



Jason Ouellette

Third Edition

Development with the Force.com Platform

Building Business Applications in the Cloud

Developer's Library



FREE SAMPLE CHAPTER

SHARE WITH OTHERS



Praise for *Development with the Force.com Platform*, Third Edition

“The third edition of *Development with the Force.com Platform* is a must-read for anyone building enterprise applications in the cloud. Whether you’re a CEO or a code ninja, Jason’s insight into the Force.com platform is priceless. Why waste time learning from your own mistakes when you can learn from a master.”

—**Howard Brown**, CEO and Founder, RingDNA

“I absolutely love this book. Jason has organized and written it in a simplified manner which makes the concepts easy to grasp for all audiences. I recommend it for any developer, consultant, or manager new to or currently working with the Force.com platform.”

—**Stephanie Buchenberger**, Salesforce.com Delivery Manager, Appirio

“Solid evolution of an already well-written book! The layout, format and content make it a great tutorial for developers new to Apex as well as an informative and thorough reference for the most experienced architect. Very up to date to the platform with practical examples that will undoubtedly be used again and again.”

—**Tom Hedgecoth**, Vice President, Global Consulting – sakonent

“This is still the best, most comprehensive book on the Force.com platform written. If you are new to Force.com, then this is the place to start. If you’re an experienced developer, then this is the book you’ll return to, over and over again. It’s an essential companion for all Force.com developers.”

—**Kevin Ott**, Senior Director, Engineering, Cisco Systems

“Jason touches on all the core elements of Force.com with a balanced blend of configuration and code. If you’re new to the platform, this book will save you countless hours as you come up to speed—and if you’re a seasoned expert you probably already own it. In either case, consider it required reading.”

—**Adam Purkiss**, Principal Architect, MondayCall Solutions, and Organizer of the Bay Area Salesforce Developer User Group

“As a Salesforce system administrator and business analyst making the transition to Force.com developer, this book helps me daily. It’s at the perfect level to cut through the vast amount of information available for developing on Force.com on the one hand, and get to the details needed to make my programs work on the other. I keep this book open perpetually, and it’s the first place I go when I get stuck. The sample coding is strong and very reusable; it’s the #1 tool in my box. I’d highly recommend *Development with the Force.com Platform* to anyone making the transition from Salesforce system administrator or business analyst to developer.”

—**Gene Teglovic**, PSA Consultant, Financialforce.com

This page intentionally left blank

Development with the Force.com Platform

Building Business Applications in the Cloud

Third Edition

Jason Ouellette

◆ Addison-Wesley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Cape Town • Sydney • Tokyo • Singapore • Mexico City

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The author and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

The publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact:

U.S. Corporate and Government Sales
(800) 382-3419
corpsales@pearsontechgroup.com

For sales outside the United States, please contact:

International Sales
international@pearsoned.com

Library of Congress Control Number: 2013950238

Visit us on the Web: informit.com/aw

Copyright © 2014 Pearson Education, Inc.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. To obtain permission to use material from this work, please submit a written request to Pearson Education, Inc., Permissions Department, One Lake Street, Upper Saddle River, New Jersey 07458, or you may fax your request to (201) 236-3290.

Screenshots © 2014 Salesforce.com, Inc. All rights reserved.

ISBN-13: 978-0-321-94916-5

ISBN-10: 0-321-94916-1

Text printed in the United States on recycled paper at RR Donnelley in Crawfordsville, Indiana.

First printing: December 2013

Editor-in-Chief

Mark Taub

Executive Editor

Laura Lewin

Development Editor

Songlin Qiu

Managing Editor

Kristy Hart

Project Editor

Andy Beaster

Copy Editor

Karen Annett

Indexer

Heather McNeill

Proofreader

Chuck Hutchinson

Technical Reviewers

Adam Purkiss

Gene Teglovic

Publishing Coordinator

Olivia Basegio

Cover Designer

Chuti Prasertsith

Compositor

Nonie Ratcliff



For Landon



Contents at a Glance

- 1 Introducing Force.com 1
- 2 Database Essentials 21
- 3 Database Security 71
- 4 Business Logic 99
- 5 Advanced Business Logic 143
- 6 User Interfaces 185
- 7 Advanced User Interfaces 233
- 8 Mobile User Interfaces 263
- 9 Batch Processing 281
- 10 Integration with Force.com 301
- 11 Advanced Integration 339
- 12 Social Applications 369
- Index 387

Table of Contents

1 Introducing Force.com 1

| | |
|--|----|
| Force.com in the Cloud Computing Landscape | 1 |
| Platform as a Service (PaaS) | 2 |
| Force.com as a Platform | 4 |
| Force.com Services | 7 |
| Inside a Force.com Project | 9 |
| Project Selection | 9 |
| Team Selection | 11 |
| Lifecycle | 12 |
| Tools and Resources | 15 |
| Sample Application: Services Manager | 17 |
| Background | 17 |
| User Roles | 18 |
| Development Plan | 19 |
| Summary | 19 |

2 Database Essentials 21

| | |
|----------------------------------|----|
| Overview of Force.com's Database | 21 |
| Objects | 22 |
| Fields | 23 |
| Relationships | 25 |
| Query Language | 26 |
| Data Integration | 29 |
| Working with Custom Objects | 32 |
| Force.com Developer Edition | 32 |
| Tools for Custom Objects | 33 |
| Object Creation | 35 |
| Field Creation | 38 |
| Entering and Browsing Data | 41 |
| Additional Database Features | 43 |
| Sample Application: Data Model | 49 |
| Data Model Design Goals | 49 |
| Data Model Specification | 50 |

- Implementing the Data Model 58
- Importing Data 64
- Summary 69

3 Database Security 71

- Overview of Database Security 71
- Object-Level Security 74
 - Profiles 74
 - Permission Sets 76
 - Field-Level Security 77
- Record-Level Security 79
 - Record Ownership 79
 - User Groups 80
 - Sharing Model 80
- Sample Application: Securing Data 84
 - Designing the Security Model 85
 - Implementing the Security Model 88
 - Testing the Security Model 94
- Summary 98

4 Business Logic 99

- Introduction to Apex 100
- Introducing the Force.com IDE 101
 - Installation 101
 - Force.com Perspective 101
 - Force.com Projects 103
 - Problems View 103
 - Schema Explorer 103
 - Apex Test Runner View 103
 - Execute Anonymous View 104
- Apex Language Basics 105
 - Variables 105
 - Operators 109
 - Arrays and Collections 110
 - Control Logic 113
 - Object-Oriented Apex 117
 - Understanding Governor Limits 120

| | |
|---|------------|
| Database Integration in Apex | 120 |
| Database Records as Objects | 121 |
| Database Queries | 122 |
| Persisting Database Records | 128 |
| Database Triggers | 130 |
| Database Security in Apex | 133 |
| Debugging Apex Using Developer Console | 133 |
| Checkpoints | 133 |
| Execution Logs | 134 |
| Unit Tests in Apex | 136 |
| Test Methods | 136 |
| Test Data | 136 |
| Running Tests | 137 |
| Sample Application: Validating Timecards | 138 |
| Force.com IDE Setup | 138 |
| Creating the Trigger | 138 |
| Unit Testing | 140 |
| Summary | 142 |
| 5 Advanced Business Logic | 143 |
| Aggregate SOQL Queries | 144 |
| Aggregate Functions | 144 |
| Grouping Records | 145 |
| Grouping Records with Subtotals | 146 |
| Additional SOQL Features | 148 |
| Inner Join and Outer Join | 148 |
| Semi-Join and Anti-Join | 150 |
| Multi-Select Picklists | 154 |
| Salesforce Object Search Language (SOSL) | 154 |
| SOSL Basics | 155 |
| SOSL in Apex | 155 |
| Transaction Processing | 156 |
| Data Manipulation Language (DML) Database Methods | 157 |
| Savepoints | 159 |
| Record Locking | 161 |

| | |
|--|-----|
| Apex Managed Sharing | 162 |
| Sharing Objects | 162 |
| Creating Sharing Rules in Apex | 163 |
| Sending and Receiving Email | 168 |
| Sending Email | 168 |
| Receiving Email | 172 |
| Dynamic Apex | 174 |
| Dynamic Database Queries | 175 |
| Schema Metadata | 177 |
| Dynamic Instance Creation | 179 |
| Custom Settings in Apex | 180 |
| Sample Application: Adding Email Notifications | 181 |
| Summary | 183 |

6 User Interfaces 185

| | |
|---|-----|
| Introduction to Visualforce | 186 |
| Overview of Visualforce | 186 |
| Getting Started with Visualforce | 188 |
| Visualforce Controllers | 191 |
| Standard Controllers | 191 |
| Custom Controllers | 193 |
| Controller Extensions | 197 |
| View Components | 198 |
| View Component Basics | 198 |
| Data Components | 200 |
| Action Components | 203 |
| Primitive Components | 204 |
| Force.com-Styled Components | 205 |
| Force.com User Interface Components | 208 |
| Visualforce and the Native User Interface | 209 |
| Standard Pages | 210 |
| Standard Buttons | 213 |
| Page Layouts | 213 |
| Custom Buttons and Links | 215 |
| Custom Tabs | 215 |

| | |
|-----------------------------------|-----|
| Visualforce in Production | 215 |
| Debugging and Tuning | 215 |
| Security | 218 |
| Error Handling | 220 |
| Governor Limits | 221 |
| Unit Tests | 222 |
| Sample Application: Skills Matrix | 222 |
| Basic Implementation | 224 |
| Full Implementation | 224 |
| Implementation Walk-Through | 225 |
| Summary | 232 |

7 Advanced User Interfaces 233

| | |
|--|-----|
| Asynchronous Actions | 233 |
| Partial Page Refresh | 234 |
| Action as JavaScript Function | 235 |
| Action as Timed Event | 237 |
| Action as JavaScript Event | 237 |
| Indicating Action Status | 238 |
| Modular Visualforce | 240 |
| Static Resources | 241 |
| Inclusion | 242 |
| Composition | 242 |
| Custom Visualforce Components | 244 |
| Dynamic Visualforce | 246 |
| Dynamic Field References | 246 |
| Component Generation | 248 |
| Single-Page Applications in Force.com | 250 |
| JavaScript Remoting | 250 |
| Force.com with AngularJS | 251 |
| Introduction to Force.com Sites | 254 |
| Enabling and Creating a Site | 254 |
| Security Configuration | 255 |
| Adding Pages to a Site | 256 |
| Authenticating Users | 257 |
| Sample Application: Enhanced Skills Matrix | 258 |
| Summary | 262 |

8 Mobile User Interfaces 263

- Overview of Salesforce Mobile Technology 263
 - Salesforce Applications 264
 - Custom Applications 265
- Getting Started with Mobile Web Applications 267
 - Frameworks 268
 - Data Access 269
 - Deployment 270
- Sample Application: Mobile Timecard Entry 272
- Summary 279

9 Batch Processing 281

- Introduction to Batch Apex 282
 - Batch Apex Concepts 282
 - Understanding the `Batchable` Interface 283
 - Applications of Batch Apex 284
- Getting Started with Batch Apex 285
 - Developing a Batch Apex Class 285
 - Working with Batch Apex Jobs 286
 - Using Stateful Batch Apex 289
 - Using an Iterable Batch Scope 290
 - Limits of Batch Apex 292
- Testing Batch Apex 293
- Scheduling Batch Apex 293
 - Developing Schedulable Code 293
 - Scheduling Batch Apex Jobs 294
- Sample Application: Missing Timecard Report 296
 - Creating the Custom Object 297
 - Developing the Batch Apex Class 298
 - Testing the Missing Timecard Feature 299
- Summary 300

10 Integration with Force.com 301

- Apex Callouts 301
 - Calling RESTful Services from Apex 302
 - Calling SOAP Services from Apex 304
- Calling into Force.com Using REST 306

| | |
|--|-----|
| Getting Started with Force.com REST API | 306 |
| Force.com REST API Walk-Through | 308 |
| Creating Custom Apex REST Web Services | 312 |
| Calling into Force.com Using SOAP | 314 |
| Understanding Force.com SOAP API | 314 |
| Using the Enterprise API | 322 |
| Creating Custom Apex SOAP Web Services | 326 |
| Sample Application: Anonymous Benchmarking | 329 |
| Visualforce Page Design | 330 |
| Visualforce Controller Design | 331 |
| Integrating the SOAP Web Service | 333 |
| Sample Implementation | 335 |
| Summary | 338 |

11 Advanced Integration 339

| | |
|--|-----|
| Introduction to the Force.com Streaming API | 340 |
| Overview | 340 |
| Getting Started with Force.com Streaming API | 341 |
| Working with the Force.com Bulk API | 344 |
| Overview | 345 |
| Importing Records | 346 |
| Exporting Records | 347 |
| Getting Started with Force.com Canvas | 349 |
| Overview | 349 |
| Getting Started with Force.com Canvas | 350 |
| Introduction to the Force.com Tooling API | 354 |
| Overview | 355 |
| Getting Started with Force.com Tooling API | 355 |
| Understanding the Force.com Metadata API | 360 |
| Overview | 360 |
| Getting Started with the Metadata API | 361 |
| Sample Application: Database Integration | 363 |
| Integration Scenario | 363 |
| Implementation Strategy | 363 |
| Sample Implementation | 364 |
| Summary | 366 |

12 Social Applications 369

Overview of the Chatter Data Model 370

Chatter Posts 370

Chatter Comments 374

Feed-Tracked Changes 376

Followed Records 376

Using Chatter in Apex 378

Introduction to the Chatter REST API 379

Working with Chatter Visualforce Components 380

Sample Application: Follow Project Team 382

Summary 386

Index 387

Acknowledgments

There are many people to thank for this book.

- **Laura Lewin:** Laura is an Executive Editor at Pearson. She's the person I email when I'm late on a chapter to apologize and offer unique excuses. No matter how friendly her response, which is always extremely friendly, the exchange helps pressure and shame me into working harder to meet the deadlines.
- **Adam Purkiss, Gene Teglovic:** The technical reviewers for this edition have really impressed me with what they caught in the draft. They verified all of the code listings and made countless suggestions for improvement throughout.
- **Songlin Qiu:** Songlin is a Development Editor at Pearson. There are no figure/listing numbering, styling, grammatical, or consistency problems that go unnoticed when she's on the job.
- **Olivia Basegio:** Olivia is an Editorial Assistant at Pearson. She's a big part of making the publishing process fairly painless.
- **Kavindra Patel, Nick Tran:** These two work at Salesforce.com and have been longtime supporters of the book, especially this third edition. I can't thank them enough.
- **Jay Gauthier:** Jay is the VP of R&D at Software AG. His detailed feedback on the second edition of this book drove some of the improvements found in this edition.
- **Gretchen, Mark, Tom, and Nate:** Writing this book made me true to my panda name, so +1,000 for your associated pain and suffering. Now that it's done, I need a new name, like Well-Tempered Panda.
- **Tracey:** Thank you for supporting me as always, checking on me to see if I'm still alive in my writing chair, and making "rocket fuel" (iced coffee), which lost its kick somewhere around Chapter 6.

About the Author

Jason Ouellette is a SaaS entrepreneur and independent technology consultant with 17 years of experience in the enterprise software industry, including 9 years of hands-on work with Salesforce.com. He is currently CTO and Co-Founder of SocialPandas, a SaaS product company focused on converting social data into actionable intelligence for salespeople. In his prior role as Chief Architect of Appirio, a leading Salesforce.com consultancy, he led the development of popular Salesforce AppExchange applications such as Cloud Sync, Cloud Factor, and Professional Services Enterprise. He was recognized by Salesforce as a Force.com MVP in 2011–2013, and Force.com Developer Hero in 2009. He has a B.S. in Information and Decision Systems from Carnegie Mellon University.

Preface

I wrote this book to help developers discover Force.com as a viable, even superior tool for building business applications.

I'm always surprised at how many developers I meet who aren't aware of Force.com as a platform. They know of Salesforce, but only that it's a CRM. Even those who have heard of Force.com are amazed when I describe what Appirio and other companies are building with it. "I didn't know you could do that with Force.com" is a common reaction, even to the simplest of things such as creating custom database tables.

Since the second edition of this book, Salesforce has delivered more than six major releases. This third edition refocuses the book on custom application development and away from "clicks not code"-style, configuration-driven features. It contains updates throughout to cover new capabilities such as Developer Console, JSON support, Streaming and Tooling APIs, REST integration, and support for MVC frameworks like AngularJS in Visualforce. It also features a new chapter: Chapter 8, "Mobile User Interfaces."

Although there are more cloud-based application development platforms than ever before, Force.com continues to offer unique and outstanding value for business applications. With its core strength in customer data management, deep set of thoughtfully integrated features, and support for open standards, Force.com can save you significant time and effort throughout the software development lifecycle.

Key Features of This Book

This book covers areas of Force.com relevant to developing applications in a corporate environment. It takes a hands-on approach, providing code examples and encouraging experimentation. It includes sections on the Force.com database, Apex programming language, Visualforce user interface technology, integration to other systems, and supporting features such as workflow and analytics. SFA, CRM, customer support, and other prebuilt applications from Salesforce are not discussed, but general Force.com platform skills are helpful for working in these areas as well. The book does not cover cloud computing in general terms. It also avoids comparing Force.com with other technologies, platforms, or languages. Emphasis is placed on understanding Force.com on its own unique terms rather than as a database, application server, or cloud computing platform.

Although Force.com is a commercial service sold by Salesforce, all the material in this book was developed using a free Force.com Developer Edition account. Additionally, every feature described in this book is available in the free edition.

Throughout the text, you will see sidebar boxes labeled Note, Tip, or Caution. Notes explain interesting or important points that can help you understand key concepts and techniques. Tips are little pieces of information that will help you in real-world situations, and often offer shortcuts to make a task easier or faster. Cautions provide information about detrimental performance issues or dangerous errors. Pay careful attention to Cautions.

Target Audience for This Book

This book is intended for application developers who use Java, Ruby, or other high-level languages to build Web and rich client applications for end users. It assumes knowledge of relational database design and queries, Web application development using HTML and JavaScript, and exposure to Web services.

Code Examples for This Book

The code listings in this book are available on Github: <http://goo.gl/fjRqMX>. They are also available as a Force.com IDE project, also freely available on Github: <https://github.com/jmouel/dev-with-force-3e>.

Editor's Note: We Want to Hear from You!

As the reader of this book, you are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

You can email or write me directly to let me know what you did or didn't like about this book—as well as what we can do to make our books stronger.

Please note that I cannot help you with technical problems related to the topic of this book, and that due to the high volume of mail I receive, I might not be able to reply to every message.

When you write, please be sure to include this book's title and author as well as your name and phone number or email address. I will carefully review your comments and share them with the author and editors who worked on the book.

Email: laura.lewin@pearson.com

Mail: Laura Lewin
Executive Editor
Addison-Wesley/Pearson Education, Inc.
75 Arlington St., Ste. 300
Boston, MA 02116

Business Logic

Business logic in Force.com is developed in Apex, a programming language designed for the Force.com platform. Through Apex code, many platform features, such as the database and user interface, can be customized to meet the needs of individual users and companies.

This chapter introduces Apex as a language for writing business logic, specifically where it interacts with the Force.com database. It uses a combination of explanatory text and code snippets to introduce concepts and encourage experimentation. This approach assumes you're already experienced in some other high-level, object-oriented programming language and would like to see for yourself how Apex is different.

The chapter consists of the following sections:

- **Introduction to Apex**—Learn basic facts about Apex and how it differs from other programming languages.
- **Introducing the Force.com IDE**—Take a brief tour of the Force.com IDE, a user interface for developing, debugging, and testing Apex code.
- **Apex language basics**—Learn the building blocks of the Apex language, such as data types and loops.
- **Database integration in Apex**—Incorporate the Force.com database into your Apex programs through queries, statements that modify data, and code executed automatically when data is changed.
- **Debugging Apex using Developer Console**—With Developer Console, you can directly inspect the state of your Apex code as it runs.
- **Unit tests in Apex**—Write tests for your code and run them in Developer Console.
- **Sample application**—Walk through the implementation of a data validation rule for the Services Manager sample application.

Note

The code listings in this chapter are available in a GitHub Gist at <http://goo.gl/evtet>.

Introduction to Apex

Apex is a stored procedure-like language that runs entirely on the Force.com platform. It provides object-oriented features and tight integration with the Force.com database. It's mainly used in custom user interfaces and in triggers, code that is executed when data is changed in the database.

Apex is not a general-purpose programming language like Java or C. Its scope is limited to business and consumer applications that operate on relational data and can benefit from the feature set of the surrounding Force.com platform.

Apex programs exist in a multitenant environment. The computing infrastructure used to execute Apex is operated by Salesforce and shared among many developers or tenants of the system. As a result, unlike general-purpose programming languages you are familiar with, the execution of Apex programs is closely controlled to maintain a consistently high quality of service for all tenants.

This control is accomplished through governor limits, rules that Force.com places on programs to keep them operating within their allotted share of system resources. Governor limits are placed on database operations, memory and bandwidth usage, and lines of code executed. Some governor limits vary based on the type of licensing agreement you have in place with Salesforce or the context that the code is running in, and others are fixed for all users and use cases.

Note

The most prevalent governor limits are discussed throughout this book, but it is not a complete treatment of the subject. The authoritative guide to governor limits is the *Force.com Apex Code Developer's Guide*, available at <http://developer.force.com>. Educate yourself on governor limits early in the development process. This education will alter the way you architect your Apex code and prevent costly surprises. Additionally, test all of your Apex code with production-like data volumes. This helps to expose governor-related issues prior to a production deployment.

Here are a few important facts about Apex:

- **It includes integrated testing features.** Code coverage is monitored and must reach 75% or greater to be deployed into a production environment.
- **It is automatically upgraded.** Salesforce executes all of its customers' unit tests to verify that they pass before deploying a major release of the Force.com platform. Your code is always running on the latest version of Force.com and can take advantage of any and all new functionality without the hassle and risks of a traditional software upgrade process.
- **There is no offline runtime environment for Force.com.** You can edit your code on your desktop computer, but it must be sent to Force.com for execution.
- **Apex is the only language that runs on the Force.com platform.** You can integrate Apex with programs running outside of Force.com using HTTP-based techniques such as REST.

- **The Force.com database is the only database integrated into the Apex language.**
Other databases can be integrated through Web services or other technology using HTTP.

The two primary choices for developing Apex code are the Web-based App Builder Tools and the Force.com IDE, provided as a stand-alone application as well as a plug-in to the standard Eclipse IDE. The Force.com IDE is the more powerful and developer-friendly of the two, so it is used throughout this book.

Introducing the Force.com IDE

The Force.com IDE is an extension to the standard Eclipse development tool for building, managing, and deploying projects on the Force.com platform. This section covers installation and gives a brief walk-through of the Force.com IDE components used throughout this book.

Installation

The Force.com IDE is distributed in two forms: a stand-alone application and a plug-in to the Eclipse IDE. If Force.com is your primary development language or you are not an existing Eclipse IDE user, the stand-alone version is a good choice. The plug-in version of the Force.com IDE requires Eclipse, which you can find at www.eclipse.org. Only specific versions of Eclipse are supported by the Force.com IDE. If you are already using Eclipse but it's an unsupported version, keep your existing Eclipse version and install the supported version just for use with the Force.com IDE. Multiple versions of Eclipse can coexist peacefully on a single computer.

Visit http://wiki.developerforce.com/index.php/Apex_Toolkit_for_Eclipse to learn how to install the stand-alone and plug-in versions of the Force.com IDE.

Force.com Perspective

A perspective is a concept used by Eclipse to describe a collection of user interface components. For example, Eclipse has built-in perspectives called Java and Java Debug. By installing the Force.com IDE, you've added a perspective called Force.com. Figure 4.1 shows the Force.com perspective, indicated in the upper-right corner.

If you do not see the Force.com perspective, click the menu option Window, Open Perspective, Other; select Force.com from the Open Perspective dialog; and click the OK button. The Open Perspective dialog is shown in Figure 4.2.

The Force.com perspective includes several user interface panels, called Views. You can see two of them at the bottom of Figure 4.1: Execute Anonymous and Apex Test Runner. It also adds a new type of project called the Force.com Project, which is shown in the left-side Navigator tab. The first step to using the Force.com IDE is to create a Force.com Project.

