Jason Ouellette

Third Edition

Development with the Force.com Platform

Building Business Applications in the Cloud

Developer's Library

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

Library of Congress Control Number: 2013950238

Visit us on the Web: informit.com/aw

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

Table of Contents

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!

Email: laura.lewin@pearson.com

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

4 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.

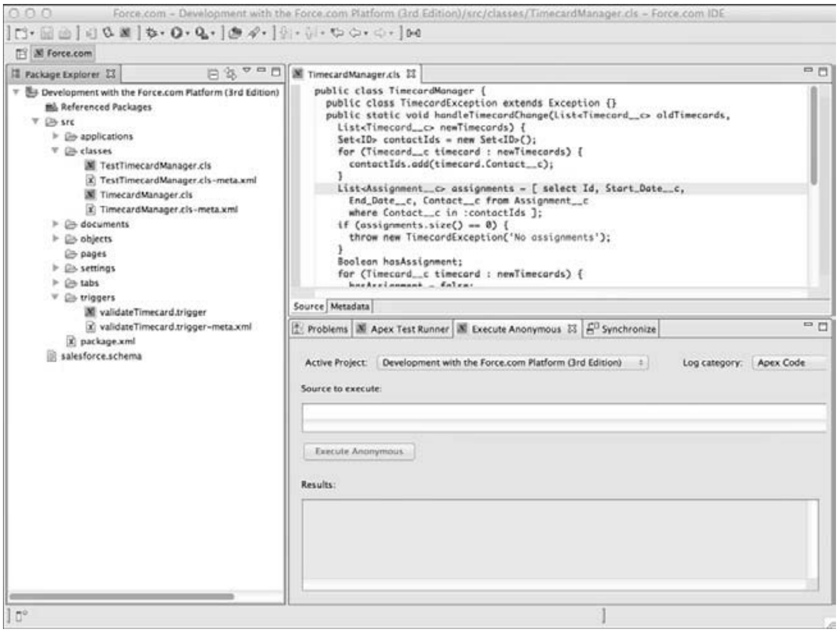
Visit http://wiki.developerforce.com/index.php/Apex_Toolkit_for_Eclipse to learn how to install the stand-alone and plug-in versions of the Force.com IDE.

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.


Force.com – Development with the Force.com Platform (3rd Edition)/src/classes/TimecardManager.cls – Force.com IDE

Package Explorer

```
Development with the Force.com Platform (3rd Edition)
  Referenced Packages
  src
    applications
    classes
      TestTimecardManager.cls
      TestTimecardManager.cls-meta.xml
      TimecardManager.cls
      TimecardManager.cls-meta.xml
    documents
    objects
    pages
    settings
    tabs
    triggers
      validateTimecard.trigger
      validateTimecard.trigger-meta.xml
    package.xml
  salesforce.schema
```

TimecardManager.cls

```
public 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;
```

Active Project: Development with the Force.com Platform (3rd Edition)

