to access a relational database is to use Java Database Connectivity (JDBC). Some common problems with using JDBC directly are as follows:

- You must manually handle database connections. There is always the risk that connections aren't closed, which can lead to other problems.
- You have to write a lot of bulky code, because all the fields required for inserts, updates, and queries must be explicitly mentioned.
- You have to manually handle associations. For complex data, this can be a major issue.
- The code isn't portable to other databases.

Solution

This section shows how you perform basic Create, Read, Update, and Delete (CRUD) operations using JDBC and describes the problems with using basic JDBC. You see how the object model is translated into the relational data model.

How It Works

To Start, you will need to create an eclipse project. You will need to install Derby jars and configure

Creating an Eclipse Project

To begin developing your Java application, you create a bookshop project in Eclipse. To set up the Derby database, you can install the core and UI plug-ins or download the Derby jar files and add them to your Eclipse project classpath.

To install the Derby plug-ins for Eclipse, do the following:

1. Download the plug-ins from <http://db.apache.org/derby/releases/release-10.5.3.0.cgi>.
2. Extract the zip files to your Eclipse home. If Eclipse is located at `C:\eclipse`, extract the zips to the same location.
3. Restart Eclipse. You should see the Derby jar files—`derby.jar`, `derbyclient.jar`, `derbynet.jar`, and `derbytools.jar`—added to your project's classpath.
4. Select your project, and right-click. Select Apache Derby, and then select Add Network Server.
5. Click the “Start Derby Network Server” button.

6. After the server starts, select the `ij` option. The SQL prompt appears in the console window.
7. At the prompt, execute the SQL statements from the previous recipe's "Creating the Tables (Relational Model)" section.

JDBC Initialization and Cleanup

You must load the JDBC driver and create a connection to that database before you can execute any SQL statements. The JDBC driver for Derby is in the `derby.jar` file that was added to your project's build path during Derby's installation. Be careful: you must remember to close that connection (whether an exception is raised or not). Connections are a costly resource—if they aren't closed after use, the application will run out of them and stop working:

```
Class.forName("org.apache.derby.jdbc.EmbeddedDriver");
Connection connection = DriverManager.getConnection(
    "jdbc:derby://localhost:1527/BookShopDB", "book", "book");
try {
    // Using the connection to query or update database
} finally {
    connection.close();
}
```

Using JDBC to Query a Database

For demonstration purpose, let's query for a book whose ISBN is 1932394419. Here's the JDBC code for this task:

```
PreparedStatement stmt = connection.prepareStatement
    ("SELECT * FROM BOOK WHERE ISBN = ?");
stmt.setString(1, "1932394419");
ResultSet rs = stmt.executeQuery();
while (rs.next())
{
    System.out.println("ISBN : " + rs.getString("ISBN"));
    System.out.println("Book Name : " + rs.getString("BOOK_NAME"));
    System.out.println("Publisher Code : " +
rs.getString("PUBLISHER_CODE"));
    System.out.println("Publish Date : " + rs.getDate("PUBLISH_DATE"));
    System.out.println("Price : " + rs.getInt("PRICE"));
    System.out.println();
}
rs.close();
stmt.close();
```

Using JDBC to Update a Database

Let's update the title of the book whose ISBN is 1932394419. Here's the JDBC code:

```

PreparedStatement stmt = connection.prepareStatement(
    "UPDATE BOOK SET BOOK NAME = ? WHERE ISBN = ?");
stmt.setString(1, "Hibernate Quickly 2nd Edition");
stmt.setString(2, "1932394419");
int count = stmt.executeUpdate();
System.out.println("Updated count : " + count);
stmt.close();

```

Creating the Domain Model

You use normal JavaBeans to build your object/domain model. These JavaBeans are called Plain Old Java Objects (POJOs). This term is used to distinguish them from Enterprise JavaBeans (EJBs). EJBs are Java objects that implement one of the `javax.ejb` interfaces and that need to be deployed in an EJB container. Note that each of these POJOs must have a no-argument constructor:

```

public class Publisher {
    private String code;
    private String name;
    private String address;
    // Getters and Setters
}
public class Book {
    private String isbn;
    private String name;
    private Publisher publisher;
    private Date publishDate;
    private int price;
    private List chapters;
    // Getters and Setters
}
public class Chapter {
    private int index;
    private String title;
    private int numOfPages;
    // Getters and Setters
}

```

You use a foreign key to reference PUBLISHER from the BOOK table, but it's a many-to-one association represented with a list of chapters in the Book class. You use a foreign key to reference BOOK from the CHAPTER table, but there's nothing referencing the Book class from the Chapter class. In contrast, a Book object has a list of Chapter objects (one-to-many association). This is a case of an *object-relational mismatch*, which focuses on the association between two classes or tables in their corresponding model. To handle the incompatibility of these models, you need to do some conversion/translation when you retrieve and save your object model. This is called object/relational mapping (O/R Mapping or ORM).

Let's say your Bookshop sells audio and video discs. In the object-oriented model, this can be represented by using a Disc superclass and two subclasses called AudioDisc and VideoDisc (see Figure 1-4). On the relational database side, you don't have a way to map this inheritance relationship. This is a major object-relational system mismatch.