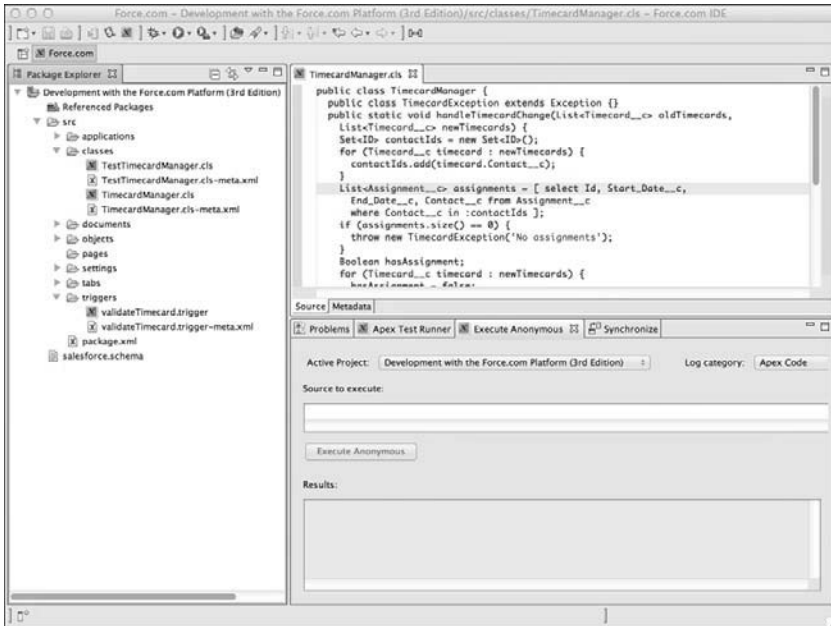


Figure 4.1 Force.com perspective

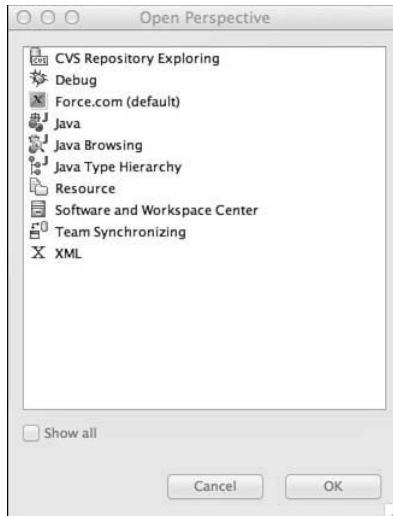


Figure 4.2 Open Perspective dialog

Force.com Projects

A Force.com Project allows you to read and write code, user interfaces, and other metadata objects within a Force.com organization from your local computer. Although this metadata is edited locally, it must be deployed to the Force.com service to run. Deployment to Force.com occurs automatically every time you make a modification to an object in a Force.com Project and save the changes. The contents of a Force.com Project are visible in the Navigator or Package Explorer Views.

Note

Force.com does not provide its own integrated source control system, but Force.com Projects can be integrated into your company's source control system through the built-in Team features of Eclipse. Refer to the Eclipse documentation for more information.

Problems View

The Force.com IDE leverages the standard Eclipse View called Problems to display compilation errors. When you save changes to an object in a Force.com Project, it is sent over the network to the Force.com service for compilation. If compilation fails, Force.com-specific errors are added to the Problems View. In most cases, you can double-click a problem row to navigate to the offending line of code.

Schema Explorer

The Schema Explorer allows direct interaction with the Force.com database. Use it to inspect objects and fields and to execute database queries and preview their results. To open the Schema Explorer, double-click the object named `salesforce.schema` in any Force.com Project. In Figure 4.3, the Schema Explorer is open and displaying the fields in the Project object in its right panel. In its left panel, a query has been executed and has returned a list of Contact records.

Apex Test Runner View

All business logic written in Force.com must be accompanied by unit tests to deploy it to a production environment. Apex Test Runner View is a user interface to run unit tests and view test results, including statistics on code performance and test coverage. If the Apex Test Runner is not already visible on the bottom of your screen, go to the Window menu and select Show View, Apex Test Runner.

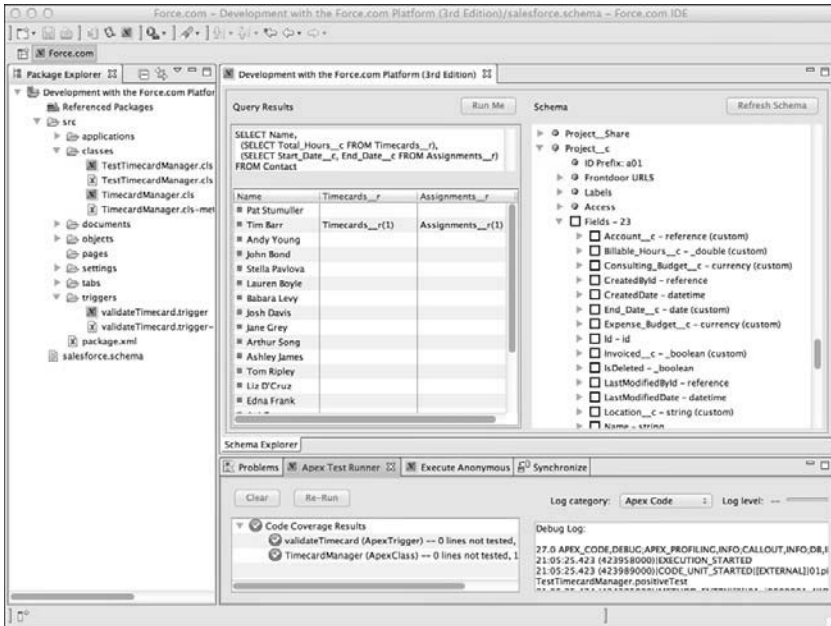


Figure 4.3 Force.com IDE Schema Explorer

Execute Anonymous View

The Execute Anonymous View provides an interactive, immediate way to execute arbitrary blocks of Apex code. Unless noted otherwise, you can execute all the code snippets in this chapter directly from the Force.com IDE using the Execute Anonymous View.

To try the Execute Anonymous View, first create a new Force.com Project. Go to the File menu and select File, New Force.com Project. Enter a project name; enter your Force.com username, password, and security token; and click the Next button. If you receive an error on this step, double-check your username, password, and security token. Also make sure you're providing the credentials for a Developer Edition organization, given that other types of organizations might not have access to the Force.com API. Select the metadata components Apex and Visualforce; then click the Finish button to create the project.

After you've created a project for your Development Edition organization, the Execute Anonymous View should be visible in the lower-right half of the screen. If not, go to the Window menu and select Show View, Execute Anonymous. In the Source to Execute text box, enter the code given in Listing 4.1. If the text box is not visible, resize your Execute Anonymous View until it's tall enough to see it. If the text box is disabled, double-click the Execute Anonymous tab to maximize and enable it. After you've entered the code, click the Execute Anonymous button to run it.

Listing 4.1 Hello World

```
String helloWorld(String name) {  
    return 'Hello, ' + name;  
}  
System.debug(helloWorld('Apex'));
```

This sample code defines a function called `helloWorld` that accepts a single `String` parameter. It then invokes it with the name `Apex` and displays the results, `Hello Apex`, to the debug log.

Apex Language Basics

This section describes the building blocks of the Apex language. The building blocks are variables, operators, arrays and collections, and control logic. Basic knowledge of the syntax and operation of Apex is valuable for almost any custom development task in Force.com, including triggers, custom user interfaces, and integration with external systems. The section concludes with an introduction to Apex governor limits. Knowledge of governor limits is a critical part of writing business logic that scales from Developer Edition organizations to production organizations with real-world data volumes.

Variables

This subsection covers variable declaration, data types, constants and enums, and type conversions. It also provides detail on rounding numbers and converting dates to and from strings, common tasks in business applications.

Variable Declaration

Apex is a strongly typed language. All variables must be declared before they're referenced. At minimum, a variable declaration consists of the data type followed by the variable name. For example, Listing 4.2 is a valid statement.

Listing 4.2 Variable Declaration

```
Integer i;
```

The variable `i` is declared to be an `Integer`. Apex does not require variables to be initialized before use, but doing so is good practice. The variable `i` initially contains a null value.

Variable names cannot start with numbers or symbols, cannot contain two or more consecutive underscore characters, and must not conflict with Apex reserved words. These are special keywords used by the Apex language itself. The list of reserved words is available in the *Force.com Apex Code Developer's Guide*.

Variable names are not case sensitive. Try defining two variables with the same name, one in uppercase and one in lowercase, to prove this, as in Listing 4.3. If you try to execute this code, you will receive a compilation error citing a duplicate variable.

Listing 4.3 Case Insensitivity of Variable Names

```
Integer i;
String I;
```

Data Types

In Apex, all data types are objects. There is no concept of a primitive type such as an `int` in Java. Table 4.1 lists Apex’s standard atomic data types. These types contain a single value at a time or a null value.

Table 4.1 Standard Atomic Data Types

| Data Type | Valid Values |
|-----------|---|
| String | Zero or more Unicode characters. |
| Boolean | True or false. |
| Date | Date only; no time information is included. |
| Datetime | Date and time value. |
| Time | Time only; no date information is included. |
| Integer | 32-bit signed whole number (–2,147,483,648 to 2,147,483,647). |
| Long | 64-bit signed whole number (–263 to 263–1). |
| Decimal | Signed number with whole (<i>m</i> , Integer) and fractional components (<i>n</i>), expressed as <i>m.n</i> . Total length of number, including sign and decimal point, cannot exceed 19 characters. |
| Double | 64-bit signed number with a decimal point (–263 to 263–1). |
| Blob | Binary data. |
| ID | ID is a variation of the String type to store the unique identifiers for Force.com database records. ID values are restricted to 18 characters. Values are checked at compile and runtime, and a <code>StringException</code> is thrown if they do not conform. |
| Object | Object is the generic type. Variables defined as Object are essentially typeless and can receive any value. Typeless code is vulnerable to runtime errors because it is invisible to the compiler’s type checking functionality. |

Constants and Enums

A constant is a variable that cannot be modified after it has been initialized. It is declared using the `final` keyword and can be initialized only in constructors, in initializers, or in the declaration itself.

An enum is a set of identifiers. Listing 4.4 provides an example of a constant as well as an enum. The constant is an Integer type; the enum is named `MyConstants` and contains three members. The variable `x` is initialized to the first member, and its data type is the enum itself, which can be thought of as a user-defined data type.

Listing 4.4 Defining an Integer Constant and an Enum

```
final Integer MAGIC_NUMBER = 42;
Enum MyConstants { One, Two, Three }
MyConstants x = MyConstants.One;
```

After it has been declared, an enum can be referenced in Apex code like any built-in data type. It can also be converted into an Integer from its zero-indexed position using its `ordinal` method or into a String using its `name` method.

Converting Data Types

The two ways to convert one data type to another are implicit and through conversion methods. Implicit conversion means that no method calls or special notation is required to convert one type into another. Conversion methods are functions that explicitly convert a value from one type to another type.

Implicit conversion is supported for numeric types and String types. For numbers, the rule is this: Integer → Long → Double → Decimal. Conversions can move from left to right without casting, as Listing 4.5 demonstrates.

Listing 4.5 Implicit Conversion of Numeric Types

```
Integer i = 123;
Long l = i;
Double d = l;
Decimal dec = d;
```

For Strings, ID and String are interchangeable, as shown in Listing 4.6. If conversion is attempted from String to ID but the String is not a valid ID, a `System.StringException` is thrown.

Listing 4.6 Converting between ID and String

```
String s = 'a0I80000003hazV';
ID id = s;
String s2 = id;
```

When implicit conversion is not available for a pair of types, you must use a conversion method. Data type objects contain a static conversion method called `valueOf`. Most conversions can be handled through this method. Listing 4.7 is a series of statements that convert a string into the various numeric types.

Listing 4.7 Type Conversion Methods

```
String s = '1234';
Integer i = Integer.valueOf(s);
Double d = Double.valueOf(s);
Long l = Long.valueOf(s);
Decimal dec = Decimal.valueOf(s);
```

When a type conversion method fails, it throws a `NumberFormatException`. For example, when the code in Listing 4.8 executes, it results in an error: `System.TypeException: Invalid integer: 1234.56`.

Listing 4.8 Type Conversion Error

```
String s = '1234.56';
Integer i = Integer.valueOf(s);
```

Rounding Numbers

Rounding occurs when the fractional component of a `Decimal` or `Double` is dropped (`round`), or when a `Decimal` is divided (`divide`) or its scale (number of decimal places) reduced (`setScale`). Apex has a set of rounding behaviors called rounding modes that apply in all three of these situations. By default, the rounding mode is `HALF_EVEN`, which rounds to the nearest neighbor, or to the even neighbor if equidistant. For example, 0.5 rounds to 0, and 0.6 to 1. For the complete list of rounding modes, refer to the *Force.com Apex Code Developer's Guide* at www.salesforce.com/us/developer/docs/apexcode/index.htm.

Listing 4.9 demonstrates the three operations that can cause rounding.

Listing 4.9 Three Rounding Operations

```
Decimal d = 123.456;
Long rounded = d.round(RoundingMode.HALF_EVEN);
Decimal divided = d.divide(3, 3, RoundingMode.HALF_EVEN);
Decimal reducedScale = d.setScale(2, RoundingMode.HALF_EVEN);
```

Converting Strings to Dates

Strings can be converted to Date and Datetime types using the `valueOf` conversion methods, but the string values you're converting from must be in a specific format. For Date, the format is `YYYY-MM-DD`; for Datetime, `YYYY-MM-DD HH:MM:SS`, regardless of the locale setting of the user. Time does not have a `valueOf` method, but you can create one with its `newInstance` method, providing hours, minutes, seconds, and milliseconds. Listing 4.10 shows the creation of all three types.

Listing 4.10 Creating Date, Datetime, and Time

```
Date d = Date.valueOf('2015-12-31');
Datetime dt = Datetime.valueOf('2015-12-31 02:30:00');
Time t = Time.newInstance(2,30,0,0);
```

Converting Dates to Strings

Dates can be converted to strings through the `String.valueOf` method. This applies a default format to the date values. If you want control over the format, Datetime has a `format` method that accepts a Date pattern. This pattern follows the `SimpleDateFormat` pattern found in the Java API, which is documented at the following URL: <http://download.oracle.com/javase/1.4.2/docs/api/java/text/SimpleDateFormat.html>. For example, the code in Listing 4.11 outputs `Thu Dec 31, 2020`.

Listing 4.11 Formatting a Datetime

```
Datetime dt = Datetime.valueOf('2020-12-31 00:00:00');
System.debug(dt.format('E MMM dd, yyyy'));
```

Operators

Apex supports the standard set of operators found in most languages. Each operator is listed in Table 4.2 along with its valid data types, precedence if mathematical, and a brief description. In an expression with two operators, the operator with lower precedence is evaluated first.

Table 4.2 Operators, Their Data Types, and Precedence

| Operators | Operands | Precedence | Description |
|-----------|----------------------|------------|--|
| = | Any compatible types | 9 | Assignment |
| +, - | Date, Datetime, Time | 4 | Add or subtract days on Date, Datetime, milliseconds on Time, argument must be Integer or Long |
| + | String | N/A | String concatenation |

| Operators | Operands | Precedence | Description |
|--|--------------------------------|---|--|
| <code>+</code> , <code>-</code> , <code>*</code> , <code>/</code> | Integer, Long, Double, Decimal | 4 | Numeric add, subtract, multiply, divide |
| <code>!</code> | Boolean | 2 | Logical negation |
| <code>-</code> | Integer, Long, Double, Decimal | 2 | Arithmetic negation |
| <code>++</code> , <code>--</code> | Integer, Long, Double, Decimal | 1 | Unary increment, decrement |
| <code>&</code> , <code> </code> , <code>^</code> | Integer, Long, Boolean | 10 | Bitwise AND, OR, XOR |
| <code><<</code> , <code>>></code> , <code>>>></code> | Integer, Long | 10 | Signed shift left, signed shift right, unsigned shift right |
| <code>==</code> , <code><</code> , <code>></code> , <code><=</code> , <code>>=</code> , <code>!=</code> | Any compatible types | 5 (<code><</code> , <code>></code> , <code><=</code> , <code>>=</code>), 6 (<code>==</code> , <code>!=</code>) | Not case sensitive, locale-sensitive comparisons: equality, less than, greater than, less than or equal to, greater than or equal to, not equal to |
| <code>&&</code> , <code> </code> | Boolean | 7 (<code>&&</code>), 8 (<code> </code>) | AND, OR, with short-circuiting behavior (second argument is not evaluated if first argument is sufficient to determine result) |
| <code>===</code> , <code>!==</code> | Map, List, Set, Enum, SObject | N/A | Exact equality, exact inequality |
| <code>()</code> | Any | 1 | Group an expression and increase its precedence |
| <code>? :</code> | Boolean | N/A | Shortcut for <code>if/then/else</code> expression |

Operators not included in Table 4.2 are the assignment variations of date, string, and numeric (`+=`, `-=`, `*=`, `/=`) and bitwise (`|=`, `&=`, `^=`, `<<=`, `>>=`, `>>>=`) arithmetic. For example, `x = x + 3` assigns `x` to itself plus 3, but so does `x += 3`.

Arrays and Collections

Arrays and collections are a family of data types that contain a sequence of values. It includes Lists and Arrays, Sets, and Maps. This subsection covers each of the three types and describes how to create them and perform some basic operations. Each collection type is different, but there are four methods you can invoke on all of them:

1. **clear**—Removes all elements from the collection
2. **clone**—Returns a copy of the collection
3. **isEmpty**—Returns `false` if the collection has elements, `true` if empty
4. **size**—Returns the number of elements in the collection as an Integer

Lists and Arrays

Lists and Arrays contain an ordered sequence of values, all the same type. Duplicate values are allowed. Unlike Lists, the length of an Array is fixed when you initialize it. Lists have a dynamic length that is adjusted as you add and remove elements.

To declare a List variable, use the `List` keyword followed by the data type of its values in angle brackets. Because Lists and Arrays are containers for other values, they must be initialized before values can be added to them. The `new` keyword creates an instance of the List. Listing 4.12 declares a variable called `stringList` that contains Strings, initializes it, and adds a value.

Listing 4.12 Creating a List

```
List<String> stringList = new List<String>();
stringList.add('Hello');
```

To create an Array, specify a variable name, data type, and length. Listing 4.13 creates an Array of Strings named `stringArray`, initializes it to accommodate five elements, and then assigns a value to its first element.

Listing 4.13 Creating an Array

```
String[] stringArray = new String[5];
stringArray[0] = 'Hello';
```

Multidimensional Arrays are not supported. But you can create a two-dimensional List object by nesting a List within another List. In Listing 4.14, `list2` is defined as a List containing Lists of Strings. A String List called `childList` is initialized, populated with a value, and added to `list2`.

Listing 4.14 Nested List Usage

```
List<List<String>> list2 = new List<List<String>>();
List<String> childList = new List<String>();
childList.add('value');
list2.add(childList);
```

Arrays and Lists have interchangeable behavior and syntax in Apex, as demonstrated in Listing 4.15. Lists can be initialized using an Array initializer, and its elements accessed using the square-bracket notation. Arrays can be initialized using the List constructor, and accessed using the List getters and setters. But for the sake of code clarity, picking one usage style and sticking with it is a good idea. In this book, List is the standard because it better reflects the object-oriented nature of these collection types.

Listing 4.15 Mixed Array and List Syntax

```
List<Integer> intList = new Integer[3];
intList[0] = 123;
intList.add(456);
Integer[] intArray = new List<Integer>();
intArray.add(456);
intArray.set(0, 123);
```

Arrays and Lists preserve the order in which elements are inserted. They can also be sorted in ascending order using the `sort` method of the `List` object. For custom sorting behavior, you can implement the `Comparable` interface on the classes in your list. This interface allows you to examine two objects and let Force.com know if the objects are equal or if one occurs before the other.

Sets

The `Set` is another collection type. Like a `List`, a `Set` can store only one type of element at a time. But `Sets` do not allow duplicate values and do not preserve insertion order. `Sets` are initialized like `Lists`. In Listing 4.16, a set named `stringSet` is created, and two values are added.

Listing 4.16 Basic Set Usage

```
Set<String> stringSet = new Set<String>();
stringSet.add('abc');
stringSet.add('def');
System.debug(stringSet.contains('abc'));
```

The final statement in Listing 4.16 outputs `true`, illustrating one of the most valuable features of the `Set` collection type: its `contains` method. To test whether a particular `String` exists in an `Array` or a `List`, every element of the `List` must be retrieved and checked. With a `Set`, this test can be done more efficiently thanks to the `contains` method.

Maps

The `Map` type stores pairs of keys and values and does not preserve their insertion order. It maintains the relationship between key and value, functioning as a lookup table. Given a key stored in a `Map`, you can retrieve its corresponding value.

`Maps` are initialized with a key data type and value data type. Listing 4.17 initializes a new `Map` called `myMap` to store `Integer` keys and `String` values. It inserts a single value using the `put` method and then retrieves it using the `get` method. The last line of code prints `abc` because that is the value associated with the key `123`.

Listing 4.17 Basic Map Usage

```
Map<Integer, String> myMap = new Map<Integer, String>();
myMap.put(123, 'abc');
System.debug(myMap.get(123));
```

Other useful methods of Maps include `containsKey` (returns `true` if the given key exists in the Map), `remove` (returns and removes an element by key), `keySet` (returns a Set of all keys), and `values` (returns an Array of all values).

Control Logic

This subsection describes how to control the flow of Apex code execution. It covers conditional statements, loops, exception statements, recursion, and asynchronous execution.

Conditional Statements

Conditional statements evaluate a Boolean condition and execute one code block if `true`, another if `false`. Listing 4.18 provides an example, defining a function that prints `true` if an Integer argument is greater than 100, `false` otherwise.

Listing 4.18 Conditional Statement Usage

```
void testValue(Integer value) {
    if (value > 100) {
        System.debug('true');
    } else {
        System.debug('false');
    }
}
testValue(99);
testValue(101);
```

In addition to this simple `if, else` structure, you can chain multiple conditional statements together using `else if`.

Note

In conditional code blocks that contain a single statement, the curly braces around them can be omitted. This is true of all the control logic types in Apex. For example, `if (a > 0) return 1 / a; else return a;` is a valid statement.

Loops

Loops in Apex behave consistently with other high-level languages. Table 4.3 lists the loop statements available in Apex.

Table 4.3 Types of Loops

| Name | Syntax | Description |
|-----------------------------|--|--|
| Do-While Loop | <pre>do { code_block } while (condition);</pre> | Executes code block as long as Boolean condition is true. Evaluates condition after running code block, executing the code block at least once. |
| While Loop | <pre>while (condition) { code_block; }</pre> | Executes code block as long as Boolean condition is true. Evaluates condition before running code block, so code block might not be executed at all. |
| Traditional For Loop | <pre>for (init; exit condition; increment) { code_block; }</pre> | Executes init statement once. Loops on the following steps: exit loop if Boolean exit condition evaluates to false, executes code block, executes increment statement. |
| List/Set Iteration For Loop | <pre>for (var : list/set) { code_block }</pre> | For every element of the list or set, assigns var to the current element and executes the code block. Cannot modify the collection while iterating. |

The keywords `break` and `continue` can be used to further control the loops. To immediately exit a loop at any point in its execution, use `break` in the code block. To abort a cycle of loop execution in the middle of a code block and move to the next cycle, use `continue`.

Exception Statements

Exceptions are classes used to signal a problem at runtime. They abort the normal flow of code execution, bubbling upward until explicitly handled by some other code, carrying with them information about the cause of the problem.

Apex allows custom exception classes to be defined that are meaningful to your programs. It also provides system exception classes corresponding to areas of the Force.com platform. Some common system exceptions are `DmlException` (issues with changes to the database), `NullPointerException` (attempt to dereference a null value), `QueryException` (issues with database queries), and `TypeException` (issues converting data types).

The two ways to use exceptions in your code are to raise an exception with the `throw` keyword and handle an exception with the `try`, `catch`, and `finally` keywords:

1. **Raise an exception**—When your code cannot proceed due to a problem with its input or other issue, you can raise an exception. An exception stops execution of the code and provides information about the problem to its callers. Only custom exceptions,

classes that are subclasses of Force.com’s `Exception` class, can be raised. The names of all custom exception classes must end with the word *Exception*. Construct an instance of your exception class using an optional message or another exception as the preceding cause and provide it as an argument to the `throw` keyword.

- 2. Handle an exception**—An exception handler in Apex is a code block defined to expect and take action on one or more named exception classes. It consists of a `try` code block, zero or more `catch` code blocks, and optionally a `finally` code block. The `try` code block is executed first. If an exception is raised, Apex looks for a `catch` code block that matches the exception class. If it’s found, execution skips to the relevant `catch`. If not, the exception is bubbled upward to the caller. After the code in the `try` completes, successfully or not, the `finally` code block is executed.

Listing 4.19 demonstrates both forms of exception statements. It inserts a Timecard record within a `try` block, using a `catch` block to handle a database exception (`DmlException`). The code to handle the database exception itself raises an exception, a custom exception class called `MyException`. It ends by printing a final message in the `finally` block.

Listing 4.19 Sample Exception Statements

```
class MyException extends Exception {}
Timecard__c timecard = new Timecard__c();
try {
    insert timecard;
} catch (DmlException e) {
    throw new MyException('Could not create Timecard record: ' + e);
} finally {
    System.debug('Exiting timecard creation code');
}
```

Recursion

Apex supports the use of recursion in code. The maximum stack depth is not documented, so experiment with your own code before committing to a recursive algorithm. For example, the code in Listing 4.20 fails with `System.Exception: Maximum stack depth reached: 1001`.

Listing 4.20 Recursion with Unsupported Depth

```
Integer counter = 0;
void recursive() {
    if (counter < 500) {
        counter++;
        recursive();
    }
}
recursive();
```

Asynchronous Execution

Code in Apex normally is executed synchronously. From the user's point of view, there is a single thread of execution that must complete before another can begin. But Apex also supports an asynchronous mode of execution called future methods. Code entering a future method completes immediately, but the body of the method isn't executed until later, at a time determined by the Force.com platform.

The code in Listing 4.21 declares a future method called `asyncMethod` with a single parameter: a list of strings. It might use these strings to query records via SOQL and perform DML operations on them.