Log category: Apex Code

Figure 4.1 Force.com perspective

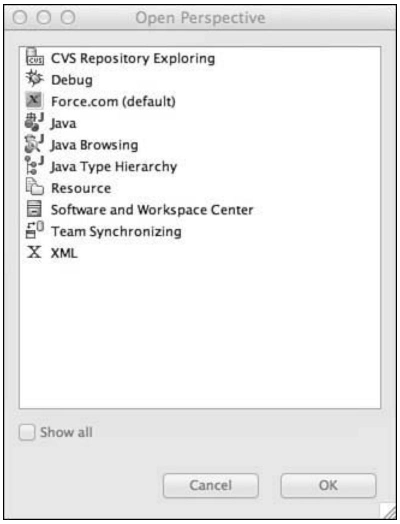


Figure 4.2 Open Perspective dialog

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

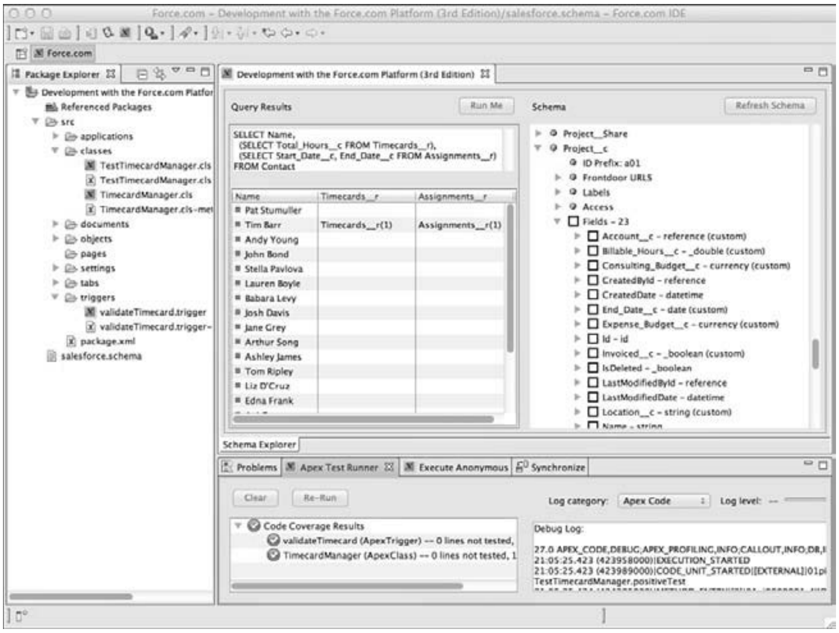
The Force.com IDE leverages the standard Eclipse View called Problems to display compilation errors. When you save changes to an object in a Force.com Project, it is sent over the network to the Force.com service for compilation. If compilation fails, Force.com-specific errors are added to the Problems 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.


Force.com - Development with the Force.com Platform (3rd Edition)/salesforce.schema - Force.com IDE
Package Explorer
Development with the Force.com Platfor
Referenced Packages
src
applications
classes
TestTimecardManager.cls
TestTimecardManager.cls
TimecardManager.cls
TimecardManager.cls-met
documents
objects
pages
settings
tabs
triggers
validateTimecard.trigger
validateTimecard.trigger-
package.xml
salesforce.schema
Development with the Force.com Platform (3rd Edition)
Query Results
Run Me
Schema
Refresh Schema
Project__Share
Project__c
ID Prefix: a01
Frontdoor URLS
Labels
Access
Fields - 23
Account__c - reference (custom)
Billable_Hours__c - _double (custom)
Consulting_Budget__c - currency (custom)
CreatedById - reference
CreatedDate - datetime
End_Date__c - date (custom)
Expense_Budget__c - currency (custom)
Id - id
Invoiced__c - _boolean (custom)
IsDeleted - _boolean
LastModifiedById - reference
LastModifiedDate - datetime
Location__c - string (custom)
Name - string
Schema Explorer
Problems
Apex Test Runner
Execute Anonymous
Synchronize
Clear
Re-Run
Log category:
Apex Code
Log level:
Code Coverage Results
validateTimecard (ApexTrigger) -- 0 lines not tested,
TimecardManager (ApexClass) -- 0 lines not tested, 1

```
SELECT Name,
  (SELECT Total_Hours__c FROM Timecards__r),
  (SELECT Start_Date__c, End_Date__c FROM Assignments__r)
FROM Contact
```

Name	Timecards__r	Assignments__r
Pat Stumuller		
Tim Barr	Timecards__r(1)	Assignments__r(1)
Andy Young		
John Bond		
Stella Pavlova		
Lauren Boyle		
Babara Levy		
Josh Davis		
Jane Grey		
Arthur Song		
Ashley James		
Tom Ripley		
Liz D'Cruz		
Edna Frank		

```
Debug Log:

27.0 APEX_CODE,DEBUG;APEX_PROFILING,INFO;CALLOUT,INFO;DB,I
21:05:25.423 (423958000)|EXECUTION_STARTED
21:05:25.423 (423989000)|CODE_UNIT_STARTED|[EXTERNAL]|01p
TestTimecardManager.positiveTest
```