Listing 4.21 Future Method Declaration

```
@future
public static void asyncMethod(List<String> idsToProcess) {
    // code block
}
```

Future methods typically are used to perform expensive tasks that are not time-critical. A regular synchronous method can begin some work and invoke a future method to finish it. The future method starts fresh with respect to governor limits.

Future methods have many limitations, as follows:

- You cannot invoke more than ten future methods in a single scope of execution. There is no guarantee of when these methods will be executed by Force.com or in what order.
- Future methods cannot call other future methods.
- Future method signatures are always static and return void. They cannot use custom classes or database objects as parameters—only primitive types such as String and Integer and collections of primitive types.
- You cannot test future methods like ordinary methods. To write testable code that includes future methods, keep your future methods limited to a single line of code that invokes a normal method to perform the actual work. Then in your test case, call the normal method so that you can verify its behavior.
- Force.com limits your usage of future methods in a 24-hour period to 250,000 or 200 per licensed user, whichever is greater. This limit is shared with Batch and Scheduled Apex.

Note

Batch Apex is an additional feature for asynchronous execution. It provides much greater control than future methods and supports processing of millions of records. Batch Apex is covered in Chapter 9, “Batch Processing.”

Object-Oriented Apex

Apex is an object-oriented language. This subsection describes Apex in terms of five standard characteristics of object-oriented languages, summarized here:

- **Encapsulation**—Encapsulation combines the behavior and internal state of a program into a single logical unit.
- **Information hiding**—To minimize tight coupling between units of a program, information hiding limits external visibility into the behavior and state of a unit.
- **Modularity**—The goal of modularity is to establish clear boundaries between components of a program.
- **Inheritance**—Inheritance allows one unit of code to define its behavior in terms of another.
- **Polymorphism**—Polymorphism is the capability to interact with multiple units of code interchangeably without special cases for each.

These principles of object-oriented programming help you learn the Apex syntax and behaviors from a language-neutral point of reference.

Encapsulation

Encapsulation describes the bundling of a program's behavior and state into a single definition, usually aligned with some real-world concept. In Apex that definition is a class.

When a class is defined, it becomes a new data type in Apex. Classes contain variables, methods, properties, constructors, initializers, and inner classes. These components are summarized in the following list, and their usage is demonstrated in Listing 4.22:

- **Variables**—Variables hold the state of an object instance or class. By default, variables declared inside a class are scoped to individual object instances and are called member variables. Every instance of an object gets its own member variables and can read and write their values independently without interfering with the values stored in other object instances. There are also class variables, also known as static variables. They are declared using the `static` keyword. Static variables are shared across all instances of the object.
- **Methods**—Methods define the verbs in a class, the actions to be taken. By default, they operate within the context of individual object instances, able to access all visible member variables. Methods can also be static, operating on the class itself. Static methods have access to static variables but never member variables.
- **Properties**—A property is a shortened form of a method that provides access to a static or instance variable. An even shorter form is called an automatic property. These are properties with no code body. When no code is present, the logic is implied. Getters return their value; setters set their value.

- **Constructors**—A constructor is a special method executed when a class is instantiated. Constructors are declared much like methods, but share their name with the class name, and have no return type declaration.
- **Initializers**—An initializer contains code that runs before any other code in the class.
- **Inner classes**—An inner class is a class defined within another class.

Listing 4.22 Class Definition

```
class MyClass {
    static Integer count; /* Class variable */
    Integer cost; /* Member variable */
    MyClass(String c) { /* Constructor */ }
    void doSomething() { /* Method */ }
    Integer unitCost { get { return cost; } set { this.cost = value; } }
    Integer q { get; set; }
    { /* Initializer */ }
    class MyInnerClass { /* Inner class */ }
}
```

Tip

Code listings containing static variables or inner class declarations cannot be tested in the Execute Anonymous View of the Force.com IDE. Create a stand-alone class and then invoke it from the Execute Anonymous view. To create a stand-alone class in the Force.com IDE, select your Force.com Project and then select New, Apex Class from the File menu.

Information Hiding

Class definitions include notation to limit the visibility of their constituent parts to other code. This information-hiding notation protects a class from being used in unanticipated and invalid ways and simplifies maintenance by making dependencies explicit. In Apex, information hiding is accomplished with access modifiers. There are two places to use access modifiers: on classes, and on methods and variables:

- **Classes**—An access modifier of `public` makes a class visible to the entire application namespace, but not outside it. A `global` class is visible to Apex code running in every application namespace.
- **Methods and variables**—If designated `private`, a method or variable is visible only within its defining class. This is the default behavior. An access modifier of `protected` is visible to the defining class and subclasses, `public` is visible to any Apex code in the same application namespace but not accessible to other namespaces, and `global` can be used by any Apex code running anywhere in the organization, in any namespace.

Modularity

Apex supports interfaces, which are skeletal class definitions containing a list of methods with no implementation. A class built from an interface is said to implement that interface, which requires that its method names and the data types of its argument lists be identical to those specified in the interface.

The proper use of interfaces can result in modular programs with clear logical boundaries between components, making them easier to understand and maintain.

Inheritance

Apex supports single inheritance. It allows a class to extend one other class and implement many interfaces. Interfaces can also extend one other interface. A class extending from another class is referred to as its subclass.

For a class to be extended, it must explicitly allow it by using the `virtual` or `abstract` keyword in its declaration. Without one of these keywords, a class is `final` and cannot be subclassed. This is not true of interfaces because they are implicitly `virtual`.

By default, a subclass inherits all the functionality of its parent class. All the methods defined in the parent class are also valid on the subclass without any additional code. This behavior can be selectively overridden if the parent class permits. Overriding a method is a two-step process:

1. The parent class must specify the `virtual` or `abstract` keywords on the methods to be overridden.
2. In the subclass, the `override` keyword is used on the `virtual` or `abstract` methods to declare that it's replacing the implementation of its parent.

After it's overridden, a subclass can do more than replace the parent implementation. Using the `super` keyword, the subclass can invoke a method in its parent class, incorporating its functionality and potentially contributing its own.

Polymorphism

An object that inherits a class or implements an interface can always be referred to in Apex by its parent class or interface. References in variable, property, and method declarations treat the derived objects identically to objects they are derived from, even though they are different types.

This polymorphic characteristic of object types can help you write concise code. It works with the hierarchy of object types to enable broad, general statements of program behavior, behavior applying to many object types at once, while preserving the option to specify behavior per object type.

One example of using polymorphic behavior is method overloading, in which a single method name is declared with multiple argument lists. Consumers of the method simply invoke it by name, and Apex finds the correct implementation at runtime based on the object types.

Understanding Governor Limits

Governor limits are imposed on your running Apex code based on the type of resource consumed. When a governor limit is encountered, your code is immediately terminated with an exception indicating the type of limit reached. Examples of resource types are heap (memory used during execution) and SOQL queries.

Table 4.4 lists a few of the most important governor limits. Additional governor limits are introduced later in the book.

Table 4.4 **Subset of Governor Limits**

| Resource Type | Governor Limit |
|---------------|---|
| Heap | 6MB |
| Apex code | 1,000,000 lines of code executed, 3MB code size |
| Database | 50,000 records retrieved via SOQL |

Note

Namespaces are used to separate and isolate Apex code and database objects developed by different vendors so that they can coexist and interoperate in a single Force.com organization. Governor limits are applied independently to each namespace. For example, if you install a package from Force.com AppExchange, the resources consumed by code running inside that package do not count against the limits applied to your code.

Database Integration in Apex

In Apex, the Force.com database is already integrated into the language and runtime environment. There are no object-relational mapping tools or database connection pools to configure. Your Apex code is automatically aware of your database, including all of its objects and fields and the security rules protecting them.

This section examines the five ways the database is exposed in Apex code, which are summarized here:

1. **Database records as objects**—Database objects are directly represented in Apex as classes. These classes are implicitly imported into your code, so you're always developing from the latest database schema.
2. **Database queries**—SOQL is a concise expression of the records to be queried and returned to your programs.
3. **Persisting database records**—Apex has a built-in Data Manipulation Language (DML), providing verbs that create, update, or delete one or more records in the database.

4. **Database triggers**—Triggers are code that register interest in a specific action or actions on a database object, such as an insert or delete on the Account object. When this action occurs, the trigger code is executed and can inhibit or enhance the behavior of the database action.
5. **Database security in Apex**—Normally, Apex code runs in a privileged mode, granting it full access to all the data in the system. Alternatively, you can configure it to run under the same restrictions imposed on the current user, including object and record-level sharing rules.

Database Records as Objects

All database objects, standard and custom, are available as first-class members of the Apex language, automatically and transparently. This eliminates the mind-numbing, error-prone work of importing, mapping, and translating between relational and program data structures, chores commonly required in general-purpose programming languages. In Apex, references to database objects are verified at compile time. This reduces the possibility of runtime surprises caused by field or object mismatches. Listing 4.23 shows an example of creating a record in the Contact object and setting its first name field.

Listing 4.23 Creating a Record

```
Contact contact = new Contact();
contact.FirstName = 'Larry';
```

Database relationships are also exposed in Apex. The `__r` syntax refers to a relationship field, a field that contains a reference to another object or list of objects. Listing 4.24 builds on the previous listing, creating an Assignment record and associating it with the Contact record.

Listing 4.24 Creating a Record with Relationship

```
Assignment__c assignment = new Assignment__c();
assignment.Contact__r = contact;
```

The Force.com IDE's Schema Explorer can take the mystery out of relationship fields like `Contact__r`. It displays the correct syntax for referring to fields and relationships, based on the actual schema of the database object. Its Schema list on the right side displays all objects, custom and standard. Drilling into an object, the `Fields` folder lists all fields in the object and their types. A reference type indicates that a field is the child object in a Lookup relationship. Expand these fields to reveal their parent object's type and name. For example, in the Project custom object, `Account__r` is the foreign key to the Account object. This is demonstrated in Figure 4.4.

The focus of this section is SOQL because it is the workhorse of typical business applications. This section includes subsections on the basics of SOQL, filtering and sorting, how to query related objects, and how to use SOQL from Apex code.

As you read this section, you can experiment with the sample SOQL queries using the Force.com IDE's Schema Explorer. In the Navigator or Package Explorer View, expand the node for your Force.com Project and double-click `salesforce.schema`. Enter a query in the text box in the upper-left corner and click the Run Me button. The results appear in the table below the query. In Figure 4.5, a query has been executed against the Project object, returning four records. Note that many of the queries rely on objects from the Services Manager sample application rather than standard Force.com objects.

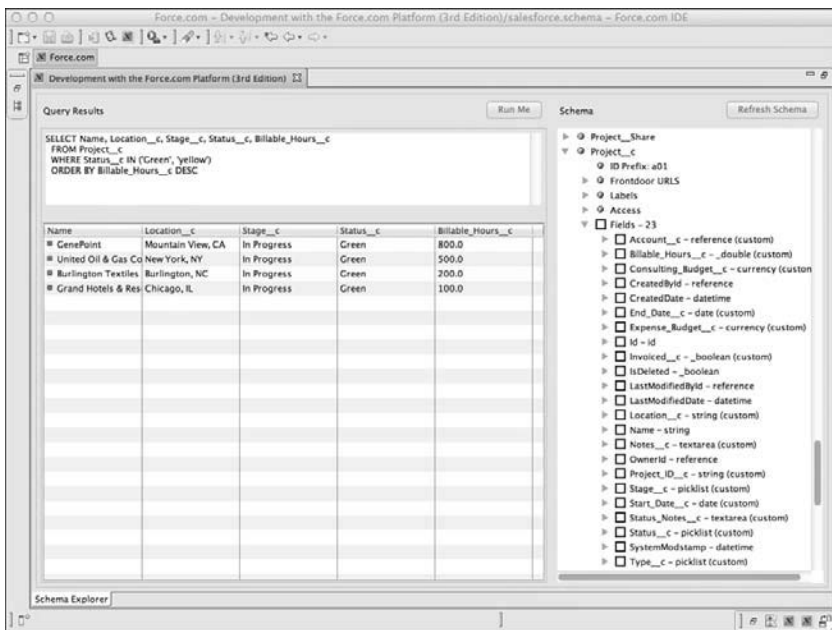


Figure 4.5 Running SOQL queries in Schema Explorer

Note

This book does not cover every feature and nuance of SOQL. For the complete specification, visit <http://developer.force.com> and download the latest Force.com SOQL and SOSL Reference.

SOQL Basics

Despite being one letter away from SQL and borrowing some of its syntax, SOQL is completely different and much easier to understand on its own terms. Just as Apex is not a general-purpose

programming language like Java, SOQL is not a general-purpose database query language like SQL. SOQL is specifically designed and optimized for the Force.com database.

A SOQL statement is centered on a single database object, specifying one or more fields to retrieve from it. The fields to select are separated by commas. Listing 4.26 is a simple SOQL statement that returns a list of Account records with Id and Name fields populated. SOQL is not case sensitive. SOQL keywords are shown throughout the book in uppercase and metadata objects in title case for readability only.

Listing 4.26 Simple SOQL Statement

```
SELECT Id, Name
FROM Account
```

Filtering Records

SOQL supports filter conditions to reduce the number of records returned. A filter condition consists of a field name to filter, an operator, and a literal value.

Valid operators are > (greater than), < (less than), >= (greater than or equal to), <= (less than or equal to), = (equal to), != (not equal to), IN and NOT IN (matches a list of literal values, and supports semi-joins and anti-joins), and INCLUDES and EXCLUDES (match against multi-select picklist values). On String fields, the LIKE operator is also available, which applies a pattern to filter records. The pattern uses the % wildcard to match zero or more characters, _ to match one character, and the \ character to escape the % and _ wildcards, treating them as regular characters.

Multiple filters are combined in a single SOQL statement using the Boolean operators AND and OR and grouped with parentheses. Listing 4.27 returns the names of accounts with a type of direct customer, a modification date sometime during the current year, and more than \$100 million in annual revenue.

Listing 4.27 SOQL Statement with Filter Conditions

```
SELECT Name
FROM Account
WHERE AnnualRevenue > 100000000
AND Type = 'Customer - Direct'
AND LastModifiedDate = THIS_YEAR
```

Notice the way literal values are specified. Single quotation marks must be used around String literals but never with other data types. THIS_YEAR is a built-in relative time function. The values of relative time functions vary based on when the query is executed. Other relative time functions are YESTERDAY, TODAY, TOMORROW, LAST_WEEK, THIS_WEEK, NEXT_WEEK, and so forth.

Absolute dates and times can also be specified without single quotation marks.

Dates must use the YYYY-MM-DD format. Datetimes can be YYYY-MM-DDThh:mm:ssZ,

YYYY-MM-DDThh:mm:ss+hh:mm, or YYYY-MM-DDThh:mm:ss-hh:mm, indicating the positive or negative offset from Coordinated Universal Time (UTC).

In addition to filter conditions, SOQL supports the `LIMIT` keyword. It sets an absolute upper bound on the number of records that can be returned from the query. It can be used in conjunction with all the other SOQL features. For example, the SOQL statement in Listing 4.28 returns up to ten Account records modified today.

Listing 4.28 SOQL Statement with Record Limit

```
SELECT Name, Type
FROM Account
WHERE LastModifiedDate = TODAY
LIMIT 10
```

Sorting Query Results

Results of a query can be sorted by up to 32 fields in ascending (`ASC`, the default) or descending (`DESC`) order. Sorting is not case sensitive, and nulls appear first unless otherwise specified (`NULLS LAST`). Multi-select picklists, long text areas, and reference type fields cannot be used as sort fields. The SOQL query in Listing 4.29 returns records first in ascending order by `Type` and then in descending order by `LastModifiedDate`.

Listing 4.29 SOQL Statement with Sort Fields

```
SELECT Name, Type, AnnualRevenue
FROM Account
ORDER BY Type, LastModifiedDate DESC
```

Querying Multiple Objects

The result of a SOQL query can be a simple list of records containing rows and columns or hierarchies of records containing data from multiple, related objects. Relationships between objects are navigated implicitly from the database structure. This eliminates the work of writing accurate, efficient join conditions common to development on traditional SQL databases.

The two ways to navigate object relationships in SOQL are child-to-parent and parent-to-child. Listing 4.30 is an example of a child-to-parent query, returning the name, city, and Force.com username creating its contact of all resources with a mailing address in the state of California. It selects and filters fields of the Project object, the parent object of Account. It also selects the Name field from the User object, a parent two levels removed from Project via the Account's CreatedBy field.

Listing 4.30 SOQL with Child-to-Parent Relationship

```
SELECT Name, Account__r.Name, Account__r.CreatedBy.Name
  FROM Project__c
  WHERE Account__r.BillingState = 'CA'
```

Caution

The results of child-to-parent relationship queries are not completely rendered in the Force.com IDE. You can double-click a row and column to view fields from a parent record, but this is limited to direct parents only. Fields from parent-of-parent objects, such as the `Contact__r.CreatedBy` relationship in Listing 4.29, are omitted from the results. This is a limitation not of SOQL, but of the Force.com IDE.

At most, five levels of parent objects can be referenced in a single child-to-parent query, and the query cannot reference more than 25 relationships in total.

The second form of relationship query is the parent-to-child query. Listing 4.31 provides an example. The parent object is Resource, and the child is Timecard. The query selects from every Contact its Id, Name, and a list of hours from its Timecards in the current month.

Listing 4.31 SOQL with Parent-to-Child Relationship

```
SELECT Id, Name,
  (SELECT Total_Hours__c
   FROM Timecards__r
   WHERE Week_Ending__c = THIS_MONTH)
  FROM Contact
```

A parent-to-child query cannot reference more than 20 child objects. Double-clicking the parent record in the results table brings up the child records for viewing in the Force.com IDE.

Using SOQL in Apex

Like database objects, SOQL queries are an integrated part of the Apex language. They are developed in-line with your code and verified at compile time against your database schema.

Listing 4.32 is an example of a SOQL query used in Apex. It retrieves a list of Project records for this year and loops over them, summing their billable hours in the variable `totalHours`. Note the usage of the variable named `statuses` directly in the SOQL query, preceded by a colon. This is known as a *bind variable*. Bind variables can appear on the right side of a `WHERE` clause, as the value of an `IN` or `NOT IN` clause, and in the `LIMIT` clause.

Listing 4.32 SOQL Query in Apex

```

Decimal totalHours = 0;
List<String> statuses = new String[] { 'Green', 'Yellow' };
List<Project__c> projects = [ SELECT Billable_Hours__c
    FROM Project__c
    WHERE Start_Date__c = THIS_YEAR and Status__c IN :statuses ];
for (Project__c project : projects) {
    totalHours += project.Billable_Hours__c;
}
System.debug(totalHours);

```

This code relies on a List to store the results of the SOQL query. This means the entire SOQL query result must fit within the heap size available to the program. A better syntax for looping over SOQL records is a variation of the List/Set Iteration For Loop called a SOQL For Loop. The code in Listing 4.33 is a rewrite of Listing 4.32 using the SOQL For Loop. This allows it to run when the Project object contains up to 50,000 records for this year without consuming 50,000 records' worth of heap space at one time.

Listing 4.33 SOQL Query in Apex Using SOQL For Loop

```

Decimal totalHours = 0;
for (Project__c project : [ SELECT Billable_Hours__c
    FROM Project__c
    WHERE Start_Date__c = THIS_YEAR ]) {
    totalHours += project.Billable_Hours__c;
}
System.debug(totalHours);

```

An additional form of the SOQL For Loop is designed for use with Data Manipulation Language (DML). Consider how the code in Listing 4.32 could be adapted to modify Project records returned from the SOQL query rather than simply summing them. With the existing code, one Project record would be modified for each loop iteration, an inefficient approach and a quick way to run afoul of the governor limits. But if you change the type of variable in the For Loop to a list of Project records, Force.com provides up to 200 records per loop iteration. This allows you to modify a whole list of records in a single operation.

Note

Looping through a list of records to calculate the sum of a field is provided as an example of using SOQL with Apex. It is not an optimal way to perform calculations on groups of records in the database. Chapter 5 introduces aggregate queries, which enable calculations to be returned directly from a SOQL query, without Apex.

Any valid SOQL statement can be executed in Apex code, including relationship queries. The result of a child-to-parent query is returned in a List of objects whose types match the child object. Where fields from a parent object are included in the query, they are available as nested variables in Apex code. For example, running the query in Listing 4.30 within a block of Apex code returns a `List<Project__c>`. If this List is assigned to a variable named `projects`, the first Account record's billing state is accessible by `projects[0].Account__r.BillingState`.

Parent-to-child queries are returned in a List of objects, their type matching the parent object. Each record of the parent object includes a nested List of child objects. Using Listing 4.31 as an example, if `results` contains the `List<Contact>` returned by the query, `results[0].Timecards__r[0].Total_Hours__c` accesses a field in the first Contact's first Timecard child record.

Note

Usage of SOQL in Apex is subject to governor limits. For example, you are limited to a total of 100 SOQL queries, or 300 including parent-to-child queries. The cumulative maximum number of records returned by all SOQL queries, including parent-to-child, is 50,000.

Persisting Database Records

Changes to database records in Force.com are saved using Data Manipulation Language (DML) operations. DML operations allow you to modify records one at a time, or more efficiently in batches of multiple records. The five major DML operation types are listed next. Each is discussed in more detail later in this subsection.

- **Insert**—Creates new records.
- **Update**—Updates the values in existing records, identified by Force.com unique identifier (`Id`) field or a custom field designated as an external identifier.
- **Upsert**—If records with the same unique identifier or external identifier exist, this updates their values. Otherwise, it inserts them.
- **Delete**—Moves records into the Recycle Bin.
- **Undelete**—Restores records from the Recycle Bin.

DML operations can be included in Apex code in one of two ways: DML statements and database methods. Beyond the syntax, they differ in how errors are handled. If any one record in a DML statement fails, all records fail and are rolled back. Database methods allow for partial success. This chapter uses DML statements exclusively. Chapter 5 provides information on database methods.

Note

Usage of DML in Apex is subject to governor limits. For example, you are limited to a total of 150 DML operations. The cumulative maximum number of records modified by all DML operations is 10,000.

Insert

The `Insert` statement adds up to 200 records of a single object type to the database. When all records succeed, they contain their new unique identifiers. If any record fails, a `DmlException` is raised and the database is returned to its state prior to the `Insert` statement. For example, the code in Listing 4.34 inserts a `Contact` record and uses it as the parent of a new `Resource` record.

Listing 4.34 **Inserting a Record**

```
try {
    Contact c = new Contact(FirstName = 'Justin', LastName = 'Case',
        Hourly_Cost_Rate__c = 75, Region__c = 'West');
    insert c;
} catch (DmlException e) {
    System.debug(LoggingLevel.ERROR, e.getMessage());
}
```

Update

`Update` saves up to 200 existing records of a single object type. Existing records are identified by unique identifier (Id). Listing 4.35 illustrates the usage of the `Update` statement by creating a `Resource` record for Doug and updating it. Refresh the `Resources` tab in the native user interface to see the new record.

Listing 4.35 **Updating Records**

```
Contact doug = new Contact(FirstName = 'Doug', LastName = 'Hole');
insert doug;
doug.Hourly_Cost_Rate__c = 100;
doug.Home_Office__c = 'London';
update doug;
```

Upsert

`Upsert` combines the behavior of the `Insert` and `Update` operations on up to 200 records of the same object type. First, it attempts to locate a matching record using its unique identifier or external identifier. If one is found, the statement acts as an `Update`. If not, it behaves as an `Insert`.

The syntax of the `Upsert` statement is identical to `Update` and `Insert`, but adds a second, optional argument for specifying an external identifier. If an external identifier is not provided, the record's unique identifier is used. The code in Listing 4.36 upserts a record in the `Contact` object using the field `Resource_ID__c` (created in Chapter 11, "Advanced Integration") as an external identifier. If a `Contact` record with a `Resource_ID__c` value of 1001 exists, it is updated. If not, it is created.

Listing 4.36 Upserting a Record

```
Contact c = new Contact(Resource_ID__c = 1001,
    FirstName = 'Terry', LastName = 'Bull');
upsert c Resource_ID__c;
```

Delete and Undelete

`Delete` and `Undelete` statements move up to 200 records of the same object type to and from the Recycle Bin, respectively. Listing 4.37 shows an example of the `Delete` statement. A new `Resource` record named Terry is added and then deleted.

Listing 4.37 Deleting Records

```
Contact terry = new Contact(FirstName = 'Terry', LastName = 'Bull');
insert terry;
delete terry;
```

Listing 4.38 builds on Listing 4.37 to undelete the Terry record. Concatenate the listings in the `Execute Anonymous` view to test. The database is queried to prove the existence of the undeleted record. Try running the code a second time with the `undelete` statement commented out to see that it is working as intended.

Listing 4.38 Undeleting Records

```
undelete terry;
Contact terry2 = [ SELECT Id, Name
    FROM Contact WHERE Name LIKE 'Terry%' LIMIT 1 ];
System.debug(terry2.Name + ' exists');
delete terry;
```

Database Triggers

Triggers are Apex code working in concert with the Force.com database engine, automatically invoked by Force.com when database records are modified. Trigger code can perform any necessary processing on the modified data before or after Force.com completes its own work. The following list describes scenarios commonly implemented with triggers:

- A validation rule is required that is too complex to define on the database object using formula expressions.
- Two objects must be kept synchronized. When a record in one object is updated, a trigger updates the corresponding record in the other.
- Records of an object must be augmented with values from another object, a complex calculation, or external data via a Web service call.

This subsection covers the essentials of trigger development, including definition, batch processing, and error handling.

Definition

A trigger definition consists of four parts:

1. A unique trigger name to differentiate it from other triggers. Multiple triggers can be defined on the same database object.
2. The name of the database object on which to create the trigger. You can create triggers on standard and custom objects.
3. A comma-separated list of one or more trigger events that cause the trigger code to be executed. An event is specified using two keywords. The first keyword is either *before* or *after*, indicating that the trigger is to be executed before or after the database operation is saved. The second keyword is the DML operation: *insert*, *update*, *delete*, or *undelete*. For example, the trigger event *before update* means that the trigger is fired before a record is updated. Note that *before undelete* is an invalid trigger event.
4. The block of Apex code to execute when the trigger event occurs. The code typically loops over the list of records in the transaction and performs some action based on their contents. For *insert* and *update* triggers, the list of records in the transaction is provided in the variable `Trigger.new`. In a *before* trigger, these records can be modified. In *update*, *delete*, and *undelete* triggers, `Trigger.old` contains a read-only list of the original versions of the records. Also available to your trigger code is a set of Boolean variables indicating the event type that fired the trigger. They are useful when your trigger is defined on multiple events yet requires separate behavior for each. These variables are `Trigger.isBefore`, `Trigger.isAfter`, `Trigger.isInsert`, `Trigger.isUpdate`, `Trigger.isDelete`, and `Trigger.isUndelete`.

Listing 4.39 is an example of a trigger named `validateTimecard`. It is triggered before inserts and updates to the `Timecard` custom object. It doesn't do anything yet because its code block is empty.

Listing 4.39 Trigger Definition

```
trigger validateTimecard on Timecard__c(before insert, before update) {
    // code block
}
```

Triggers cannot be created in the Execute Anonymous view. Create them in the Force.com IDE by selecting File, New, Apex Trigger. To test triggers, use the native user interface to manually modify a relevant record, or write a unit test and invoke it from the Apex Test Runner or Execute Anonymous view.

Tip

A best practice for organizing trigger logic is to place it in an Apex class rather than the body of the trigger itself. This does not change anything about the behavior of the trigger or its governor limits, but encourages code reuse and makes the trigger easier to test.

Batch Processing in Triggers

Manual testing in the native user interface and simplistic unit tests can lull you into the false belief that triggers operate on a single record at a time. Not to be confused with Batch Apex, triggers can always be invoked with a list of records and should be optimized accordingly. Many ways exist to get a batch of records into the Force.com database, including the Data Loader and custom user interfaces. The surest way to a production issue with governor limits is to write a trigger that operates inefficiently when given a batch of records. The process of hardening a trigger to accept a batch of records is commonly called *bulkifying* the trigger.

Batches can be up to 200 records. When writing your trigger code, look at the resources consumed as you loop over `Trigger.new` or `Trigger.old`. Study the governor limits and make sure your code splits its work into batches, doing as little work as possible in the loop. For example, if you have some additional data to query, build a set of IDs from the trigger's records and query them once. Do not execute a SOQL statement for each loop iteration. If you need to run a DML statement, don't put that in the loop either. Create a List of objects and execute a single DML statement on the entire List. Listing 4.40 shows an example of looping over a batch of Contact records (in the variable `contacts`) to produce a list of Assignment records to insert.

Listing 4.40 Batching DML Operations

```
List<Assignment__c> toInsert = new List<Assignment__c>();
for (Contact contact : contacts) {
    toInsert.add(new Assignment__c(
        Contact__r = contact));
}
insert toInsert;
```

Error Handling

Errors are handled in triggers with `try`, `catch` blocks, consistent with other Apex code. But uncaught errors within a trigger differ from other Apex code in how they can impact execution of the larger database transaction the trigger participates in.

A common use of errors in triggers is for validation. Strings describing validation errors can be added to individual records or fields using the `addError` method. Force.com continues to process the batch, collecting any additional errors, and then rolls back the transaction and returns the errors to the initiator of the transaction.

Note

Additional error-handling behavior is available for transactions initiated outside of Force.com; for example, through the SOAP API. Records can fail individually without rolling back the entire transaction. This is discussed in Chapter 10, “Integration with Force.com.”

If an uncaught exception is encountered in a trigger, whether thrown by the system or the trigger code itself, the batch of records is immediately aborted, and all changes are rolled back.

Database Security in Apex

Outside of Anonymous blocks, Apex always runs in a privileged, system context. This gives it access to read and write all data. It does not honor object-, field-, and record-level privileges of the user invoking the code. This works well for triggers, which operate at a low level and need full access to data.

Where full access is not appropriate, Apex provides the `with sharing` keyword. For example, custom user interfaces often require that access to data is limited by the privileges of the current user. Using `with sharing`, the sharing rules applying to the current user are evaluated against the data requested by queries and updated in DML operations. This option is discussed in detail in Chapter 6, “User Interfaces.”

Debugging Apex Using Developer Console

Because Apex code cannot be executed on your local machine, debugging Apex requires some different tools and techniques than traditional software development. This section describes how to debug your code using two features of the Force.com’s Developer Console. Developer Console allows you to set checkpoints to capture a snapshot of the state of your program. It also records execution logs when users perform actions in your application, allowing you to step through the logic and resources consumed.

Checkpoints

Checkpoints allow you to freeze variables at a specific point of execution in your program and examine them later. The point in the code at which the checkpoint is captured is called a checkpoint location. It is similar to a breakpoint in a standard development environment.

To work with checkpoints, open Developer Console and click the Checkpoints tab. To set a checkpoint location, locate the code using the Tests or Repository tab and click to the left of

the desired line. In Figure 4.6, a checkpoint location has been set at line 10, indicated by the dot to the left of the line number.

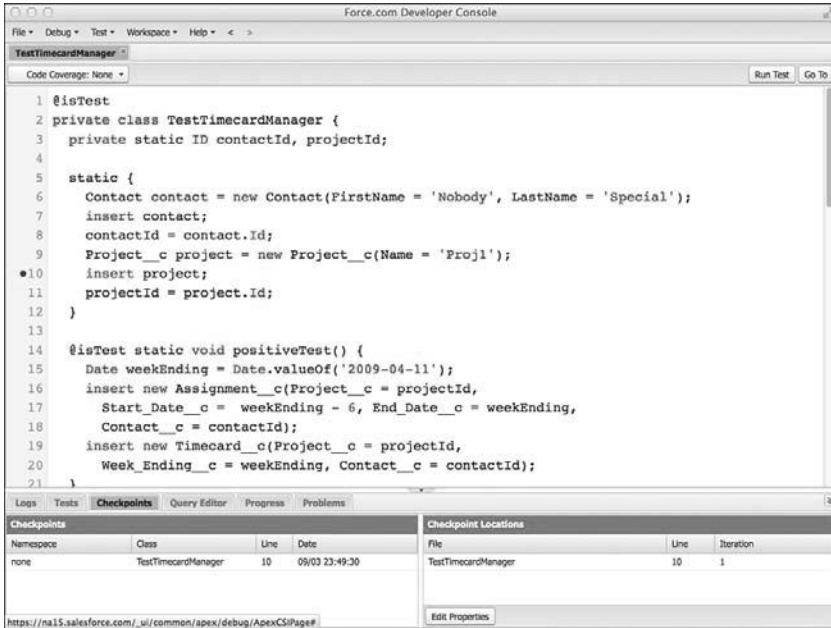


Figure 4.6 Setting a heap dump location

When code is executed at a checkpoint location, a checkpoint is generated. It can be viewed by double-clicking on a row in the Checkpoints tab, as shown in Figure 4.7. A checkpoint has been selected in the Checkpoints tab at the bottom, and its details shown in the top panel. The Symbols tab lists the program's variables and their values at the point in time of the checkpoint.

Execution Logs

Testing or debugging code from a user's point of view, directly from the native user interface, is often necessary. With the Developer Console pop-up window open, you can continue using Force.com in the main browser window. Actions you perform in the application result in execution log entries. Click the Logs tab in Developer Console to examine them.

In Figure 4.8, the user's action has resulted in a log entry, shown in the top table, which is selected and opened by double-clicking it. The top and middle of the screen display the raw execution log on the right panel, and an analysis in the left panels. The Stack Tree, Execution Overview, and Execution Stack provide different views of the Force.com resources consumed and their impact on response time.

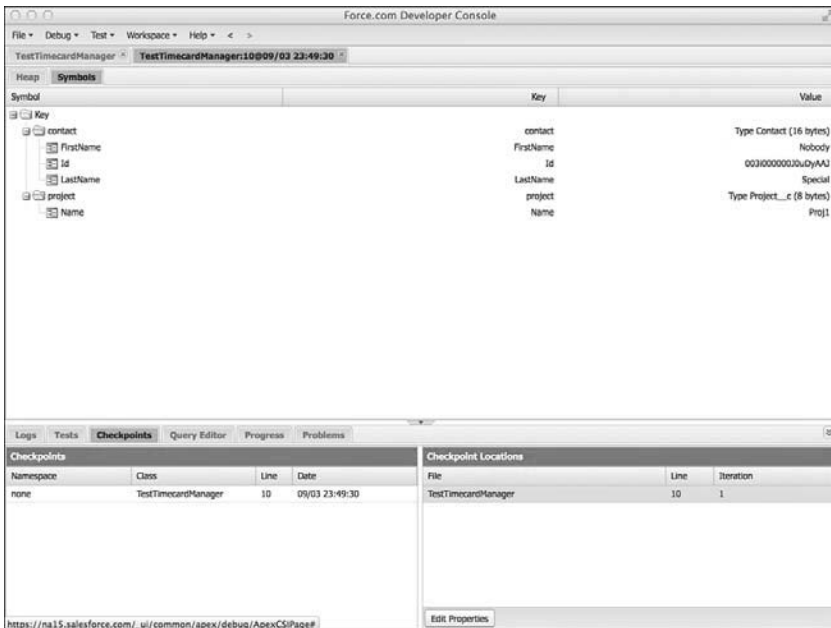


Figure 4.7 Examining a heap dump

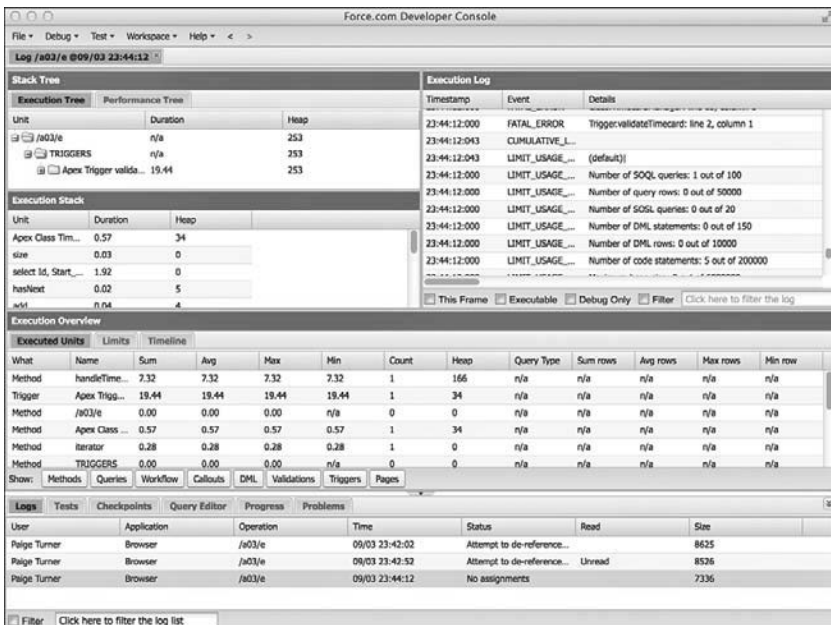


Figure 4.8 Examining the execution log

Unit Tests in Apex

Testing Apex code consists of writing and executing unit tests. Unit tests are special methods written to exercise the functionality of your code. The goal of testing is to write unit tests that execute as many lines as possible of the target code. The number of lines of code executed during a test is called *test coverage* and is expressed as a percentage of the total lines of code. Unit tests also typically perform some pretest preparation, such as creating sample data, and posttest verification of results.

Test Methods

Test methods are static Apex code methods, annotated with `@isTest`. They are written within an outer class, also annotated with `@isTest`. Tests are subject to the same governor limits as all Apex code, but every test method is completely independent for the purposes of limit tracking, not cumulative. Also, test classes are not counted against the code size limit for a Force.com organization.

A test is considered successful if its method is executed without encountering an uncaught exception. A common testing pattern is to make a series of assertions about the target code's state using the built-in method `System.assert`. The argument of `assert` is a Boolean expression. If it evaluates to `true`, the program continues; otherwise, a `System.Exception` is thrown and causes the test to fail.

Listing 4.41 shows a simple test method. It asserts two statements. The second is false, so the test always fails.

Listing 4.41 Test Method

```
@isTest static void negativeTest() {
    Integer i = 2 + 2;
    System.assert(i == 4);
    System.assert(i / 2 == 1);
}
```

Rather than adding two numbers together, most unit tests perform substantial operations in one or more other classes. Sometimes it's necessary to examine the contents of a private variable or invoke a protected method from a test. Rather than relaxing the access modifiers of the code to make them visible to tests, annotate the code you are testing with `@TestVisible`. This annotation provides your test code with privileged access but otherwise preserves the access modifiers in your code.

Test Data

With the exception of users and profiles, tests do not have access to the data in the Force.com database. You can annotate a class or method with `@isTest(SeeAllData=true)` to make the organization's data visible to tests, but this is not a best practice. The recommended approach

is for tests to create their own temporary test data. All database modifications occurring during execution of a test method are automatically rolled back after the method is completed. Create your own test data in a setup phase before your tests are executed, and limit your assertions to that test data.

Running Tests

All tests are automatically executed when migrating code to a production environment, even unchanged and existing tests not included in the migration. Tests can and should be executed manually throughout the development process. Three ways to run tests are described in the following list:

1. The Force.com native user interface includes a test runner. In the App Setup area, click Develop, Apex Classes, and then click the Run All Tests button.
 2. In the Force.com IDE, right-click an Apex class containing test methods and select Force.com, Run Tests.
 3. From Developer Console, click the Tests tab and the New Run button. Select the tests to include, and click the Run button. Alternatively, right-click on the Classes folder in Eclipse and select Force.com, Run Tests to execute all tests in your organization.
- Figure 4.9 shows Developer Console after running a test.

The screenshot shows the Force.com Developer Console interface. At the top, the title bar reads "Force.com Developer Console". Below it, the "TestTimecardManager" class is open, showing its source code. The code includes a test method and a private class with static variables and methods for creating test data.

Below the code editor, the "Tests" tab is active, displaying a table of test results. The table has columns for Status, Test Run, Duration, Failures, Total, and Overall Code Coverage. The test results show that all tests passed successfully.

| Status | Test Run | Duration | Failures | Total | Overall Code Coverage |
|--------|--|----------|----------|-------|-----------------------|
| ✓ | 2013-09-03 23:32:47, paigetturner.d.3e@gmail.com | | 4 | 4 | |
| ✓ | 2013-09-03 23:36:34, paigetturner.d.3e@gmail.com | | 0 | 4 | |
| ✓ | TestTimecardManager | | 0 | 4 | |
| ✓ | testNoValidAssignments | 0:00 | | | |
| ✓ | testNoAssignments | 0:00 | | | |
| ✓ | testBatch | 0:02 | | | |
| ✓ | positiveTest | 0:02 | | | |

On the right side of the console, there is a table for "Overall Code Coverage":

| Class | Percent | Lines |
|--|---------|---------|
| AppController | 0% | No data |
| AppUtil | 0% | No data |
| BaseConfig | 0% | No data |
| BaseConfigController | 0% | No data |
| BenchmarkWS | 0% | No data |
| ChangePasswordController | 0% | No data |
| ChangePasswordControllerTest | 0% | No data |
| CompareSkillsController | 0% | No data |
| ContentController | 0% | No data |
| DetailController | 0% | No data |
| FollowProjectControllerExtension | 0% | No data |
| FooterController | 0% | No data |
| ForceStyledComponentsControllerExtension | 0% | No data |

Figure 4.9 Viewing test results in Developer Console

Sample Application: Validating Timecards

This section applies Apex, SOQL, DML, and triggers to ensure that timecards entered into the Services Manager sample application have a valid assignment. An *assignment* is a record indicating that a resource is staffed on a project for a certain time period. A consultant can enter a timecard only for a project and time period he or she is authorized to work. Triggers are one way to enforce this rule.

The following subsections cover the process of configuring the Force.com IDE for Apex development, creating the trigger code to implement the timecard validation rule, and writing and running unit tests.

Force.com IDE Setup

Begin by creating the Force.com IDE Project for the Services Manager sample application, if you have not already done so. Select the menu option File, New, Force.com Project. Enter a project name, username, password, and security token of your Development Edition organization and click the Next button and then the Finish button. The Force.com IDE connects to Force.com, downloads the metadata in your organization to your local machine, and displays a new project node in your Navigator view.

Creating the Trigger

Listing 4.42 defines the trigger to validate timecards. It illustrates a best practice for trigger development: Keep the trigger's code block as small as possible. Place code in a separate class for easier maintenance and to encourage code reuse. Use naming conventions to indicate that the code is invoked from a trigger, such as the `Manager` suffix on the class name and the `handle` prefix on the method name.

Listing 4.42 **Trigger** `validateTimecard`

```
trigger validateTimecard on Timecard__c(before insert, before update) {
    TimecardManager.handleTimecardChange(Trigger.old, Trigger.new);
}
```

To create this trigger, select File, New, Apex Trigger. Enter the trigger name, select the object (`Timecard__c`), enable the two trigger operations (`before insert`, `before update`), and click the Finish button. This creates the trigger declaration and adds it to your project. It is now ready to be filled with the Apex code in Listing 4.42. If you save the trigger now, it will fail with a compilation error. This is because the dependent class, `TimecardManager`, has not yet been defined.

Continue on to creating the class. Select File, New, Apex Class to reveal the New Apex Class Wizard. Enter the class name (`TimecardManager`), leave the other fields (Version and Template) set to their defaults, and click the Finish button.

Listing 4.43 is the `TimecardManager` class. It performs the work of validating the timecard on behalf of the trigger. First, it builds a Set of resource Ids referenced in the incoming set of timecards. It uses this Set to query the Assignment object. For each timecard, the assignment List is looped over to look for a match on the time period specified in the timecard. If none is found, an error is added to the offending timecard. This error is ultimately reported to the user or program initiating the timecard transaction.

Listing 4.43 `TimecardManager` Class

```
public with sharing class TimecardManager {
    public class TimecardException extends Exception {}
    public static void handleTimecardChange(List<Timecard__c> oldTimecards,
        List<Timecard__c> newTimecards) {
        Set<ID> contactIds = new Set<ID>();
        for (Timecard__c timecard : newTimecards) {
            contactIds.add(timecard.Contact__c);
        }
        List<Assignment__c> assignments = [ select Id, Start_Date__c,
            End_Date__c, Contact__c from Assignment__c
            where Contact__c in :contactIds ];
        if (assignments.size() == 0) {
            throw new TimecardException('No assignments!');
        }
        Boolean hasAssignment;
        for (Timecard__c timecard : newTimecards) {
            hasAssignment = false;
            for (Assignment__c assignment : assignments) {
                if (assignment.Contact__c == timecard.Contact__c &&
                    timecard.Week_Ending__c - 6 >= assignment.Start_Date__c &&
                    timecard.Week_Ending__c <= assignment.End_Date__c) {
                    hasAssignment = true;
                    break;
                }
            }
            if (!hasAssignment) {
                timecard.addError('No assignment for contact ' +
                    timecard.Contact__c + ', week ending ' +
                    timecard.Week_Ending__c);
            }
        }
    }
}
```

Unit Testing

Now that the trigger is developed, you must test it. During development, taking note of the code paths and thinking about how they are best covered by unit tests is a good idea. An even better idea is to write the unit tests as you develop.

To create unit tests for the timecard validation code using the Force.com IDE, follow the same procedure as that for creating an ordinary Apex class. An optional variation on this process is to select the Test Class template from the Create New Apex Class Wizard. This generates skeleton code for a class containing only test methods.

Listing 4.44 contains unit tests for the `TimecardManager` class. Before each unit test, test data is inserted in a static initializer. The tests cover a simple positive case, a negative case in which no assignments exist for the timecard, a second negative case in which no valid assignments exist for the time period in a timecard, and a batch insert of timecards. The code demonstrates a best practice of placing all unit tests for a class in a separate test class with an intuitive, consistent naming convention. In our example, the `TimecardManager` class has a test class named `TestTimecardManager`, the class name prefaced by the word *Test*.

Listing 4.44 Unit Tests for TimecardManager Class

```
@isTest
private class TestTimecardManager {
    private static ID contactId, projectId;

    static {
        Contact contact = new Contact(FirstName = 'Nobody', LastName = 'Special');
        insert contact;
        contactId = contact.Id;
        Project__c project = new Project__c(Name = 'Proj1');
        insert project;
        projectId = project.Id;
    }

    @isTest static void positiveTest() {
        Date weekEnding = Date.valueOf('2015-04-11');
        insert new Assignment__c(Project__c = projectId,
            Start_Date__c = weekEnding - 6, End_Date__c = weekEnding,
            Contact__c = contactId);
        insert new Timecard__c(Project__c = projectId,
            Week_Ending__c = weekEnding, Contact__c = contactId);
    }

    @isTest static void testNoAssignments() {
        Timecard__c timecard = new Timecard__c(Project__c = projectId,
            Week_Ending__c = Date.valueOf('2015-04-11'),
            Contact__c = contactId);
```

```

try {
    insert timecard;
} catch (DmlException e) {
    System.assert(e.getMessage().indexOf('No assignments') > 0);
    return;
}
System.assert(false);
}

@isTest static void testNoValidAssignments() {
    Date weekEnding = Date.valueOf('2015-04-04');
    insert new Assignment__c(Project__c = projectId,
        Start_Date__c = weekEnding - 6, End_Date__c = weekEnding,
        Contact__c = contactId);
    try {
        insert new Timecard__c(Project__c = projectId,
            Week_Ending__c = Date.today(), Contact__c = contactId);
    } catch (DmlException e) {
        System.assert(e.getMessage().indexOf('No assignment for contact') > 0);
        return;
    }
    System.assert(false);
}

@isTest static void testBatch() {
    Date weekEnding = Date.valueOf('2015-04-11');
    insert new Assignment__c(Project__c = projectId,
        Start_Date__c = weekEnding - 6, End_Date__c = weekEnding,
        Contact__c = contactId);
    List<Timecard__c> timecards = new List<Timecard__c>();
    for (Integer i=0; i<200; i++) {
        timecards.add(new Timecard__c(Project__c = projectId,
            Week_Ending__c = weekEnding, Contact__c = contactId));
    }
    insert timecards;
}
}

```

After saving the code in the unit test class, run it by right-clicking in the editor and selecting Force.com, Run Tests. After a few seconds, you should see the Apex Test Runner view with a green check box indicating that all tests passed, as shown in Figure 4.10. Expand the results node to see 100% test coverage of the `TimecardManager`, and scroll through the debug log to examine performance information and resource consumption for each of the tests.

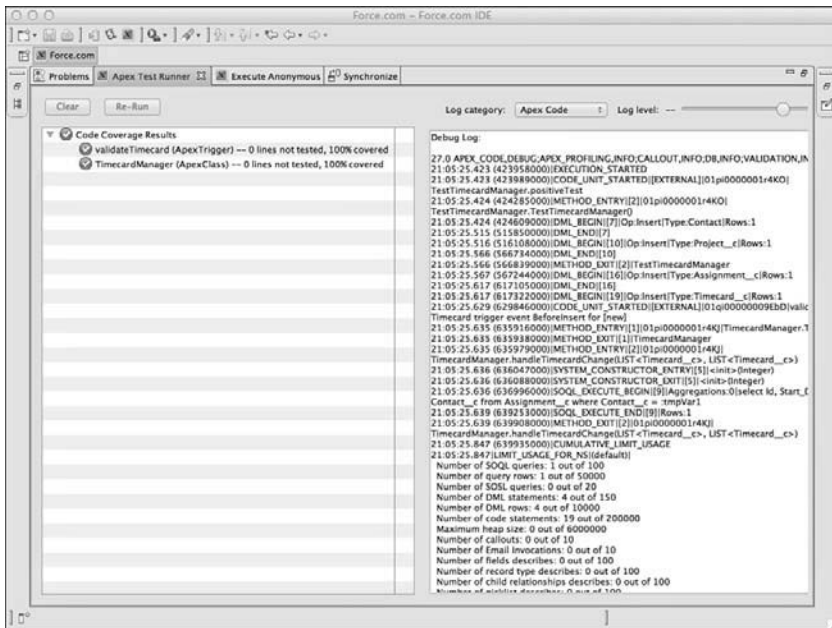


Figure 4.10 Viewing test results

Summary

This chapter is arguably the most important chapter in the book. It describes core Apex concepts and syntax that form the basis of all subsequent chapters. Absorb this chapter, augmenting it with the information available through the developer.force.com Web site and community, and you will be well prepared to write your own Force.com applications.