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 (`m`, Integer) and fractional components (`n`), expressed as `m.n`. 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 `StringException` 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 `TypeException`. 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
`+, -, *, /`	Integer, Long, Double, Decimal	4	Numeric add, subtract, multiply, divide
`!`	Boolean	2	Logical negation
`-`	Integer, Long, Double, Decimal	2	Arithmetic negation
`++, --`	Integer, Long, Double, Decimal	1	Unary increment, decrement
`&, |, ^`	Integer, Long, Boolean	10	Bitwise AND, OR, XOR
`<<, >>, >>>`	Integer, Long	10	Signed shift left, signed shift right, unsigned shift right
`==, <, >, <=, >=, !=`	Any compatible types	5 (<, >, <=, >=), 6 (==, !=)	Not case sensitive, locale-sensitive comparisons: equality, less than, greater than, less than or equal to, greater than or equal to, not equal to
`&&, ||`	Boolean	7 (&&), 8 (||)	AND, OR, with short-circuiting behavior (second argument is not evaluated if first argument is sufficient to determine result)
`===, !==`	Map, List, Set, Enum, SObject	N/A	Exact equality, exact inequality
`()`	Any	1	Group an expression and increase its precedence
`? :`	Boolean	N/A	Shortcut for `if/then/else` 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	`do { code_block } while (condition);`	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	`while (condition) { code_block; }`	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	`for (init; exit condition; increment) { code_block; }`	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	`for (var : list/set) { code_block }`	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**

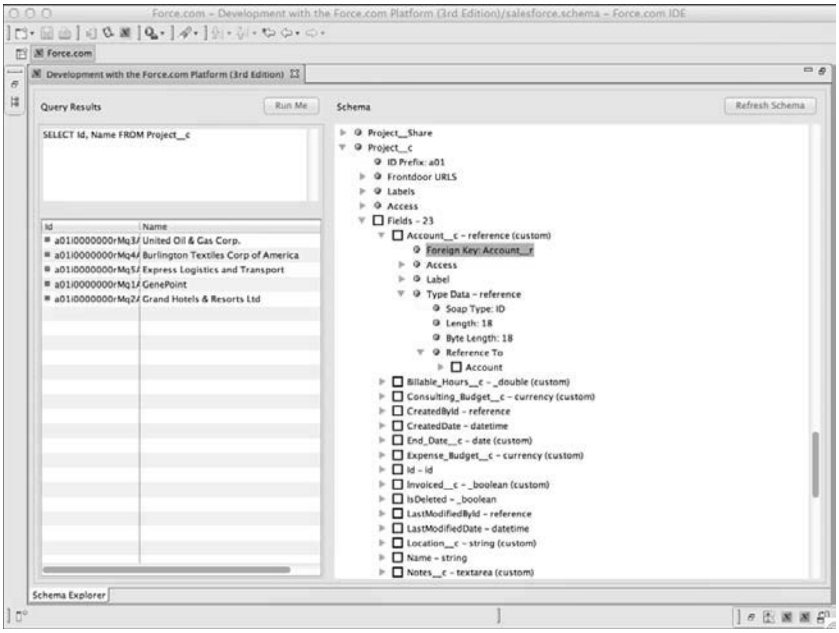
```
Contact contact = new Contact();
contact.FirstName = 'Larry';
```

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.


Force.com - Development with the Force.com Platform (3rd Edition)/salesforce.schema - Force.com IDE
Force.com
Development with the Force.com Platform (3rd Edition)
Query Results
Run Me
Schema
Refresh Schema
Schema Explorer

```
SELECT Id, Name FROM Project__c
```

Id	Name
a01i0000000rMq3A	United Oil & Gas Corp.
a01i0000000rMq4A	Burlington Textiles Corp of America
a01i0000000rMq5A	Express Logistics and Transport
a01i0000000rMq1A	GenePoint
a01i0000000rMq2A	Grand Hotels & Resorts Ltd

```
Project__Share
Project__c
    ID Prefix: a01
    Frontdoor URLS
    Labels
    Access
    Fields - 23
        Account__c - reference (custom)
            Foreign Key: Account__r
            Access
            Label
            Type Data - reference
                Soap Type: ID
                Length: 18
                Byte Length: 18
                Reference To
                    Account
        Billable_Hours__c - _double (custom)
        Consulting_Budget__c - currency (custom)
        CreatedById - reference
        CreatedDate - datetime
        End_Date__c - date (custom)
        Expense_Budget__c - currency (custom)
        Id - id
        Invoiced__c - _boolean (custom)
        IsDeleted - _boolean
        LastModifiedById - reference
        LastModifiedDate - datetime
        Location__c - string (custom)
        Name - string
        Notes__c - textarea (custom)
```

Figure 4.4 Viewing relationships in Schema Explorer

Data integrity is protected in Apex at compile and runtime using object metadata. For example, `Name` is defined as a read-only field in Contact, so the code in Listing 4.25 cannot be compiled.