Before moving on, take a few minutes to review these major areas:

- Apex is the only language that runs inside the Force.com platform and is tightly integrated with the Force.com database. Apex is strongly typed and includes object-oriented features.
- The Force.com database is queried using SOQL and SOSL, and its records are modified using DML. All three languages can be embedded directly inside Apex code.
- Resources consumed by Apex programs are tightly controlled by the Force.com platform through governor limits. Limits vary based on the type of resource consumed. Learn the relevant governor limits as early as possible in your development process. This ensures that you write efficient code that scales up to production data volumes.

Index

Symbols

+ (addition) operator, 110
& (AND) operator, 110
&& (AND) operator, 110
- (arithmetic negation) operator, 110
= (assignment) operator, 110
\ (backslash), UNIX line-continuation character, 309
/ (division) operator, 110
== (equality) operator, 110
=== (exact equality) operator, 110
!== (exact inequality) operator, 110
> (greater than) operator, 110
>= (greater than or equal to) operator, 110
() (grouping operators), 110
?: (if/then/else expression shortcut), 110
< (less than) operator, 110
<= (less than or equal to) operator, 110
! (logical negation) operator, 110
* (multiplication) operator, 110
!= (not equal to) operator, 110
| (OR) operator, 110
|| (OR) operator, 110
<< (signed shift left) operator, 110
>> (signed shift right) operator, 110
+ (string concatenation) operator, 110
- (subtraction) operator, 110
- (unary decrement) operator, 110
++ (unary increment) operator, 110
>>> (unsigned shift right) operator, 110
^ (XOR) operator, 110
4GL developer contributions, 12

A

abortJob method, 296
Accept button, 213
accessibility (fields), 78-79, 89-90

accessing data

mobile Web applications
 actionFunction component, 270
 authentication, 269-270
 JavaScript remoting, 270
 REST API, 270
 SmartSync, 270
 REST API, 306
AccessLevel field, 163
access modifiers, 118
accounts receivable profile, 18, 86
actionFunction component, 235-236
 mobile Web application data access, 270
 Visualforce
 controller, 236
 page code, 236
actionPoller component, 237
actions, 203-204
 asynchronous
 as JavaScript events, 237-238
 as JavaScript functions, 235-236
 partial page refreshes, 234-235
 status messages, 238-240
 as timed events, 237
 container components, 205
 custom controllers, 195-197
 custom logic, invoking, 195
 trigger page navigation, 195
 view state preservation, 195
 wrapper pattern, 195-196
 expressions
 standard controllers, 192
 standard set controllers, 193
actionStatus component, 238-240
actionSupport component, 237-238, 262
addError method, 225
addField method, 246
addInfo method, 225
addition (+) operator, 110

administrative permissions, 75

aggregate functions, 144-145

- AVG, 144
- COUNT, 144-145
- COUNT_DISTINCT, 144
- governor limits, 145
- MAX, 144
- MIN, 144
- records, grouping, 146
- SUM, 144-145

AggregateResult object, 145

aggregate SOQL queries, 144

- aggregate functions, 144-145
 - AVG, 144
 - COUNT, 144-145
 - COUNT_DISTINCT, 144
 - governor limits, 145
 - MAX, 144
 - MIN, 144
 - records, grouping, 146
 - SUM, 144-145
- grouping records, 145-146
 - with aggregate functions, 146
 - filtering grouped, 146
 - without aggregate functions, 145-146
- grouping records with subtotals, 147-148
 - debug log excerpt, 147
 - GROUP BY CUBE clause, 147-148
 - GROUP BY ROLLUP clause, 147

Ajax (Asynchronous JavaScript and XML)

- actions, 234
 - as JavaScript events, 237-238
 - as JavaScript functions, 235-236
 - partial page refreshes, 234-235
 - status messages, 238-240
 - as timed events, 237
 - Visualforce support, 234

Proxy, 270

Amazon Web Services, 2-3

AND (&) operator, 110

AND (&&) operator, 110

AngularJS, 251-253

- controllers
 - ProjectListCtrl, 253
 - timecard editing, 279
 - Visualforce, implementing, 252

- demonstration page, 251
- templates, 253
- timecard entry in-page navigation controller, 277
- tutorial Web site, 251
- Visualforce page code, 252-253
- Web site, 251

anonymous benchmarking SOAP Web service, 333-335

anti-joins

- overview, 152
- restrictions, 153

Apex, 7

- AggregateResult object, 145
- aggregate SOQL queries, 144
 - aggregate functions, 144-145
 - grouping records, 145-146
 - grouping records with subtotals, 147-148

arrays

- creating, 111
- initializing, 111-112
- sorting, 112

Batch

- Batchable interface, 283-284
- batch jobs, 282, 286-289
- classes, creating, 285-286
- iterable scope, 290-292
- limitations, 292
- missing timecard class, developing, 298-299
- project evaluation guidelines, 284-285
- scheduling, 293-296
- scope, 282
- stateful, 289-290
- testing, 293
- transactions, 283

callouts, 301-302

- REST services, 302-304
- SOAP services, 305-306

Chatter, 378-379

- Visualforce controller, 378
- Visualforce page, 378
- Web site, 378

- classes
 - ConnectApi, 378
 - custom Apex REST services, creating, 312-314
 - custom Apex SOAP Web services
 - rules, 327
 - HTTP, 302-303
- code deployment in Tooling API, 355
- Code Developer's Guide Web site, 108
- code execution
 - asynchronous, 116
 - conditional statements, 113
 - Execute Anonymous View, 104-105
 - exception statements, 114-115
 - governor limits, 120
 - loops, 114
 - recursion, 115
- collections
 - clearing, 109
 - cloning, 109
 - emptiness, 109
 - size, 109
- custom Apex REST Web services, 312-314
 - Apex class rules, 312
 - creating, 313
 - governor limits, 312
 - invoking, 313-314
- custom settings, 180-181
 - creating, 180
 - deleting, 180
 - governor limits, 180
 - hierarchy type, 181
 - updating, 180
 - values, retrieving, 180
- custom SOAP Web services, 326
 - Apex class rules, compared, 327
 - calling, 328
 - creating records example, 328
 - governor limits, 327
 - invoking, 329
 - limitations, 326-327
 - Services Manager anonymous
 - benchmarking, 333-335
- database integration
 - data integrity, 122
 - DML statements. *See* DML, statements
 - objects, referencing, 121-122
 - overview, 120-121
 - queries. *See* queries
 - security, 133
- data types, 106
 - Blob, 106
 - Boolean, 106
 - converting, 107-108
 - Date, 106
 - date to string conversions, 109
 - Datetime, 106
 - Decimal, 106
 - Double, 106
 - ID, 106
 - Integer, 106
 - Long, 106
 - Object, 106
 - String, 106
 - string to date conversions, 109
 - Time, 106
- debugging, 133
 - checkpoints, 133-135
 - execution logs, 134
- dynamic, 174
 - instances, creating, 179
 - schema metadata, 177-179
 - SOQL queries, 175-176
 - SOSL queries, 176
- governor limits, 100, 120
 - Apex code, 120
 - databases, 120
 - heaps, 120
 - namespaces, 120
- lists
 - creating, 111
 - initializing, 111-112
 - nesting, 111
 - overview, 111
 - sorting, 112
- managed sharing, 162
 - organization-wide sharing defaults,
 - changing, 163
 - rules, creating, 163-167
 - sharing objects, 162-163
- maps, 112-113
- object-oriented principles, 117
 - encapsulation, 117-118
 - information-hiding notation, 118

- inheritance, 119
 - modularity, 119
 - polymorphism, 119
 - operators, 109
 - AND (&&), 110
 - addition (+), 110
 - arithmetic negation (-), 110
 - assignment (=), 110
 - bitwise, 110
 - division (/), 110
 - equality (==), 110
 - exact equality (===), 110
 - exact inequality (!==), 110
 - greater than (>), 110
 - greater than or equal to (>=), 110
 - grouping, 110
 - if/then/else expression (? :), 110
 - less than (<), 110
 - less than or equal to (<=), 110
 - logical negation (!), 110
 - multiplication (*), 110
 - not equal to (!=), 110
 - OR (||), 110
 - signed shift left (<<), 110
 - signed shift right (>>), 110
 - string concatenation (+), 110
 - subtraction (-), 110
 - unary decrement (--), 110
 - unary increment (++), 110
 - unsigned shift right (>>>), 110
 - ORM code snippet, 30
 - overview, 100-101
 - receiving email, 172-173
 - class, creating, 173-174
 - governor limits, 173
 - personalizing based on sender identity, 173
 - services, configuring, 174-175
 - uncaught exceptions, 173
 - sending email, 168
 - attachments, 172
 - blind-carbon-copies, 171
 - carbon copies, 171
 - mass emails, 170-171
 - notifications, 181-182
 - organization-wide email address
 - unique identifiers, 172
 - reply-to addresses, 171
 - sendEmail method, 171
 - sender display names, 171
 - signatures, 172
 - SingleEmailMessage object, 168-169
 - templates, 169-170
 - tracking, 172
 - sets, 112
 - SOQL queries, 126-128
 - SOSL, 155-157
 - Test Runner View (IDE), 103
 - transaction processing
 - DML database methods, 157-158
 - record locking, 161
 - savepoints, 159-160
 - triggers, 130-131
 - batching, 132
 - bulkifying, 132
 - definitions, 131-132
 - error handling, 132-133
 - names, 131
 - timecard validation, creating, 138-139
 - unit tests, 136
 - results, viewing, 137
 - running, 137
 - test data, 137
 - test methods, 136
 - Test Runner View, 103
 - TimecardManager class, 140-141
 - variables, 105
 - access modifiers, 118
 - checkpoints, 133-135
 - classes, 117
 - constants, 107
 - declaring, 105-106
 - enums, 107
 - names, 105-106
 - rounding, 108
- APIs**
- Bulk, 344
 - authentication, 345-346
 - exporting records, 347-349
 - importing records, 346-347
 - two-tier system, 345
 - Web site, 345
 - Canvas, 349
 - authentication, 349-350
 - cross-domain XHR, 350

- example application, 350-354
 - Web site, 349
- Metadata, 360
 - object creation example, 361-363
 - services, 360-361
 - Web site, 360
- REST
 - authentication, 306-307
 - Chatter, 379-380
 - Connected Apps, creating, 307
 - creating record requests, 310
 - data access, 306
 - data integration, 31
 - deleting record requests, 311
 - Force.com REST API Developer's Guide Web site, 308
 - mobile Web application data access, 270
 - record retrieval by external identifiers, 310
 - record retrieval by unique identifiers, 309
 - services available call, 308-309
 - SObject basic information request, 309
 - SOQL query request, 310
 - updating record requests, 311
 - upserting record requests, 311
- SOAP
 - data integration, 31
 - enabled permissions, 318-319
 - Enterprise. *See* Enterprise API
 - error handling, 321
 - Force.com data types, 321
 - IP white-listing, 319
 - limits, 316
 - logging in/out, 318-320
 - login call, 320
 - login problems, troubleshooting, 320
 - Partner, 315
 - security, 316
 - security tokens, 319
 - stub code, generating, 316-317
 - Web Service Connector (WSC), 316
 - WSDL versions, 315-316
- Streaming
 - example, 341-344
 - PushTopics, 340-341
 - Web site, 340
- Tooling, 354
 - Apex code, deploying, 355
 - internal state of deployment, 355
 - overview, 355
 - query service, 355
 - status, refreshing, 355
 - user interface, 356
 - Visualforce controller example, 357-359
 - Visualforce page example, 359-360
 - Web site, 355
- App Builder Tools, 33**
- App Engine, 3**
- AppExchange, 16**
- applications**
 - AppExchange, 16
 - Connected Apps, creating, 351
 - custom, creating, 58
 - LDV deployments, 22
 - mobile
 - Chatter Mobile, 264
 - containers, 271
 - hybrid, 265, 267
 - native, 265-266
 - Salesforce Classic, 264
 - Salesforce Mobile SDK, 265
 - Salesforce Touch, 264
 - timecard entry page. *See* mobile timecard entry page
 - Web. *See* mobile applications, Web services, 6
 - Services Manager. *See* Services Manager application
 - single-page, 250
 - AngularJS, 251-253
 - JavaScript remoting, 250
 - social. *See* Chatter
- architectures**
 - application services, 6
 - declarative metadata, 7
 - multitenancy, 4-6
 - programming languages, 7
 - relational databases, 6
 - security, 71
 - Visualforce, 186-187

arithmetic negation (-) operator, 110**arrays**

- creating, 111
- initializing, 111-112
- sorting, 112

Assignment object

- fields, 54
- overview, 53

assignment (=) operator, 110**asynchronous actions**

- as JavaScript events, 237-238
- as JavaScript functions, 235-236
- partial page refreshes, 234-235
- status messages, 238-240
 - actionStatus component, 238-240
 - dynamic, 239
 - images/stylized messages, 239
- as timed events, 237

asynchronous code execution, 116**Asynchronous JavaScript and XML. See Ajax****asyncMethod, 116****attachments (email), 172****attributes**

- page components, 200
- reRender, 234
- showChatter, 381
- view components, 199

authentication

- Bulk API, 345-346
- Canvas, 349-350
- mobile Web applications, 269-270
- REST APIs, 306-307
- sites users, 258

auto numbers, 40-41, 322**availability (PushTopics), 341****AVG aggregate function, 144**

B

backslash (\), UNIX line-continuation**character, 309****Batchable interface, 283-284****Batch Apex, 116**

- Batchable interface, 283-284
- batch jobs, 282
 - executing, 286
 - execution detail, viewing, 288
 - progress, monitoring, 287-288
 - scope, 289

classes, creating, 285-286**iterable scope, 290-292****limitations, 292****missing timecard class, developing, 298-299****project evaluation guidelines, 284-285****scheduled jobs**

- creating, 295
- editing, 296
- viewing, 296

scheduling, 293-296

- Apex user interface, 294-295
- sample code, 296
- schedulable code development, 294

scope, 282**stateful, 289-290****testing, 293****transactions, 283****batch jobs, 282****bulk export**

- batches, creating, 348
- creating, 347-348
- results, retrieving, 348-349
- status, checking, 348

bulk import

- closing, 347
- creating, 346
- records, adding, 346-347
- results, retrieving, 347
- status, checking, 347

executing, 286**execution detail, viewing, 288****limitations, 292****progress, monitoring, 287-288****scheduled**

- creating, 295
- deleting, 296
- editing, 296

scheduling, 293-296

- Apex user interface, 294-295
- schedulable code development, 294

scope, 289**triggers, 132****BenchmarkWS class, 334****binary data types, 322****bitwise operators, 110****blind-carbon-copies (email), 171****blobs, 106**

Boolean data type, 106

break keyword (loops), 114

browsing data, 42-44

Bulk API, 344

authentication, 345-346

records

exporting, 347-349

importing, 346-347

two-tier system, 345

Web site, 345

bulk jobs

export

batches, creating, 348

creating, 347-348

results, retrieving, 348-349

status, checking, 348

import

closing, 347

creating, 346

records, adding, 346-347

results, retrieving, 347

status, checking, 347

bulk modifications (records), 326

business analyst contributions, 11

business units

collaboration, testing, 97-98

security, 85-88

buttons

custom

custom objects, creating, 38

Visualforce pages, 215

native user interface, 213

standard, 37

C

callouts (Apex), 301-302

REST services, 302-304

formats, 302

HTTP classes, 302-303

integrating, 303-304

invoking, 303

testing, 304

SOAP services, 305-306

Canvas, 349

authentication, 349-350

cross-domain XHR, 350

example application

adding pages, 352

callback HTML page, 354

configuring pages, 352

Connected App, creating, 351

local Web servers, configuring, 352

main HTML page, 353-354

previewing, 352

running in App Previewer, 350

Web site, 349

carbon copies (email), 171

catch keyword (exceptions), 115

channel names, 340

Chatter

Apex, 378-379

Visualforce controller example, 378

Visualforce page, 378

Web site, 378

comments, 374-375

creating, 375

deleting, 375

query, 375

schema pattern, 374

feed-tracked changes, 376

following records, 376-378

following relationships, 377

method, 377

unfollowing, 377-378

Mobile, 264

objects

dynamic, 370

high-volume design, 370

relationship-rich, 370

posts, 370-372

content, 371

creating, 372-373

custom object feeds, 373

deleting, 373

Feed objects, 370-371

news feeds, 374

schema pattern, 370

standard object feeds, 372-373

user feeds, 374

REST API, 379-380

followed records request, 380

news feed request, 379-380

post request, 380

Web site, 379

- Services Manager Follow Team button, 382-385
 - configuring, 385
 - controller extension code, 383-384
 - testing, 385
 - Visualforce page, 384-385
- Visualforce components, 380-382
 - feed, 381
 - feedWithFollowers, 381
 - follow, 381
 - followers, 381
 - limitations, 382
 - newsFeed, 381
 - showChatter attribute, 381
 - userPhotoUpload, 381
 - Visualforce page, creating, 381
- checkboxes**
 - defined, 38
 - SOAP type, mapping, 322
- checkpoints, 133-135**
- child relationships**
 - child-to-parent, 125-126
 - metadata, 178
 - semi-joins
 - child-to-child, 153
 - child-to-parent, 153
- classes**
 - access modifiers, 118
 - Apex
 - ConnectApi, 378
 - custom Apex REST services, creating, 312-314
 - custom Apex SOAP Web services
 - rules, 327
 - HTTP, 302-303
 - Batch Apex, creating, 285-286
 - BenchmarkWS, 334
 - constructors, 118
 - defining, 118
 - information-hiding notation, 118
 - inheritance, 119
 - initializers, 118
 - inner, 118
 - Iterable, 291
 - Iterator, 290
 - methods, 117
 - MissingTimecardBatch
 - creating, 298-299
 - reset results, 300
 - running, 300
 - MyEmailService, 173-174
 - properties, 117
 - TimecardManager
 - creating, 138-139
 - unit tests, 140-141
 - variables, 117
- clear method (collections), 109**
- Clone button, 213**
- clone method (collections), 109**
- closing bulk import jobs, 347**
- cloud computing**
 - benefits, 2
 - overview, 2
 - PaaS, 2
 - Amazon Web Services, 2-3
 - Force.com, 3-4
 - Google Cloud Platform, 3
 - Windows Azure, 3
- Cloudforce conference, 17**
- code execution (Apex)**
 - asynchronous, 116
 - conditional statements, 113
 - exception statements, 114-115
 - examples, 115
 - handling, 115
 - raising, 115
 - governor limits, 120
 - loops, 114
 - recursion, 115
- Code Share, 16**
- collections**
 - arrays
 - creating, 111
 - initializing, 111
 - sorting, 112
 - clearing, 109
 - cloning, 109
 - emptiness, 109
 - lists
 - creating, 111
 - initializing, 111
 - nesting, 111
 - sorting, 112

- maps, 112-113
 - sets, 112
 - size, 109
- ComeD library, 342**
- commandButton component, 203**
- commandLink component, 203**
- comments (Chatter), 374-375**
 - creating, 375
 - deleting, 375
 - query, 375
 - schema pattern, 374
- communication errors, 220-221**
- CompareSkillsComponent**
 - creating, 259-260
 - support, adding, 261
- CompareSkillsController, 260**
- composition (modular Visualforce pages), 243-244**
- conditional statements, 113**
- condition expressions, 194**
- configuration management, 14**
- configuring**
 - Canvas App pages, 352
 - email services, 174-175
 - field accessibility, 89-90
 - Follow Team button, 385
 - IDE, 138
 - local Web servers, 352
 - sharing rules, 92-93
- ConnectApi classes, 378**
- Connected Apps, creating**
 - Canvas, 351
 - REST API, 307
- constants, 107**
- constructors, 118**
- Consultant profile**
 - permissions, 86
 - Services Manager application, 18
 - testing, 96
- ContactFeed object, 372**
- Contact object**
 - CSV import file, 69
 - fields, 51
 - overview, 51
- ContainerId field, 355**

- containers**
 - dynamicComponent elements, 248
 - mobile applications, 271
 - static resources, 241-242
- continue keyword (loops), 114**
- controlled by parent records, 81**
- controller attribute (pages), 200**
- controllers, 186-187**
 - actionFunction component, 236
 - actions
 - as JavaScript events, 237-238
 - timed events, 237
 - AngularJS, 253
 - mobile timecards, editing, 279
 - project list example, 252
 - timecard entry in-page navigation, 277
 - Chatter example, 378
 - custom, 193-197
 - actions, 195-197
 - exposing data, 193-194
 - dynamic field reference, 247
 - extensions, 197
 - governor limits, 221
 - mobile timecards
 - editing, 277
 - list functionality, 274
 - partial page refresh, 235
 - Services Manager
 - business hours, configuring, 331
 - Follow Team button extension code, 383-384
 - Skills Matrix, 225-227, 229-231
 - utilization calculation, 332
 - utilization code, 335-337
 - standard, 191-193
 - multiple records, 192-193
 - single records, 191-192
 - Streaming API example, 342
 - Tooling API example, 357-359
 - unit tests, 222
- conversion methods, 108**
- converting data types, 107-108**
 - conversion methods, 108
 - dates to strings, 109
 - exceptions, 114
 - implicit conversion, 107-108
 - strings to dates, 109

- COUNT aggregate function, 144-145**
- COUNT_DISTINCT aggregate function, 144**
- Create Lookup Field dialog box, 61**
- Create New Object dialog box, 59**
- create permission, 75**
- createProject service, 329**
- create service, 324**
- cross-domain XHR, 350**
- CRUD (create, read, update, delete) operations, 31**
- Crypto class, 303**
- CSRF (Cross Site Request Forgery) attacks, 385**
- CSS (components), adding, 261**
- CSV files**
 - Contact import, 69
 - exporting, 64-65
 - Project import, 65
- cURL, 306**
- currency**
 - fields, 38
 - SOAP data type, mapping, 322
- custom Apex Web services**
 - REST, 312-314
 - Apex class rules, 312
 - creating, 313
 - governor limits, 312
 - invoking, 313-314
 - SOAP, 326
 - Apex class rules, compared, 327
 - calling, 328
 - creating records example, 328
 - governor limits, 327
 - invoking, 329
 - limitations, 326-327
 - Services Manager anonymous benchmarking, 333-335
- custom applications, creating, 58**
- custom buttons**
 - custom objects, creating, 38
 - Visualforce pages, 215
- custom components**
 - creating, 259-260
 - CSS, adding, 261
 - defining, 244-245
 - Google Map example, 245-246
 - support, adding, 259-260
- custom controllers, 193-197**
 - actions, 195-197
 - custom logic, invoking, 195
 - trigger page navigation, 195
 - view state preservation, 195
 - wrapper pattern, 195-196
 - exposing data, 193-194
- custom fields. See fields, creating**
- custom links**
 - custom objects, creating, 38
 - Visualforce pages, 215
- custom objects, 22**
 - creating, 35, 59-60
 - activities, allowing, 36
 - custom buttons/links, 38
 - custom fields, 37
 - definition, 35-36
 - deployment status, 36
 - descriptions, 36
 - field history tracking, 36
 - help settings, 36
 - labels, 35
 - names, 35
 - page layouts, 37
 - record name label, 36
 - reports, allowing, 36
 - search layouts, 37
 - standard buttons/links, 37
 - standard fields, 36
 - triggers, 37
 - validation rules, 37
 - missing timecards, creating, 297
 - tabs, creating, 63
 - tools, 33-34
 - App Builder Tools, 33
 - data, 34
 - Force.com IDE, 34
 - metadata, 33
 - Schema Builder, 34
- custom settings, 180-181**
 - defined, 47
 - governor limits, 180
 - hierarchy, 49, 181
 - list, 48

- records
 - creating, 180
 - deleting, 180
 - updating, 180
- storage limits, 49
- types, 47-48
- values, retrieving, 180

custom tabs, 215

D

data

- batch processing. *See* Batch Apex
- browsing, 42-44
- entering, 41-42
- exposing (custom controllers), 193-194
- expressions
 - standard controllers, 192
 - standard set controllers, 193
- importing, 64
 - import process, 66
 - preparations, 64-66
 - verification, 67-69
- integration, 29
 - metadata XML, 30-31
 - native user interface, 31
 - object-relational mapping, 30
 - REST APIs, 31
 - SOAP APIs, 31
- integrity, 122
- mobile Web applications access, 269-270
 - actionFunction component, 270
 - authentication, 269-270
 - JavaScript remoting, 270
 - REST API, 270
 - SmartSync, 270
- modeler contributions, 11
- relationships
 - explicitly defined, 26
 - integrity enforced, 26
 - records, creating, 121
 - Services Manager application, 55-58
 - SOQL, 26-27
 - SOQL versus SQL, 27-28
 - SOSL, 29
 - viewing, 121
- REST API access, 306

security

- architecture, 71
- field accessibility, 73
- object-level. *See* object-level security
- overview, 71-74
- permission sets, 72
- profiles, 72
- record-level, 72
- sharing model, 73
- sharing reasons, 74

Services Manager application integration

- implementation strategy, 363-364
- sample implementation, 364-366
- scenario, 363

storage custom settings

- defined, 47
- governor limits, 180
- hierarchy, 49, 181
- list, 48
- records, 180
- storage limits, 49
- types, 47-48
- values, retrieving, 180

tools, 34

- Data Loader, 34
- Excel Connector, 34
- Import Wizard, 34

Database.com, 4

databases

- administrator contributions, 12
- Apex integration
 - DML statements. *See* DML, statements
 - integrity, 122
 - objects, referencing, 121-122
 - overview, 120-121
 - queries. *See* queries
 - security, 133
- change exceptions, 114
- custom settings, 47-48
 - defined, 47
 - hierarchy, 49
 - list, 48
 - storage limits, 49
 - types, 47-48
- data. *See* data
- developer contributions, 12
- fields. *See* fields

- governor limits, 120
- integration, 29
 - logical, 13
 - metadata XML, 30-31
 - native user interface, 31
 - object-relational mapping, 30
 - REST APIs, 31
 - SOAP APIs, 31
- objects. *See* objects
- queries. *See* queries
- records. *See* records
- relational, 6
- relationships. *See* relationships
- security
 - Apex, 133
 - architecture, 71
 - field accessibility, 73
 - object-level. *See* object-level security
 - object permissions, 73
 - overview, 71-74
 - permission sets, 72
 - profiles, 72
 - record-level, 72
 - sharing model, 73
 - sharing reasons, 74
- services, 7
- tables. *See* objects
- triggers, 130-131
 - batching, 132
 - bulkifying, 132
 - custom objects, creating, 37
 - definitions, 131-132
 - email notifications, 181-182
 - error handling, 132-133
 - names, 131
 - page navigation, 195
 - timecard validation, creating, 138-139
- data components, 200-203**
 - metadata-aware, 200-201
 - inputField, 201
 - outputField, 201
 - Mobile Components for Visualforce, 268
 - primitive, 201-202
 - inputCheckbox, 202
 - inputFile, 202
 - inputHidden, 202
 - inputSecret, 202
 - inputText, 202
 - inputTextArea, 202
 - outputLabel, 202
 - selectCheckboxes, 202
 - selectList, 202
 - selectRadio, 202
 - repeating, 201-203
- dataList component, 203**
- Data Loader tool, 34**
 - data preparation, 64-66
 - Contact CSV import file, 69
 - exporting CSV files, 64-65
 - Project CSV import file, 65
 - data verification, 67-69
 - importing data, 66
- Data Manipulation Records. *See* DML**
- data model (Services Manager)**
 - design goals
 - Developer Edition, optimization, 50
 - standard objects, leveraging, 50
 - implementing
 - custom application, creating, 58
 - custom objects, creating, 59-60
 - custom object tabs, creating, 63
 - field visibility, 64
 - Lookup relationship, creating, 60
 - Master-Detail relationships, creating, 60-62
 - validation rules, creating, 63
 - specification, 50
 - assignments, 53-54
 - contacts, 51
 - data relationships, 55-58
 - projects, 52
 - skills, 53
 - timecards, 53-56
- dataTable component, 203**
- data types**
 - Apex, 106
 - blob, 106
 - Boolean, 106
 - converting, 107-108
 - converting dates to strings, 109
 - converting strings to dates, 109
 - date, 106
 - datetime, 106
 - decimal, 106
 - double, 106

- ID, 106
- Integer, 106
- long, 106
- object, 106
- string, 106
- time, 106
- arrays
 - creating, 111
 - initializing, 111-112
 - sorting, 112
- collections
 - clearing, 109
 - cloning, 109
 - emptiness, 109
 - size, 109
- converting, 114
- fields, selecting, 38
- lists
 - creating, 111
 - initializing, 111-112
 - nesting, 111
 - overview, 111
 - sorting, 112
- maps, 112-113
- rich, 25
- sets, 112
- SOAP types, mapping, 321
- dates, 38**
 - converting to strings, 109
 - defined, 106
 - SOAP type, mapping, 322
 - String conversions, 109
- datetime data type**
 - converting to strings, 109
 - defined, 106
 - SimpleDateFormat pattern, 109
 - SOAP type, mapping, 322
 - string conversions, 109
- DE accounts**
 - logging in, 32
 - orgs, 32
 - registration, 32
- debugging**
 - Apex, 133
 - checkpoints, 133-135
 - execution logs, 134
 - batch jobs execution details, viewing, 288
 - Visualforce
 - component identifier problems, 240
 - user interfaces, 216
- decimals**
 - defined, 106
 - rounding, 108
- declarative metadata, 7**
- declaring**
 - future methods, 116
 - variables, 105-106
- delegated administration sharing reason, 82**
- Delete button, 213**
- delete permission, 76**
- delete service, 325**
- Delete statement, 130**
- deleting**
 - Chatter comments, 375
 - custom setting records, 180
 - PushTopics, 341
 - record requests, 311
 - records, 130, 325
 - scheduled batch jobs, 296
- dependent fields, 46**
- deploying mobile Web applications, 271-272**
- deployment status, 36**
- detail component, 209**
- Developer Console**
 - Apex, debugging, 133-134
 - unit test results, viewing, 137
 - Visualforce user interfaces, debugging, 216-218
- Developer Force Web site, 16**
- development**
 - Batch Apex schedulable code, 294
 - discussion boards, 16
 - environments, 32
 - lifecycle, 12
 - configuration management, 14
 - end of life, 15
 - integrated logical databases, 13
 - integrated unit testing, 14-15
 - interoperability, 15
 - MVC pattern, 15
 - native user interfaces, 14
 - mobile applications
 - hybrid, 265, 267
 - native, 265-266
 - Salesforce Mobile SDK, 265
 - Web. *See* mobile applications, Web

- Visualforce
 - process, 188
 - tools, 188-190
- dialog boxes**
 - Create Lookup Field, 61
 - Create New Object, 59
 - Open Perspective, 101
- dirty writes, 161**
- division (/) operator, 110**
- DML (Data Manipulation Language), 128**
 - database methods, 157-158
 - insert example, 158
 - opt_allOrNone parameter, 158
 - statements
 - Delete, 130
 - Insert, 129
 - Undelete, 130
 - Update, 129
 - Upsert, 129-130
- DmlException exception, 114**
- domain names (sites), 255**
- double data type**
 - defined, 106
 - rounding, 108
- Do-While loops, 114**
- Dreamforce conference, 17**
- dynamic Apex, 174**
 - instances, creating, 179
 - queries
 - governor limits, 176
 - SOQL, 175-176
 - SOSL, 176
 - schema metadata, 177
 - child relationship, 178
 - field, 177-178
 - limits, 177
 - object, 177
 - picklist, 178
 - record type, 179
- dynamic Chatter objects, 370**
- dynamicComponent elements, 248**
- dynamic field references, 246-248**
- dynamic status messages, 239**
- dynamic Visualforce, 246**
 - component generation, 248-249
 - dynamic field references, 246-248

E

- EC2 (Elastic Compute Cloud), 2-3**
- editing**
 - mobile timecards, 277-279
 - scheduled batch jobs, 296
- edit page, 211**
- edit permission, 75**
- Elastic Beanstalk, 2**
- email**
 - fields, 38
 - integration, 9
 - receiving, 172-173
 - class, creating, 173-174
 - governor limits, 173
 - personalizing based on sender identity, 173
 - services, configuring, 174-175
 - uncaught exceptions, 173
 - sending, 168
 - attachments, 172
 - blind-carbon-copies, 171
 - carbon copies, 171
 - mass emails, 170-171
 - notifications (Services Manager application), 181-182
 - organization-wide email address unique identifiers, 172
 - reply-to addresses, 171
 - sendEmail method, 171
 - sender display names, 171
 - signatures, 172
 - SingleEmailMessage object, 168-169
 - templates, 169-170
 - tracking, 172
 - SOAP data type, mapping, 322
- enabled permissions (SOAP API), 318-319**
- encapsulation, 117-118**
- EncodingUtil class, 303**
- end of life, 15**
- enhancedList component, 209**
- Enhanced Profile List Views, 74**
- Enhanced Profile User Interface, 74**
- Enterprise API**
 - overview, 315
 - records
 - bulk modifications, 326
 - creating, 324-325

- deleting/undeleting, 325
- retrieving, 323-324
- updating, 325
- upserting, 325
- writing, 324

EntitySubscription object, 377

enums, 107

environments, 32

equality (==) operator, 110

error handling

- SOAP API, 321
- triggers, 132-133
- Visualforce, 220-221
 - communication, 220-221
 - uncaught exceptions, 220

errors

- communication, 220-221
- data type conversions, 108

events

- JavaScript, 237-238
- timed, 237

exact equality (===) operator, 110

exact inequality (!==) operator, 110

Excel Connector, 34

exceptions

- incoming email, 173
- statements, 114-115
 - examples, 115
 - handling, 115
 - raising, 115
- uncaught, 220

EXCLUDES keyword (multi-select picklists), 154

Execute Anonymous view (IDE)

- batch jobs, running, 286
- missing timecard report, testing, 300
- REST services integration, testing, 304

executeBatch method, 286

execute method (Batchable interface), 283

execution logs, 134

exporting

- CSV files, 64-65
- records, 347-349
 - batches, creating, 348
 - creating bulk export jobs, 347-348
 - results, retrieving, 348-349
 - status, checking, 348

expressions

- combining, 194
- condition, 194
- if/then/else, 110
- scheduling, 295
- standard controllers
 - actions, 192
 - data, 192
 - navigation, 192
- standard set controllers
 - action, 193
 - data, 193
 - filters, 193
 - navigation, 193
 - pagination, 193

extensions (controller), 197

extensions attribute (pages), 200

external IDs, 39

F

facets, 239

FeedComments relationship, 375

feed component, 381

Feed objects, 370-371

- custom objects, 373
- news, 374
- standard objects, 372-373
- users, 374

FeedTrackedChange object, 376

feed-tracked changes (Chatter), 376

feedWithFollowers component, 381

Field change Chatter posts, 371

fields, 23

- accessibility, 73, 78-79, 89-90
- Assignment object, 54
- auto number, 41
- categories, 23
- checkboxes, 38
- Contact object, 51
- ContainerId, 355
- creating, 37
 - default values, 39
 - descriptions, 39
 - external IDs, 39
 - help text, 39
 - labels, 39
 - names, 39

- required, 39
 - types, selecting, 38
 - unique, 39
 - date/time, 38
 - dependent, 46
 - dynamic references, 246-248
 - email/phone/URL, 38
 - field sets, 247-248
 - fieldsToNull, 325
 - formula, 24-25, 41
 - history tracking, 25, 36
 - logical, 23
 - metadata, 177-178
 - multi-select picklists, 154
 - NotifyForFields, 341
 - NotifyForOperations, 341
 - numbers/percent/currency/
 - geolocation, 38
 - picklists, 38
 - metadata, 178
 - multi-select, 154
 - SOAP type, mapping, 322
 - Project object, 52
 - query results, sorting, 125
 - relationships. *See* relationships
 - RequestId, 355
 - rich data types, 25
 - roll-up summary, 41, 45
 - security, 77
 - field accessibility, 78-79, 89-90
 - profiles, 78
 - sharing objects, 162-163
 - Skill object, 54
 - standard, 36
 - text, 38
 - Timecard object, 56
 - unique identifiers, 24
 - validation rules, 24
 - visibility, 64
- fieldsToNull field, 325**
- fields variable, 247**
- file-based services, 361**
- File Chatter posts, 371**
- files (CSV)**
- Contact import, 69
 - exporting, 64-65
 - Project import, 65
- filtering**
- multi-select picklists, 154
 - records
 - grouped, 146
 - SQL, 124-125
 - standard set controllers, 193
- finally keyword (exceptions), 115**
- finish method (Batchable interface), 284**
- follow component, 381**
- followers component, 381**
- following records (Chatter), 376-378**
- following relationships, 377
 - method, 377
 - relationships, 377
 - request, 380
 - unfollowing, 377-378
- Follow Team button, 382-385**
- configuring, 385
 - controller extension code, 383-384
 - testing, 385
 - Visualforce page, 384-385
- Force.com**
- architecture
 - application services, 6
 - declarative metadata, 7
 - multitenancy, 4-6
 - programming languages, 7
 - relational databases, 6
 - Database.com, 4
 - developers, 3
 - perspective, 101
 - Project, 103
 - services, 7
 - business logic, 8
 - database, 7
 - integration, 8-9
 - user interface, 8
 - technology integrations, 4
- Force.com-styled components, 204-205**
- action containers, 205
 - notifications, 205
 - page structure, 205
 - paging, 205
 - samples
 - controller, 206
 - page, 207
 - table, 205

forcetk.js library, 266
For loops, 114, 127
formatting
 datetime data types, 109
 REST services, 302
 SimpleDateFormat pattern, 109
 strings for dates, 109
formulas, 24-25, 41, 322
frameworks (mobile Web applications), 268-269
 Mobile Components for Visualforce, 268-269
 Web MVC, 269
functions (aggregate), 144-145
 AVG, 144
 COUNT, 144-145
 COUNT_DISTINCT, 144
 governor limits, 145
 MAX, 144
 MIN, 144
 records, grouping, 146
 SUM, 144-145
future methods, 116
 declaring, 116
 limitations, 116

G

geolocation fields, 38
getCurrentUserContact method, 225
getDescribe method, 177
getInstance method, 181
getObject method, 176
Google Cloud Platform, 3
governor limits, 120
 aggregate functions, 145
 Apex code, 120
 custom Apex Web services
 REST, 312
 SOAP, 327
 custom settings, 180
 databases, 120
 dynamic queries, 176
 Force.com Apex Code Developer's Guide
 Web site, 100
 heaps, 120
 inbound email, 173
 namespaces, 120

overview, 100
 Visualforce, 221
greater than (>) operator, 110
greater than or equal to (>=) operator, 110
GROUP BY clause (record groupings), 145
GROUP BY CUBE clause, 147-148
GROUP BY ROLLUP clause, 147
groups
 operators, 110
 records, 145-146
 with aggregate functions, 146
 filtering, 146
 subtotals, 147-148
 without aggregate functions, 145-146
 users, 80
 public, 80
 roles, 80

H

The Hammer, 6
handleInboundEmail method, 172
handling
 errors
 SOAP API, 321
 triggers, 132-133
 Visualforce, 220-221
 exceptions, 115
HAVING keyword (grouped records, filtering), 146
heap governor limits, 120
Hello World
 code example, 105
 Visualforce example, 189-191
help
 settings (custom objects), 36
 text, 39
hierarchy custom settings, 49, 181
high volume objects, 370
history tracking
 custom objects, 36
 fields, 25
HTTP Apex classes, 302-303
Http class, 302
HttpRequest class, 302
HttpResponse class, 302
hybrid applications, 265, 267

laaS. *See* PaaS

id attribute (view components), 199

IDE

- Execute Anonymous view
 - batch jobs, running, 286
 - missing timecard report, testing, 300
 - REST services integration, testing, 304

installation, 101

perspective, 101

Project, 103

Schema Explorer, 103

Services Manager application
configuration, 138

Views

- Apex Test Runner, 103
- Execute Anonymous, 104-105
- Problems, 103

Visualforce page editor, 189

Ideas Web site, 16

IDs

- defined, 106
- external, 39
- string conversion, 108

iframe component, 204

if/then/else expression shortcut (? :), 110

image component, 204

implicit conversions (data types), 107-108

importing

- data, 64
 - import process, 66
 - preparations, 64-66
 - Contact CSV import file, 69
 - exporting CSV files, 64-65
 - Project CSV import file, 65
 - verification, 67-69
- records, 346-347
 - adding records to bulk import jobs, 346-347
 - closing bulk import jobs, 347
 - creating bulk import jobs, 346
 - results, retrieving, 347
 - status, checking, 347

Import Wizard, 34

InboundEmailHandler interface, 172

inbound email. *See* receiving email

include component, 242

includeScript component, 205

INCLUDES keyword (multi-select
picklists), 154

inclusion (modular Visualforce), 242

information-hiding notation, 118

Infrastructure as a Service. *See* PaaS

inheritance (Apex), 119

initializers, 118

inner classes, 118

inner joins, 149-150

inputCheckbox component, 202

inputField component, 201

inputFile component, 202

inputHidden component, 202

inputSecret component, 202

inputTextArea component, 202

inputText component, 202

insert database method, 158

Insert statement, 129

installing

IDE, 101

Mobile Components for Visualforce,
268-269

instances, creating, 179

Integers, 106

integration, 29

Apex callouts, 301-304

databases in Apex, 120-121

DML statements. *See* DML,
statements

integrity, 122

objects, referencing, 121-122

queries. *See* queries

security, 133

logical databases, 13

metadata XML, 30-31

native user interface, 31

object-relational mapping, 30

REST APIs, 61

services, 8-9

Services Manager application

implementation strategy, 363-364

sample implementation, 364-366

scenario, 363

SOAP APIs, 31

specialist contributions, 12

Visualforce and native user interface,
209-210
 custom buttons/links, 215
 custom tabs, 215
 page layouts, 213
 standard buttons, 213
 standard pages, 210-211

interfaces

Batchable, 283-284
 InboundEmailHandler, 172
 Schedulable, 294
 Stateful, 290

interoperability, 15

IP white-listing, 319

isEmpty method, 109

@isTest, 136

iterable batch scope, 290-292

Iterable class, 291

Iterator class, 290

J

Java

createProject service, 329
 create service, 324
 Metadata API object creation example,
361-363
 query batch sizes, setting, 324
 SOQL queries, executing, 323
 stub code, generating, 317

JavaScript

dynamic action status messages, 239
 events, 237-238
 forcelk.js library, 266
 JQuery UI, 259
 Object Notation (JSON), 302, 364-366
 remoting, 250, 270
 Skills Matrix comparison overlay, 261
 Visualforce components,
referencing, 240

job function security, 85-86

jobs

bulk export
 batches, creating, 348
 creating, 347-348
 results, retrieving, 348-349
 status, checking, 348

bulk import

closing, 347
 creating, 346
 records, adding, 346-347
 results, retrieving, 347
 status, checking, 347

joins

anti-joins
 overview, 152
 restrictions, 153
 inner, 149-150
 outer, 148-149
 semi-joins
 child-to-child, 153
 child-to-parent, 153
 parent-to-child, 151
 restrictions, 153

jQuery UI, 259

JSON (JavaScript Object Notation), 302, 364-366

K

keywords

break, 114
 catch, 115
 continue, 114
 EXCLUDES, 154
 finally, 115
 HAVING, 146
 INCLUDES, 154
 LIMIT, 125
 throw, 115
 try, 115

L

labels

custom objects, 35
 fields, 39

layouts

page, 37
 search, 37

LDV (Large Data Volume) deployments, 22

less than (<) operator, 110

less than or equal to (<=) operator, 110

licensing

orgs, 32
 profiles, 76

lifecycles (development), 12

- configuration management, 14
- end of life, 15
- integrated logical databases, 13
- integrated unit testing, 14-15
- interoperability, 15
- MVC pattern, 15
- native user interfaces, 14

LIMIT keyword (records), 125

links

- custom
 - custom objects, creating, 38
 - Visualforce pages, 215
- standard, 37

listings

- actionFunction component (Visualforce)
 - controllers, 236
 - pages, 236
- actionPoller component, 237
- actionStatus component, 238
 - with facets, 239
 - JavaScript functions, 239
- actionSupport component, 237-238
- aggregate functions
 - COUNT, 145
 - SUM, 145
- AngularJS project list example (Visualforce)
 - controller, 252
 - page code, 252-253
- Apex ORM code snippet, 30
- arrays
 - creating, 111
 - initializing, 112
- Batch Apex
 - class, 285
 - execution scope, 289
 - iterable batch example, 291
 - project iterator, 291
 - running batch jobs, 286
 - schedulable code, 294
 - scheduling example, 296
 - stateful example, 289
 - test, 293
- Bulk API password authentication, 345
- bulk export jobs
 - creating, 348
 - results, retrieving, 348-349
 - status, checking, 348

bulk import jobs

- closing, 347
- creating, 346
- records, adding, 346
- results, retrieving, 347
- status, checking, 347

Canvas App

- callback HTML page, 354
- main HTML page, 353-354

Chatter

- feed-tracked changes, 376
- following records, 377
- following relationships, 377
- unfollowing records, 378
- Visualforce component page, 381
- Visualforce controller example, 378
- Visualforce page example, 378

Chatter comments

- creating, 375
- deleting, 375
- query, 375

Chatter posts

- creating, 372
- custom object query, 373
- deleting, 373
- standard object query, 372
- user feed query, 374

Chatter REST API requests

- followed records, 380
- news feed, 380
- posts, 380

class definitions, 118

commandButton component, 203

conditional statements, 113

constants, defining, 107

Contact CSV import file, 69

custom Apex REST Web services

- creating, 313
- invoking, 313

custom Apex SOAP Web services

- creating record example, 328
- invoking, 329

custom controllers

- extensions, 197
- read-only access to Project record, 194

wrapper patterns, 195-196

- custom settings
 - creating custom setting records, 180
 - deleting, 181
 - updating, 180
 - values, retrieving, 180
- data integrity, 122
- data type conversions
 - conversion methods, 108
 - errors, 108
 - ID and string, 108
 - implicit conversion, 107
 - strings to dates, 109
- datetime data types, formatting, 109
- dynamic queries
 - SOQL, 175
 - SOSL, 176
- Enterprise API
 - creating records, 325
 - query batch sizes, 324
 - record retrieval SOQL query, 323
- enums, defining, 107
- error-severity message, 221
- exception statements, 115
- Force.com-styled components
 - controller, 206
 - page, 207
- formula field example, 24-25
- future method declaration, 116
- Hello World, 105
- include component, 242
- inputField component, 201
- insert DML database method, 158
- instances, creating, 179
- lists
 - creating, 111
 - initializing, 112
 - nesting, 111
- maps, 113
- Metadata API object creation, 362-363
- metadata XML example, 31
- MissingTimecardBatch class
 - creating, 298-299
 - reset results, 300
 - running, 300
- mobile timecard entry page
 - editing timecards, 277-279
 - listing timecards controller, 274
 - listing timecards Visualforce
 - page, 275
 - navigation, AngularJS controller, 277
- outputField component, 201
- Project CSV import file, 65
- receiving email, 173
- records
 - creating, 121
 - deleting, 130
 - inserting, 129
 - locking, 161
 - relationships, creating, 121
 - undeleting, 130
 - updating, 129
 - upserting, 130
- records, grouping
 - with aggregate functions, 146
 - debug log excerpt, 147
 - filtering grouped, 146
 - GROUP BY CUBE clause, 147-148
 - GROUP BY ROLLUP clause, 147
 - without aggregate functions, 146
- recursion, 115
- repeat component, 203
- REST API
 - authentication, 307
 - creating record requests, 310
 - deleting record requests, 311
 - record retrieval by external
 - identifiers, 310
 - record retrieval by unique
 - identifiers, 309
 - services available call, 309
 - SObject basic information
 - request, 309
 - SOQL query request, 310
 - updating record requests, 311
 - upserting record requests, 311
- rounding operations, 108
- savepoints, 160
- schema metadata
 - child relationship, 178
 - field, 177
 - object, 177
 - picklist, 178
 - record type, 179

- sending email
 - mass email, 171
 - SingleEmailMessage object, 168
 - template, 170
 - Services Manager application
 - anonymous benchmark Web service, testing, 334
 - email notifications, 182
 - integration implementation example, 364-366
 - utilization controller, 335-337
 - Utilization page code, 337-338
 - Services Manager Follow Team button
 - controller extension code, 383
 - Visualforce page, 384
 - Services Manager Skills Matrix
 - controller, 226-227
 - unit test, 229-231
 - Visualforce page, 228
 - Services Manager Skills Matrix
 - comparison overlay
 - actionSupport, adding, 262
 - CompareSkillsComponent, 260
 - CompareSkillsController, 260
 - component CSS, adding, 261
 - component support, adding, 261
 - JavaScript integration, 261
 - sets, 112
 - sharing rules, inserting, 167
 - Skill type field error condition
 - formula, 63
 - SOAP API, logging in, 320
 - SOQL
 - child-to-child semi-join, 153
 - child-to-parent relationships, 126
 - child-to-parent semi-join, 153
 - filter conditions, 124
 - Group Object query, 166
 - inner join, 150
 - multi-select picklists, 154
 - outer join, 148
 - parent-to-child query, 151
 - parent-to-child relationships, 126
 - Project Share Object query, 165
 - query in Apex, 127
 - query in Apex with For loop, 127
 - record limits, 125
 - relationship query, 28
 - semi-join, 152
 - sort fields, 125
 - statement, 124
 - SOSL
 - Apex, 156
 - query, 29
 - SQL relationship query, 27
 - standard controllers
 - multiple records, 192
 - single records, 191
 - Streaming API Visualforce
 - controller, 342
 - test methods, 136
 - TimecardManager class
 - creating, 139
 - unit tests, 140-141
 - Tooling API example (Visualforce)
 - controller, 357-359
 - page, 359-360
 - triggers
 - batching, 132
 - definition, 131
 - validateTimecard trigger, 138
 - validation rule example, 24
 - variables
 - declaring, 105
 - name case insensitivity, 106
 - view components syntax, 199
 - Visualforce
 - controller partial page refresh, 235
 - controller unit test, 222
 - dynamic components, 248-249
 - dynamic field references, 247
 - Hello World example, 190
 - pages as templates, 243
 - partial page refresh, 235
 - record-level security, 219
 - view components, 244, 246
 - Yahoo! geocoding REST service
 - integrating, 303
 - invoking, 303
 - testing, 304

lists

- creating, 111
- custom settings, 48
- initializing, 111-112
- nesting, 111

- overview, 111
- pages, 211
- sorting, 112
- List/Set Iteration For loops, 114**
- listViews component, 208**
- local Web servers, configuring, 352**
- locking records, 161**
- logging in**
 - DE accounts, 32
 - SOAP API, 318-320
 - enabled permissions, 318-319
 - IP white-listing, 327
 - logging out, 320
 - login call, 320
 - problems, troubleshooting, 320
 - security tokens, 319
- logical databases integration, 13**
- logical negation (!) operator, 110**
- login method, 320**
- LoginResult object, 320**
- logs**
 - debug, 288
 - execution, 134
- long data type, 106**
- Lookup relationships**
 - creating, 60
 - defined, 39
 - Master-Detail relationships,
 - compared, 40
 - Services Manager application, 55
 - SOAP type, mapping, 322
- loops, 114, 127**

M

- managed sharing (Apex), 162**
 - organization-wide sharing defaults,
 - changing, 163
 - restrictions, 163
 - sharing objects, 162-163
 - sharing rules, creating, 163-167
 - inserting, 167
 - Project object, 164
 - SOQL queries, 165-166
 - viewing, 163, 167
- manual sharing reason, 82**
- maps, 112-113**
- mashups, 9**

- MassEmailMessage object, 170-171**
- mass emails, sending, 170-171**
- Master-Detail relationships**
 - creating, 60-62
 - defined, 40
 - Lookup relationships, compared, 40
 - Services Manager application, 55-57
 - SOAP type, mapping, 322
- MAX aggregate function, 144**
- messages component, 220-221**
- metadata**
 - declarative, 7
 - schema, 177
 - child relationship, 178
 - field, 177-178
 - limits, 177
 - object, 177
 - picklist, 178
 - record type, 179
 - tools, 33
 - XML, 30-31
- Metadata API, 360**
 - object creation example, 361-363
 - services, 360-361
 - Web site, 360
- metadata-aware components, 200-201**
 - inputField, 201
 - outputField, 201
- methods**
 - abortJob, 296
 - access modifiers, 118
 - action, 195-197
 - addError, 225
 - addFields, 246
 - addInfo, 225
 - Apex test, 136
 - clear, 109
 - clone, 109
 - defined, 117
 - DML database, 157-158
 - insert example, 158
 - opt_allOrNone parameter, 158
 - execute, 283
 - executeBatch, 286
 - finish, 284
 - future, 116
 - declaring, 116
 - limitations, 116

- getCurrentUserContact, 225
- getDescribe, 177
- getInstance, 181
- getSObject, 176
- handleInboundEmail, 172
- isEmpty, 109
- login, 320
- nav, 277
- navClass, 277
- overloading, 119
- query
 - SOQL, 175
 - SOSL, 176
- rollback, 159
- schedule, 295
- sendEmail, 171
- setBccSender, 171
- setCcAddresses, 171
- setDocumentAttachments, 172
- setFileAttachments, 172
- setOrgWideEmailAddressId, 172
- setReplyTo, 171
- setSaveAsActivity, 172
- setSavePoint, 159
- setSenderDisplayName, 171
- setUseSignature, 172
- size, 109
- start, 283
- testAsUser, 231
- testNoContactForUser, 231
- testNoContactSelected, 231
- testNoSkills, 231
- testSave, 231
- testWithSkills, 231
- valueOf
 - date to string conversions, 109
 - string to date conversions, 109

MIN aggregate function, 144

MissingTimecardBatch class

- creating, 298-299
- reset results, 300
- running, 300

missing timecard reports, 296-297

- missing timecards information,
 - calculating, 298-299
- missing timecards list custom object,
 - creating, 297
- testing, 299-300

mobile applications

- Chatter Mobile, 264
- containers, 271
- hybrid, 265, 267
- native, 265-266
- Salesforce
 - Classic, 264
 - Mobile SDK, 265
 - Touch, 264
- timecard entry page
 - editing timecards, 277-279
 - in-page navigation, 276-277
 - listing timecards, 273-276
 - requirements, 272
 - testing, 279
 - viewing in Web browsers, 273
 - viewing on iPhones, 273
- Web, 265
 - data access, 269-270
 - deployment, 271-272
 - frameworks, 268-269
 - overview, 266
 - Salesforce SDK libraries, 266

Mobile Components for Visualforce, 268-269

- documentation/source code Web site, 269
- installing, 268-269
- types, 268

Mobile Design templates, 269

Mobile Packs, 269

mobile timecard entry pages

- in-page navigation, 276-277
- requirements, 272
- testing, 279
- timecards
 - editing, 277-279
 - listing, 273-276
- viewing
 - iPhones, 273
 - Web browsers, 273

Model-View-Controller (MVC) pattern, 15

Modify All permission, 76

modularity (Apex), 119

modular Visualforce pages, 241

- composition, 243-244
- custom components, 244-246
 - defining, 244-245
 - Google Map example, 245-246
- inclusion, 242
- static resources, 241-242

monitoring batch jobs, 287-288
 multitenancy, 4-6
 multiplication (*) operator, 110
 multi-select picklists, 154
 MVC (Model-View-Controller) pattern, 15
 MyEmailService class, 173-174

N

NAICS (North American Industry Classification System) codes, 333

names

- channel, 340
- custom objects, 35
- fields, 39
- sender display (email), 171
- sites domain names, 255
- triggers, 131
- variables, 105-106
- view components, 199

namespaces, 120

native applications, 265-266

native user interfaces

- CRUD (create, read, update, delete)
 - operations, 31
- data integration, 31
- development lifecycle, 14
- new features, enabling, 41
- view components, 208-209
 - detail, 209
 - enhancedList, 209
 - listViews, 208
 - relatedList, 209

Visualforce development tool, 189

Visualforce integration, 209-210

- custom buttons/links, 215
- custom tabs, 215
- page layouts, 213
- standard buttons, 213
- standard pages, 210-211

navClass method, 277

navigation

- expressions
 - standard controllers, 192
 - standard set controllers, 193
- Mobile Components for Visualforce, 268
- mobile timecard entry page, 276-277

nav method, 277

nesting lists, 111

New button, 213

New Custom Field Wizard

- default values, 39
- descriptions, 39
- external IDs, 39
- help text, 39
- labels, 39
- names, 39
- required fields, 39
- types, selecting, 39
- unique fields, 39

New Custom Object Tab Wizard, 63

news feeds

- defined, 381
- requests, 379-380

North American Industry Classification System (NAICS) codes, 333

not equal to (!=) operator, 110

notifications, 205

- action status
 - actionStatus component, 238-240
 - dynamic, 239
 - images/stylized messages, 239
- Streaming API
 - PushTopics, 340-341
 - Web site, 340

NotifyForFields field, 341

NotifyForOperations field, 341

NullPointerException exception, 114

number data type, 38, 322

O

OAuth, 270, 306-307

object-level security, 72, 74

- field-level security, 77
 - field accessibility, 78-79
 - profiles, 78
- permission sets, 76-77
- profiles, 74
 - administrative permissions, 75
 - Enhanced Profile List Views, 74
 - Enhanced Profile User Interface, 74
 - field-level security, 78
 - licenses, 76
 - object permissions, 75-76
 - Services Manager, creating, 89-90
 - Services Manager, listing, 85-86
 - types, 74

object-oriented programming, 117

- analysis and design specialist contributions, 12
- encapsulation, 117-118
- information-hiding notation, 118
- inheritance, 119
- modularity, 119
- polymorphism, 119

Object-Relational Mapping (ORM), 30

objects, 22, 106

- AggregateResult, 145
- Assignment
 - fields, 53
 - overview, 53
- Chatter
 - dynamic, 370
 - high-volume design, 370
 - relationship-rich, 370
- Contact
 - CSV import file, 69
 - fields, 51
 - overview, 51
- ContactFeed, 372
- creating, 35, 59-60
 - activities, allowing, 36
 - custom buttons/links, 38
 - custom fields, 37
 - definition, 35-36
 - deployment status, 36
 - descriptions, 36
 - field history tracking, 36
 - help settings, 36
 - labels, 35
 - with Metadata API, 361-363
 - names, 35
 - page layouts, 37
 - record name label, 36
 - reports, allowing, 36
 - search layouts, 37
 - standard buttons/links, 37
 - standard fields, 36
 - triggers, 37
 - validation rules, 37
- EntitySubscription, 377
- Feed, 370-371
 - custom objects, 373
 - news, 374
 - standard objects, 372-373
 - users, 374

- FeedTrackedChange, 376
- logical, 22
- LoginResult, 320
- MassEmailMessage, 170-171
- metadata, 177
- operational tasks, 22
- permissions, 73, 75-76
- Project
 - CSV import file, 65
 - custom object tab, creating, 63
 - fields, 52
 - overview, 52
 - sharing rules, 164
- records
 - creating, 42, 121
 - relationships, creating, 121
 - types, 47
- referencing in Apex, 121-122
- SaveResult, 324
- security, 218
- services, 361
- sharing, 162-163
 - fields, 162-163
 - restrictions, 163
- SingleEmailMessage, 168-169
- Skill
 - fields, 54
 - overview, 54
 - validation rule, creating, 63
- SOQL relationships, 125-126
 - child-to-parent, 125-126
 - parent-to-child, 126
- standard, 22
- tabs, creating, 41
- Timecard
 - fields, 56
 - overview, 53
- undelete support, 23
- Views, 43-44

Open Perspective dialog box, 101

operations specialist contributions, 12

operators, 109

- & (AND) operator, 110
- AND (&&), 110
- addition (+), 110
- arithmetic negation (-), 110
- assignment (=), 110
- bitwise, 110

division (/), 110
 equality (==), 110
 exact equality (===), 110
 exact inequality (!==), 110
 greater than (>), 110
 greater than or equal to (>=), 110
 grouping, 110
 if/then/else expression (? :), 110
 less than (<), 110
 less than or equal to (<=), 110
 logical negation (!), 110
 multiplication (*), 110
 not equal to (!=), 110
 OR (||), 110
 signed shift left (<<), 110
 signed shift right (>>), 110
 string concatenation (+), 110
 subtraction (-), 110
 unary decrement (--), 110
 unary increment (++), 110
 unsigned shift right (>>>), 110
opt_allOrNone parameter (DML database methods), 158
organization-wide
 email address unique identifiers, 172
 security defaults
 overview, 80-82
 Services Manager application, 91
orgs, 32
ORM (Object-Relational Mapping), 30
OR (|) operator, 110
OR (||) operator, 110
outbound email. See sending email
outer joins, 148-149
outputField component, 201
outputLabel component, 202
outputLink component, 204
outputPanel component, 204
outputText component, 204
overloading methods, 119
overriding
 standard buttons, 213
 standard pages, 210-211
ownership (records), 79-80

P

PaaS (Platform as a Service), 2
 Amazon Web Services, 2-3
 Force.com, 3-4
 Google Cloud Platform, 3
 Windows Azure, 3
pageBlockButtons component, 228
pageBlock component, 228
pageBlockTable component, 228
pageMessages component, 220-221
pages
 adding to sites, 256-257
 Canvas App
 adding, 352
 configuring, 352
 components, 200
 layouts
 custom objects, creating, 37
 Visualforce pages, adding, 213
 security, 219
 standard native user interface
 edit, 211
 list, 211
 overriding, 211
 tab, 210
 view, 211
 structure components, 205
 view state, preserving, 195
 Visualforce, 187
 actionFunction component, 236
 adding to page layouts, 213
 adding to Salesforce Touch, 271
 AngularJS example code, 252-253
 asynchronous actions. *See*
 asynchronous actions
 Chatter components, 381
 Chatter example, 378
 dynamic, 246-249
 JavaScript events, 237-238
 mobile timecards, 275-276, 278
 modular, 241-246
 native user interface buttons/links
 navigation, 215
 as native user interface tabs, 215
 partial refreshes, 234-235

- performance tuning, 217-218
- public access. *See* sites
- security, 218-220
- Services Manager Follow Team button, 384-385
- Services Manager Skills Matrix, 228-229
- Services Manager Utilization, 337-338
- Services Manager Utilization page, creating, 330-331
- standard pages, overriding, 210-211
- Streaming API example, 343
- timed events, 237
- Tooling API example, 359-360
- viewing from native user interface buttons, 213
- viewing in Salesforce Classic, 271
- pagination expressions, 193**
- paging components, 205**
- ParentId field, 162**
- parent-to-child relationships, 126**
 - queries, 151
 - semi-join, 151
- partial page refreshes, 234-235**
- Partner SOAP API, 315**
- percent data type, 38, 322**
- performance**
 - custom Apex SOAP Web services, 327
 - Visualforce pages, tuning, 217-218
- permissions**
 - administrative, 75
 - enabled (SOAP API), 318-319
 - object, 73, 75-76
 - Services Manager profiles, 85-86
 - sets, 72, 76-77
- perspectives, 101**
- phone data type, 322**
- phone fields, 38**
- picklists, 38**
 - metadata, 178
 - multi-select, 154
 - SOAP type, mapping, 322
- Platform as a Service. *See* PaaS**
- platform documentation, 16**
- polymorphism, 119**
- posts (Chatter), 370-372**
 - content, 371
 - creating, 372-373
 - deleting, 373
- Feed objects, 370-371
 - custom object, 373
 - news, 374
 - standard objects, 372-373
 - user, 374
 - schema pattern, 370
- primitive components, 204-205**
- primitive data components, 201-202**
 - inputCheckbox, 202
 - inputFile, 202
 - inputHidden, 202
 - inputSecret, 202
 - inputText, 202
 - inputTextArea, 202
 - outputLabel, 202
 - selectCheckboxes, 202
 - selectList, 202
 - selectRadio, 202
- private records, 81**
- Problems View (IDE), 103**
- procedural sharing reasons, 82**
- profiles, 74**
 - administrative permissions, 75
 - defined, 72
 - Enhanced Profile List Views, 74
 - Enhanced Profile User Interface, 74
 - field-level security, 78
 - licenses, 76
 - object permissions, 75-76
 - Services Manager application, 18, 91-92
 - accounts receivable, 18
 - consultants, 18
 - creating, 89-90
 - listing, 85-86
 - project managers, 18
 - sales representatives, 18
 - staffing coordinators, 18
 - Vice President, 18
 - types, 74
 - user groups, 80
- programming languages, 7**
- ProjectListCtrl controller, 253**
- project manager profile**
 - permissions, 86
 - Services Manager application, 18
- projects, 103**
 - CSV import file, 65
 - custom object tab, creating, 63

- development lifecycle, 12
 - configuration management, 14
 - end of life, 15
 - integrated logical databases, 13
 - integrated unit testing, 14-15
 - interoperability, 15
 - MVC pattern, 15
 - native user interfaces, 14
- fields, 52
- overview, 52
- selecting, 10-11
- sharing rules, 164
- team selection, 11-12
- tools/resources
 - AppExchange, 16
 - Code Share, 16
 - developer discussion boards, 16
 - Developer Force Web site, 16
 - Dreamforce/Cloudforce
 - conferences, 17
 - Ideas site, 16
 - platform documentation, 16
 - systems integrators, 17
 - technical support, 17
- Visualforce, 188

properties, 117**public groups, 80****public read-only records, 81****public read/write records, 81****PushTopics, 340-341**

- availability, 341
- components, 340-341
- deleting, 341
- limitations, 341

Q

quality assurance engineer contributions, 12**queries**

- batch sizes, setting, 324
- dirty writes, 161
- exceptions, 114
- joins
 - anti-joins, 152-153
 - inner, 149-150
 - outer, 148-149
 - semi-joins. *See* semi-joins
- parent-to-child, 151
- SOQL. *See* SOQL queries

- SOSL, 29
 - Apex, 155-157
 - dynamic, 176
 - record retrieval, 324
 - syntax, 155

QueryException exception, 114**query method**

- SOQL, 175
- SOSL, 176

queryMore service, 323**query service, 323**

R

raising exceptions, 115**RCED (read, create, edit, delete)****operations, 31****Read permission, 75****receiving email, 172-173**

- class, creating, 173-174
- governor limits, 173
- personalizing based on sender identity, 173
- services, configuring, 174-175
- uncaught exceptions, 173

records

- adding to bulk import jobs, 346-347
- batch processing. *See* Batch Apex
- controlled by parent option, 81
- counts, returning, 145
- creating, 42, 121
- custom setting
 - creating, 180
 - deleting, 180
 - updating, 180
- deleting, 130
- Enterprise API
 - bulk modifications, 326
 - creating, 324-325
 - deleting/undeleting, 325
 - retrieving, 323-324
 - updating, 325
 - upserting, 325
 - writing, 324
- exporting, 347-349
 - batches, creating, 348
 - creating bulk export jobs, 347-348
 - results retrieving, 348-349
 - status, checking, 348

- feed-tracked changes (Chatter), 376
- filtering, 124-125
- following
 - method, 377
 - relationships, querying, 377
 - request, 380
- grouping, 145-146
 - with aggregate functions, 146
 - filtering grouped, 146
 - without aggregate functions, 145-146
- grouping with subtotals, 147-148
 - debug log excerpt, 147
 - GROUP BY CUBE clause, 147-148
 - GROUP BY ROLLUP clause, 147
- importing, 346-347
 - adding records to bulk import jobs, 346-347
 - closing bulk import jobs, 347
 - creating bulk import jobs, 346
 - results, retrieving, 347
 - status, checking, 347
- inserting, 129
- limits, 125
- locking, 161
- ownership, 79-80
- private, 81
- public read-only, 81
- public read/write, 81
- PushTopics, 340-341
 - availability, 341
 - components, 340-341
 - deleting, 341
 - limitations, 341
- relationships, creating, 121
- requests
 - creating, 310
 - deleting, 311
 - updating, 311
 - upserting, 311
- retrieving
 - external identifiers, 310
 - unique identifiers, 309
- security, 72, 79
 - record ownership, 79-80
 - user groups, 80
 - Visualforce user interfaces, 219
- sharing, 80-82
 - organization-wide defaults, 80-82, 163
 - Services Manager application, 87-88
 - restrictions, 163
 - sharing objects, 162-163
 - sharing reasons, 82
- sharing rules
 - inserting, 167
 - Project object, 164
 - SOQL queries, 165-166
 - viewing, 163, 167
- type metadata, 179
- types, 46-47
- undeleting, 130
- unfollowing, 377-378
- updating, 129
- upserting, 129-130
- viewing, 42
- recursion, 115**
- registration, 32**
- relatedList component, 209**
- relational databases, 6**
- relationship-rich Chatter objects, 370**
- relationships**
 - child metadata, 178
 - comparison, 40
 - creating, 39-40
 - data, 25
 - explicitly defined, 26
 - integrity enforced, 26
 - SOQL, 26-27
 - SOQL versus SQL, 27-28
 - SOSL, 29
 - viewing, 121
 - FeedComments, 375
 - fields, 38
 - comparison, 40
 - creating, 39-40
 - Lookup, 39
 - Master-Detail, 40
 - following, 377
 - Lookup, 39
 - creating, 60
 - Services Manager application, 55
 - SOAP type, mapping, 322
 - Master-Detail, 40
 - creating, 60-62
 - Services Manager application, 55-57
 - SOAP type, mapping, 322
 - parent-to-child queries, 151
 - records, creating, 121

- Services Manager application, 55-58
- SOQL, 125-126
 - child-to-parent, 125-126
 - parent-to-child, 126
 - viewing, 121
- repeat component, 203**
- repeating components, 201-203**
- reply-to addresses (email), 171**
- reports**
 - custom objects, allowing, 36
 - missing timecard, 296-297
 - missing timecards information, calculating, 298-299
 - missing timecards list custom object, creating, 297
 - testing, 299-300
- Representational State Transfer. See REST**
- RequestId field, 355**
- requests**
 - Chatter posts, 380
 - followed records, 380
 - news feed, 379-380
 - password authentication, 307
 - records
 - creating, 310
 - deleting, 311
 - retrieving, 309-310
 - updating, 311
 - upserting, 311
 - services available, 308
 - SObject basic information, 309
 - SOQL query, 310
- reRender attribute, 234**
- resources**
 - Apex Code Developer's Guide Web site, 108
 - AppExchange, 16
 - Code share, 16
 - developer discussion boards, 16
 - Developer Force Web site, 16
 - Dreamforce/Cloudforce conferences, 17
 - Ideas Web site, 16
 - platform documentation, 16
 - REST API, 308
 - security Web site, 385
 - systems integrators, 17
 - technical support, 17
- REST (Representational State Transfer), 301**
 - API
 - authentication, 306-307
 - Chatter, 379-380
 - Connected Apps, creating, 307
 - creating record requests, 310
 - data access, 306
 - deleting record requests, 311
 - Developer's Guide Web site, 308
 - mobile Web application data access, 270
 - record retrieval by external identifiers, 310
 - record retrieval by unique identifiers, 309
 - services available call, 308-309
 - SObject basic information request, 309
 - SOQL query request, 310
 - updating record requests, 311
 - upserting record requests, 311
 - custom Apex REST Web services, 312-314
 - Apex class rules, 312
 - creating, 313
 - governor limits, 312
 - invoking, 313-314
 - integration, 9
 - services, calling from Apex, 302-304
 - formats, 302
 - HTTP classes, 302-303
 - integrating, 303-304
 - invoking, 303
 - testing, 304
- rich data types, 25**
- roles. See profiles**
- rollback method, 159**
- roll-up summaries**
 - fields, 41, 45
 - SOAP type, mapping, 322
- rounding, 108**
- RowCause field, 163**
- rules**
 - Apex Web services classes
 - REST, 312
 - SOAP, 327
 - governor limits, 120
 - aggregate functions, 145
 - Apex code, 120

- custom Apex REST Web services, 312
- custom Apex SOAP Web services, 327
- custom settings, 180
- databases, 120
- dynamic queries, 176
- Force.com Apex Code Developer's Guide Web site, 100
- heaps, 120
- inbound email, 173
- namespaces, 120
- overview, 100
- Visualforce, 221
- managed sharing, 153
 - creating, 163-167
 - organization-wide sharing defaults, changing, 163
 - restrictions, 163
 - sharing objects, 162-163
- sharing, 82
 - inserting, 167
 - Services Manager application, 92-93
 - viewing, 163, 167
- validation
 - fields, 24
 - Skill object creating, 63

S

S2S (Salesforce-to-Salesforce), 9

SaaS. See PaaS

Salesforce

- Classic, 264
 - implementation guide, 271
 - Visualforce pages, viewing, 271
- Mobile Packs, 269
- Mobile SDK
 - download Web sites, 265
 - home page, 267
 - libraries, 266
- Object Query Language. *See* SOQL
- Object Search Language. *See* SOSL
- Touch, 264
 - mobile timecard entry page, testing, 279
 - Visualforce pages, viewing, 271

Salesforce-to-Salesforce (S2S), 9

sales representatives profile

- permissions, 86
- Services Manager application, 18

sample application. See Services Manager application

savepoints, 159-160

- example, 159-160
- limitations, 159
- restoring to, 159
- setting, 159

SaveResult objects, 324

Schedulable interface, 294

schedule method, 295

scheduling Batch Apex, 293-296

- Apex user interface, 294-295
- sample code, 296
- schedulable code development, 294
- scheduled jobs
 - creating, 295
 - deleting, 296
 - editing, 296

Schema Builder, 34

- custom objects, creating, 59-60
- Lookup relationships, creating, 60
- Master-Detail relationships, creating, 60-62

Schema Explorer, 103

- relationships, viewing, 121
- SOQL queries, running, 123

scope

- Batch Apex, 282
- batch jobs
 - adjusting, 289
 - iterable batch, 290-292

search layouts, 37

sectionHeader component, 228

Secure Coding Guideline document Web site, 218

security

- Apex, 133
- architecture, 71
- authentication
 - Bulk API, 345-346
 - Canvas, 349-350
 - mobile Web applications, 269-270
 - REST APIs, 306-307
 - sites users, 258
- Cross Site Request Forgery attacks, 385
- custom Apex SOAP Web services, 327
- fields, 77
 - accessibility, 73, 78-79, 89-90
 - profiles, 78

- object-level. *See* object-level security
- objects, 218
- overview, 71-74
- permission sets, 72, 76-77
- profiles, 72, 74
 - administrative permissions, 75
 - Enhanced Profile List Views, 74
 - Enhanced Profile User Interface, 74
 - field-level security, 78
 - licenses, 76
 - object permissions, 75-76
 - Services Manager, creating, 89-90
 - Services Manager, listing, 85-86
 - types, 74
- records, 72, 79
 - record ownership, 79-80
 - sharing model, 80-82
 - user groups, 80
 - Visualforce user interfaces, 219
- resources Web site, 385
- Secure Coding Guideline document Web site, 218
- Services Manager application
 - business units, 85-88
 - designing, 85
 - field accessibility, 89-90
 - implementing, 88-89
 - job functions, 85-86
 - organization-wide defaults, 91
 - profiles, 89-92
 - sharing rules, 92-93
 - Skills Matrix, 224-225
 - testing, 94-98
- sharing model, 73
- sharing reasons, 74
- sites, 255-256
- SOAP API
 - IP white-listing, 319
 - overview, 316
 - tokens, 319
- Visualforce user interfaces, 218
 - object-level, 218
 - page-level, 219
 - record-level, 219
- selectCheckboxes component, 202**
- selectedContactId variable, 225**
- selecting**
 - field types, 38
 - projects, 10-11
 - teams, 11-12
- selectList component, 202**
- selectRadio component, 202**
- semi-joins**
 - child-to-child, 153
 - child-to-parent, 153
 - parent-to-child, 151
 - restrictions, 153
- sendEmail method, 171**
- sender display names (email), 171**
- sending email, 168**
 - attachments, 172
 - blind-carbon-copies, 171
 - carbon copies, 171
 - mass emails, 170-171
 - notifications (Services Manager application), 181-182
 - organization-wide email address unique identifiers, 172
 - reply-to addresses, 171
 - sendEmail method, 171
 - sender display names, 171
 - signatures, 172
 - SingleEmailMessage object, 168-169
 - templates, 169-170
 - tracking, 172
- services, 7**
 - application, 6
 - business logic, 8
 - create, 324
 - createProject, 329
 - custom Apex REST Web, 312-314
 - Apex class rules, 312
 - creating, 313
 - governor limits, 312
 - invoking, 313-314
 - custom Apex SOAP Web, 326
 - Apex class rules, compared, 327
 - calling, 328
 - creating records example, 328
 - governor limits, 327
 - invoking, 329
 - limitations, 326-327
 - Services Manager anonymous benchmarking, 333-335

- database, 7
- delete, 325
- email, configuring, 174-175
- integration, 8-9
- Metadata API, 360-361
- query, 323
- queryMore, 323
- REST, calling from Apex, 302-304
 - formats, 302
 - HTTP classes, 302-303
 - integrating, 303-304
 - invoking, 303
 - testing, 304
- SOAP, calling from Apex, 305-306
- subjects
 - record retrieval by external identifiers, 310
 - record retrieval by unique identifiers, 309
 - SObject basic information request, 309
- update, 325
- upsert, 325
- user interface, 8
- Web, integration, 9
- Services Manager application**
 - anonymous benchmarking service, 333-335
 - background, 17-18
 - business hours, configuring, 331
 - database integration
 - implementation strategy, 363-364
 - sample implementation, 364-366
 - scenario, 363
 - data model design goals, 49-50
 - Developer Edition optimization, 50
 - standard objects, leveraging, 50
 - data model implementation
 - custom application, creating, 58
 - custom objects, creating, 59-60
 - custom object tabs, creating, 63
 - field visibility, 64
 - Lookup relationship, creating, 60
 - Master-Detail relationships, creating, 60-62
 - validation rules, creating, 63
 - data model specification, 50
 - assignments, 53-54
 - contacts, 51
 - data relationships, 55-58
 - projects, 52
 - skills, 53
 - timecards, 53-56
 - email notifications, 181-182
 - Follow Team button, 382-385
 - configuring, 385
 - controller extension code, 383-384
 - testing, 385
 - Visualforce page, 384-385
 - hours utilization calculation, 332
 - IDE configuration, 138
 - importing data, 64
 - data preparation, 64-66
 - import process, 66
 - verification, 67-69
 - missing timecard report, 296-297
 - missing information, calculating, 298-299
 - missing timecards list custom object, creating, 297
 - testing, 299-300
 - mobile timecard entry page
 - editing timecards, 277-279
 - in-page navigation, 276-277
 - listing timecards, 273-276
 - requirements, 272
 - testing, 279
 - viewing in Web browsers, 273
 - viewing on iPhones, 273
 - security
 - business units, 85-88
 - designing, 85
 - field accessibility, 89-90
 - implementing, 88-89
 - job functions, 85-86
 - organization-wide defaults, 91
 - profiles, creating, 89-90
 - roles, 91-92
 - sharing rules, 92-93
 - testing, 94-98
 - Skills Matrix
 - complete list of skill types, creating, 224
 - contacts drop-down list, creating, 224
 - controller, creating, 225-227
 - controller tests, 229-231
 - data security, 224-225

- page, creating, 224
- requirements, 223
- sample implementation, 223
- skills list, creating, 224
- Visualforce page, 228-229
- Skills Matrix comparison overlay, 259-262
 - actionSupport, adding, 262
 - component CSS, adding, 261
 - component support, adding, 261
 - custom components, creating, 259-260
 - JavaScript integration, 261
- TimecardManager class, creating, 138-139
- timecard validation
 - trigger, creating, 138-139
 - unit testing, 140-141
- user roles, 18
 - accounts receivable, 18
 - consultants, 18
 - project managers, 18
 - sales representatives, 18
 - staffing coordinators, 18
 - Vice President, 18
- utilization
 - controller code, 335-337
 - page code, 337-338
 - Visualforce page, creating, 330-331
- session sharing, 270
- setBccSender method, 171
- setCcAddresses method, 171
- setDocumentAttachments method, 172
- setFileAttachments method, 172
- setOrgWideEmailAddressId method, 172
- setReplyTo method, 171
- sets, 112
- setSaveAsActivity method, 172
- setSavepoint method, 159
- setSenderDisplayName method, 171
- setUseSignature method, 172
- sharing
 - reasons, 74
 - delegated administration, 82
 - manual, 82
 - procedural, 82
 - records, 82
 - sharing rules, 82
 - records, 73, 80-82
 - organization-wide defaults, 80-82, 163
 - procedural, 82
 - restrictions, 163
 - Services Manager application, 87-88
 - sharing reasons, 82
 - rules, 82, 92-93
 - inserting, 167
 - Services Manager application, 92-93
 - viewing, 163, 167
- sharing objects, 162-163
 - fields, 162-163
 - restrictions, 163
- showChatter attribute, 381
- signatures (email), 172
- signed shift left (<<) operator, 110
- signed shift right (>>) operator, 110
- SimpleDateFormat pattern, 109
- Simple Object Access Protocol. *See* SOAP
- SingleEmailMessage object, 168-169
- single-page applications, 250
 - AngularJS, 251-253
 - controllers, 253
 - demonstration page, 251
 - templates, 253
 - tutorial Web site, 251
 - Visualforce controller, implementing, 252
 - Visualforce page code, 252-253
 - Web site, 251
 - JavaScript remoting, 250
- sites
 - creating, 255
 - domain name, 255
 - enabling, 254
 - main page, 255
 - pages, adding, 256-257
 - security, 255-256
 - user authentication, 258
- size
 - collections, 109
 - query batches, 324
 - static resources, 241
- size method (collections), 109
- Skill object
 - fields, 54
 - overview, 54
 - validation rule, creating, 63

Skills Matrix

- comparison overlay, 259-262
 - actionSupport, adding, 262
 - component CSS, adding, 261
 - component support, adding, 261
 - custom component, creating, 259-260
 - JavaScript integration, 261
- complete list of skill types, creating, 224
- contacts drop-down list, creating, 224
- controllers
 - creating, 225-227
 - tests, 229-231
- data security, 224-225
- page, creating, 224
- requirements, 223
- sample implementation, 223
- skills list, creating, 224
- Visualforce page, 228-229

SmartSync

- library, 266
- mobile Web applications data access, 270

SOAP (Simple Object Access Protocol), 301

- API, 31
 - enabled permissions, 318-319
 - Enterprise. *See* Enterprise API
 - error handling, 322
 - Force.com data types, 321
 - IP white-listing, 319
 - limits, 316
 - logging in/out, 318-320
 - login call, 320
 - login problems, troubleshooting, 320
 - Partner, 315
 - security, 316
 - security tokens, 319
 - stub code, generating, 316-317
 - Web Service Connector (WSC), 316
 - WSDL versions, 315-316
- custom Apex SOAP Web services, 326
 - Apex class rules, compared, 327
 - calling, 328
 - creating records example, 328
 - governor limits, 327
 - invoking, 329
 - limitations, 326-327

Services Manager anonymous

- benchmarking, 333-335

- services, calling from Apex, 305-306

subjects service

- record retrieval

- external identifiers, 310

- unique identifiers, 309

- SObject basic information request, 309

social applications. See Chatter**Software as Service. See PaaS****SOQL (Salesforce Object Query Language), 26-27**

- aggregate queries, 144

- aggregate functions, 144-145

- grouping records, 145-146

- grouping records with subtotals, 147-148

- Chatter queries

- comments, 375

- custom object, 373

- feed-tracked changes, 376

- following relationships, 377

- standard object, 372

- user feed, 374

- dirty writes, 161

- joins

- anti-joins, 152-153

- inner, 149-150

- outer, 148-149

- semi-joins. *See* semi-joins

- multi-select picklists, 154

- queries

- Apex, 126-128

- child-to-parent, 125-126

- dynamic, 175-176

- example, 26-27

- parent-to-child, 126, 151

- PushTopics, 340

- record retrieval, 323

- record sharing, 165-166

- relationships, 125-126

- REST API request, 310

- results, sorting, 125

- Schema Explorer, 123

- records

- filter conditions, 124-125

- limits, 125

- SQL, compared, 27-28
 - column list functions, 28
 - governor limits, 28
 - implicit joins, 27
 - nested resultsets, 27-29
- statements, 124
 - filter conditions, 124
 - record limits, 125
 - sort fields, 125

sorting

- lists/arrays, 112
- query results, 125

SOSL (Salesforce Object Search Language), 29

- Apex, 155-157
- dirty writes, 161
- queries
 - dynamic, 176
 - example, 29
- record retrieval, 324
- syntax, 155
 - column list functions, 28
 - governor limits, 28
 - implicit joins, 27
 - nested resultsets, 27-29

SQL versus SOQL, 27-28**Staffing Coordinator profile**

- permissions, 86
- Services Manager application, 18
- testing, 96-97

standard buttons

- custom objects, creating, 37
- listing of, 213
- overriding, 213

standardController attribute, 200**standard controllers, 191-193**

- multiple records, 192-193
- single records, 191-192

standard fields

- custom objects, creating, 36
- defined, 23

standard links, 37**standard objects, 22****standard pages**

- edit, 211
- list, 211
- overriding, 210-211
- tab, 210
- view, 211

standard set controllers, 192-193**start method (Batchable interface), 283****stateful Batch Apex, 289-290****Stateful interface, 290****statements**

- conditional, 113
- Delete, 130
- DML. *See* DML, statements
- exception, 114-115
 - examples, 115
 - handling, 115
 - raising, 115
- Insert, 129
- loops, 114
- SOQL, 124
 - filter conditions, 124
 - record limits, 125
 - sort fields, 125
- Undelete, 130
- Update, 129
- Upsert, 129-130

static resources, 241-242**status**

- bulk export jobs, 348
- bulk import jobs, 347
- messages, displaying, 238
 - dynamic, 239
 - images/stylized, 239

Status update Chatter posts, 371**storage custom settings, 47-48**

- defined, 47
- hierarchy, 49
- limits, 49
- list, 48
- types, 47-48

Streaming API

- example, 341-344
 - CometD library, 342
 - Visualforce controller, 342
 - Visualforce page, 343
- PushTopics, 340-341
 - availability, 341
 - components, 340-341
 - deleting, 341
 - limitations, 341
- Web site, 340

strings

- concatenation (+) operator, 110
- converting to dates, 109

- date conversions, 109
- defined, 106
- ID conversion, 108
- structural components (Mobile Components for Visualforce), 268**
- stub code, generating, 316-317**
- stylesheet component, 205**
- subtraction (-) operator, 110**
- SUM aggregate function, 144-145**
- systems integrators, 17**

T

- table components, 205**
- tables. See objects**
- tabs**
 - creating, 41, 63, 215
 - page, 210
- targetObjectIds unique identifiers**
 - email templates, 169
 - MassEmailMessage object, 170
- teams, selecting, 11-12**
- technical support, 17**
- technology integrations, 4**
- templatelds unique identifiers, 170**
- templates**
 - AngularJS, 253
 - Mobile Design, 269
 - sending email, 169-170
 - Visualforce pages as, 243-244
- testAsUser method, 231**
- testing**
 - anonymous benchmarking Web service, 334
 - Batch Apex, 293
 - REST services integration, 304
 - Services Manager application, 97-98
 - Follow Team button, 385
 - mobile timecard entry page, 279
 - Services Manager security, 94-98
 - additional users, creating, 94-95
 - Consultant profile, 96
 - data preparation, 95-96
 - Staffing Coordinator profile, 96-97
 - Vice President profile, 97
 - unit tests
 - Apex. *See* Apex, unit tests
 - integrated, 14-15
 - missing timecard report, 299-300

- Skills Matrix controllers, 229-231
- TimecardManager class, 140-141
- Visualforce controllers, 222
- test methods (Apex), 136**
- testNoContactForUser method, 231**
- testNoContactSelected method, 231**
- testNoSkills method, 231**
- testSave method, 231**
- testWithSkills method, 231**
- text**
 - Chatter posts, 371
 - fields, 38
 - SOAP data type, mapping, 322
- Text Area data type, 322**
- throw keyword (exceptions), 115**
- time data type, 38, 106**
- TimecardManager class**
 - creating, 138-139
 - unit tests, 140-141
- Timecard object**
 - fields, 56
 - overview, 53
- timed events, 237**
- Tooling API, 354**
 - Apex code, deploying, 355
 - internal state of deployment, 355
 - overview, 355
 - query service, 355
 - status, refreshing, 355
 - user interface, 356
 - Visualforce examples
 - controller, 357-359
 - page, 359-360
 - Web site, 355
- tools**
 - cURL, 306
 - custom objects, 33-34
 - App Builder Tools, 33
 - data, 34
 - Force.com IDE, 34
 - metadata, 33
 - Schema Builder, 34
 - data, 34
 - Data Loader, 34
 - Excel Connector, 34
 - Import Wizard, 34
 - Data Loader
 - data preparation, 64-66
 - data verification, 67-69
 - importing data, 66

- Schema Builder
 - custom objects, creating, 59-60
 - Lookup relationships, creating, 60
 - Master-Detail relationships, creating, 60-62
- Visualforce development, 188-190
- Web Service Connector, 316

tracking email, 172**transaction processing**

- DML database methods, 157-158
 - insert example, 158
 - opt_allOrNone parameter, 158
- record locking, 161
- savepoints, 159-160
 - example, 159-160
 - limitations, 159
 - restoring to, 159
 - setting, 159

transactions

- Batch Apex, 283
- custom Apex SOAP Web services, 327

triggers, 130-131

- batching, 132
- bulkifying, 132
- custom objects, creating, 37
- definitions, 131-132
- email notifications, 181-182
- error handling, 132-133
- names, 131
- page navigation, 195
- timecard validation, creating, 138-139

troubleshooting SOAP API login problems, 320**try keyword (exceptions), 115****tuning Visualforce user interfaces, 217-218****TypeException exception, 114**

U

- unary decrement (-) operator, 110**
- unary increment (++) operator, 110**
- uncaught exceptions, 220**
- undelete service, 325**
- Undelete statement, 130**
- undeleting records, 130, 325**
- unfollowing records, 377-378**
- unique identifiers**
 - email templates, 169
 - fields, 24

- mass emails, 170
- organization-wide email addresses, 172
- record retrieval, 309, 324

unit tests

- Apex, 136
 - results, viewing, 137
 - running, 137
 - test data, 137
 - test methods, 136
 - Test Runner View, 103
- integrated, 14-15
- missing timecard report, 299-300
- Skills Matrix controllers, 229-231
- TimecardManager class, 140-141
- Visualforce controllers, 222

UNIX line-continuation character (\), 309**unsigned shift right (>>) operator, 110****update service, 325****Update statement, 129****updating**

- custom setting records, 180
- records, 129
 - Enterprise API, 325
 - requests, 311

upserting records

- Enterprise API, 325
- requests, 311

upsert service, 325**Upsert statement, 129-130****URLs**

- Chatter posts, 371
- fields, 38
- SOAP data type, mapping, 322

user feeds (Chatter posts), 374**user interfaces**

- Apex Test Runner View, 103
- custom, creating. *See* Visualforce designer contributions, 12
- Enhanced Profile, 74
- jQuery, 259
- modularity, 119
- native. *See* native user interface services, 8
- Tooling API example, 356

UserOrGroupId field, 162**userPhotoUpload component, 381****users**

- authentication
 - Bulk API, 345-346

- Canvas, 349-350
 - mobile Web applications, 269-270
 - REST APIs, 306-307
 - sites, 258
- creating, 94-95
- groups, 80
 - public, 80
 - roles, 80
- permission sets, 72
- profiles, 74
 - administrative permissions, 75
 - defined, 72
 - Enhanced Profile List Views, 74
 - Enhanced Profile User Interface, 74
 - field-level security, 78
 - licenses, 76
 - object permissions, 75-76
 - Services Manager, 85-86, 89-90
 - types, 74
- roles (Services Manager application), 18, 91-92
 - accounts receivable, 18
 - consultants, 18
 - project managers, 18
 - sales representatives, 18
 - staffing coordinators, 18
 - Vice President, 18

V

- validateTimecard trigger, 131**
- validation rules**
 - custom objects, 37
 - fields, 24
 - Skill object, creating, 63
- valueOf method**
 - date to string conversions, 109
 - string to date conversions, 109
- variables, 105**
 - access modifiers, 118
 - checkpoints, 133-135
 - classes, 117
 - constants, 107
 - data types, 106
 - blob, 106
 - Boolean, 106
 - converting, 107-108
 - converting dates to strings, 109
 - converting strings to dates, 109
 - date, 106
 - datetime, 106
 - decimal, 106
 - double, 106
 - ID, 106
 - Integer, 106
 - long, 106
 - object, 106
 - string, 106
 - time, 106
 - declaring, 105-106
 - enums, 107
 - fields, 247
 - names, 105-106
 - rounding, 108
 - selectedContactId, 225
- verifying data imports, 67-69**
- Vice President profile**
 - permissions, 86
 - Services Manager application, 18
 - testing, 97
- View All permission, 76**
- view components (Visualforce), 198**
 - action, 203-204
 - attributes, 199
 - Chatter support, 380-382
 - feed, 381
 - feedWithFollowers, 381
 - follow, 381
 - followers, 381
 - limitations, 382
 - newsFeed, 381
 - userPhotoUpload, 381
 - component body, 199
 - custom, 244-246
 - CompareSkillsComponent, creating, 259-260
 - CSS, adding, 261
 - defining, 244-245
 - Google Map example, 245-246
 - support, adding, 261
 - data, 200-203
 - metadata-aware, 200-201
 - primitive, 201-202
 - repeating, 201-203
 - facets, 239

- Force.com-styled, 204-205
 - action containers, 205
 - notifications, 205
 - page structure, 205
 - paging, 205
 - sample controller, 206
 - sample page, 207
 - table, 205
- identifier problems, debugging, 240
- Mobile Components for Visualforce, 268-269
 - documentation/source code Web site, 269
 - installing, 268-269
 - types, 268
- names, 199
- native user interface, 208-209
 - detail, 209
 - enhancedList, 209
 - listViews, 208
 - relatedList, 209
- page, 200
- primitive, 204-205
- referencing from JavaScript, 240
- syntax, 198-199
- visibility, 200
- viewing**
 - batch jobs execution detail, 288
 - fields, 64
 - mobile timecard entry pages
 - iPhones, 273
 - Web browsers, 273
 - relationships, 121
 - scheduled batch jobs, 296
 - sharing rules, 163, 167
 - unit test results, 137
 - Visualforce pages
 - native user interface buttons, 213
 - Salesforce Classic, 271
 - Salesforce Touch, 271
- view page, 211**
- Views, browsing data, 43-44**
 - Apex Test Runner, 103
 - Execute Anonymously, 104-105
 - Problems, 103
- view state, preserving, 195**
- Visualforce**
 - architecture, 186-187
 - asynchronous actions
 - Ajax support, 234
 - as JavaScript events, 237-238
 - as JavaScript functions, 235-236
 - partial page refreshes, 234-235
 - status messages, 238-240
 - as timed events, 237
 - Chatter components, 380-382
 - feed, 381
 - feedWithFollowers, 381
 - follow, 381
 - followers, 381
 - limitations, 382
 - newsFeed, 381
 - userPhotoUpload, 381
 - controllers, 186-187
 - actionFunction component, 236
 - AngularJS project list example, 252
 - Chatter example, 378
 - custom, 193-197
 - dynamic field reference, 247
 - editing mobile timecards, 277
 - extensions, 197
 - governor limits, 221
 - mobile timecard list
 - functionality, 274
 - partial page refresh, 235
 - Services Manager Follow Team
 - button extension code, 383-384
 - Services Manager Skills Matrix, 225-227, 229-231
 - standard, 191-193
 - Streaming API example, 342
 - unit tests, 222
 - debugging, 216
 - development
 - process, 188
 - tools, 188-190
 - dynamic, 246
 - component generation, 248-249
 - dynamic field references, 246-248
 - error handling, 220-221
 - communication, 220-221
 - uncaught exceptions, 220
 - Hello World example, 189-191
 - Mobile Components, 268-269
 - documentation/source code Web site, 269

- installing, 268-269
- types, 268
- modular, 241
 - composition, 243-244
 - custom components, 244-246
 - inclusion, 242
 - static resources, 241-242
- native user interface integration, 209-210
 - custom buttons/links, 215
 - custom tabs, 215
 - page layouts, 213
 - standard buttons, 213
 - standard pages, 210-211
- overview, 186
- pages, 187
 - actionFunction component, 236
 - adding to page layouts, 213
 - adding to Salesforce Touch, 271
 - AngularJS example code, 252-253
 - Chatter components, 381
 - Chatter example, 378
 - dynamic, 246-249
 - JavaScript events, 237-238
 - mobile timecards, 275-276, 278
 - native user interface buttons/links
 - navigation, 215
 - as native user interface tabs, 215
 - performance tuning, 217-218
 - security, 218-220
 - Services Manager Follow Team
 - button, 384-385
 - Services Manager Skills Matrix, 228-229
 - Services Manager Utilization, 337-338
 - Services Manager Utilization page,
 - creating, 330-331
 - standard pages, overriding, 210-211
 - Streaming API example, 343
 - timed events, 237
 - Tooling API example, 359-360
 - viewing from native user interface
 - buttons, 213
 - viewing in Salesforce Classic, 271
- performance, tuning, 217-218
- public access. *See* sites
- security, 218
 - object-level, 218
 - page-level, 219
 - record-level, 219
- Services Manager application
 - business hours, configuring, 331
 - hours utilization calculation, 332
 - utilization controller code, 335-337
 - Utilization page, 330-331, 337-338
- Streaming API page, 343
- Tooling API example
 - controller, 357-359
 - page, 359-360
- view components, 198
 - action, 203-204
 - attributes, 199
 - component body, 199
 - custom. *See* custom components
 - data, 200-203
 - facets, 239
 - Force.com-styled, 205-208
 - identifier problems, debugging, 240
 - names, 199
 - native user interface, 208-209
 - page, 200
 - primitive, 204-205
 - referencing from JavaScript, 240
 - syntax, 198-199
 - visibility, 200

W

- web developer contributions, 12**
- Web development frameworks, 268-269**
 - Mobile Components for Visualforce, 268-269
 - documentation/source code Web site, 269
 - installing, 268-269
 - types, 268
 - Web MVC, 269
- Web servers, configuring, 352**
- Web services**
 - Connector (WSC), 316
 - Description Language. *See* WSDL
 - integration, 9

Web sites

- AJAX Proxy, 270
- AngularJS, 251
- anonymous benchmark WSDL, 333
- Apex Code Developer's Guide, 100, 108
- AppExchange, 16
- Bulk API, 345
- Canvas, 349
- Chatter
 - Apex, 378
 - REST API, 379
- Code Share, 16
- CometD library, 342
- cURL, 306
- Data Loader Mac OS X version, 34
- DE account registration, 32
- developer discussion boards, 16
- Developer Force, 16
- Dreamforce/Cloudforce conferences, 17
- Excel Connector, 34
- expressions, scheduling, 295
- Force.com IDE, 34
- Ideas, 16
- IDE installation, 101
- jQuery UI, 259
- Large Data Volume (LDV)
 - deployments, 22
- Metadata API, 360
- Mobile Components for Visualforce, 269
- Mobile Packs, 269
- multitenancy whitepaper, 5
- NAICS codes, 333
- OAuth, 307
- REST API Developer's Guide, 308
- Salesforce
 - Classic implementation guide, 271
 - Mobile SDK, 265, 267
- Secure Coding Guideline document, 218
- security resources, 385
- SimpleDateFormat pattern, 109
- SOAP Partner API, 315
- Streaming API, 340
- systems integrators, 17
- Tooling API, 355
- Visualforce pages, performance
 - tuning, 218
- Web Service Connector, 316
- Yahoo! geocoding REST service, 303

whatIds unique identifiers

- email templates, 169
- MassEmailMessage object, 170

While loops, 114

Windows Azure, 3

wizards

- Import, 34
- New Custom Field
 - default values, 39
 - descriptions, 39
 - external IDs, 39
 - help text, 39
 - labels, 39
 - names, 39
 - required fields, 39
 - types, selecting, 38
 - unique fields, 39
- New Custom Object Tab, 63

wrapper patterns, 195-196

write locks, 161

WSC (Web Service Connector), 316

WSDL (Web Services Description Language)

- Services Manager anonymous
 - benchmark, 333
- stub code, generating, 316-317
- versions, 315-316

X

XML metadata, 30-31

XOR (^) operator, 110

Y

Yahoo! geocoding REST service

- integrating, 303
- invoking, 303
- testing, 304