Listing 4.25 **Attempted Assignment to Read-Only Field**

```
Contact c = new Contact();
c.Name = 'Larry';
```

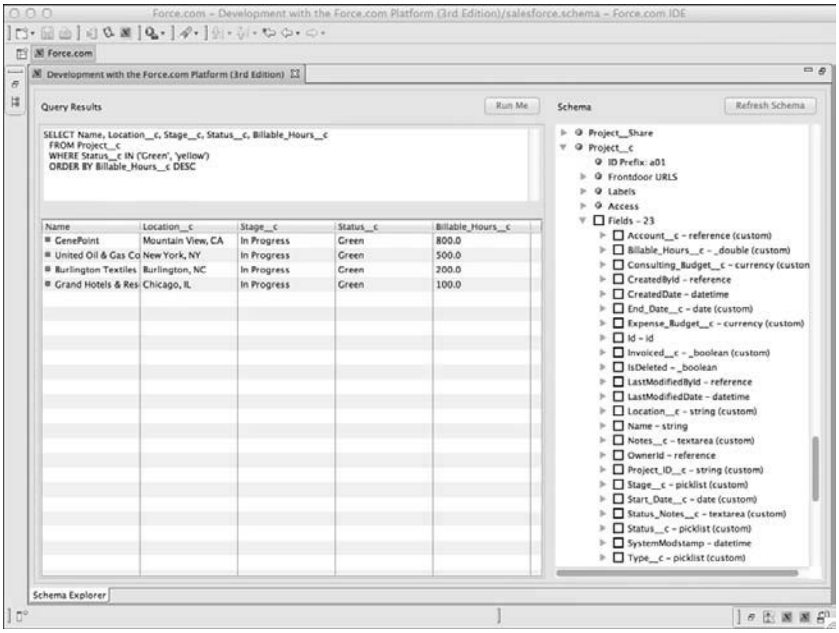
After a database object is referenced in Apex code, that object cannot be deleted or edited in a way that invalidates the code. This protects your code from changes to the database schema. Impacted code must be commented out before the database objects are modified.

Database Queries

You've seen how data structures in Apex are implicitly defined by the objects in your database. Force.com provides two query languages to populate these objects with data: Salesforce Object Query Language (SOQL) and Salesforce Object Search Language (SOSL). SOSL, addressed in Chapter 5, "Advanced Business Logic," provides unstructured, full-text search across many objects from a single query.

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.


Force.com – Development with the Force.com Platform (3rd Edition)/salesforce.schema – Force.com IDE
Force.com
Development with the Force.com Platform (3rd Edition)
Query Results
Run Me
Schema
Refresh Schema
Schema Explorer

```
SELECT Name, Location__c, Stage__c, Status__c, Billable_Hours__c
  FROM Project__c
  WHERE Status__c IN ('Green', 'yellow')
  ORDER BY Billable_Hours__c DESC
```

Name	Location__c	Stage__c	Status__c	Billable_Hours__c
GenePoint	Mountain View, CA	In Progress	Green	800.0
United Oil & Gas Co	New York, NY	In Progress	Green	500.0
Burlington Textiles	Burlington, NC	In Progress	Green	200.0
Grand Hotels & Res	Chicago, IL	In Progress	Green	100.0

```
Project__Share
Project__c
    ID Prefix: a01
    Frontdoor URLS
    Labels
    Access
    Fields - 23
        Account__c - reference (custom)
        Billable_Hours__c - _double (custom)
        Consulting_Budget__c - currency (custon
        CreatedById - reference
        CreatedDate - datetime
        End_Date__c - date (custom)
        Expense_Budget__c - currency (custom)
        Id - id
        Invoiced__c - _boolean (custom)
        IsDeleted - _boolean
        LastModifiedById - reference
        LastModifiedDate - datetime
        Location__c - string (custom)
        Name - string
        Notes__c - textarea (custom)
        OwnerId - reference
        Project_ID__c - string (custom)
        Stage__c - picklist (custom)
        Start_Date__c - date (custom)
        Status_Notes__c - textarea (custom)
        Status__c - picklist (custom)
        SystemModstamp - datetime
        Type__c - picklist (custom)
```

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."

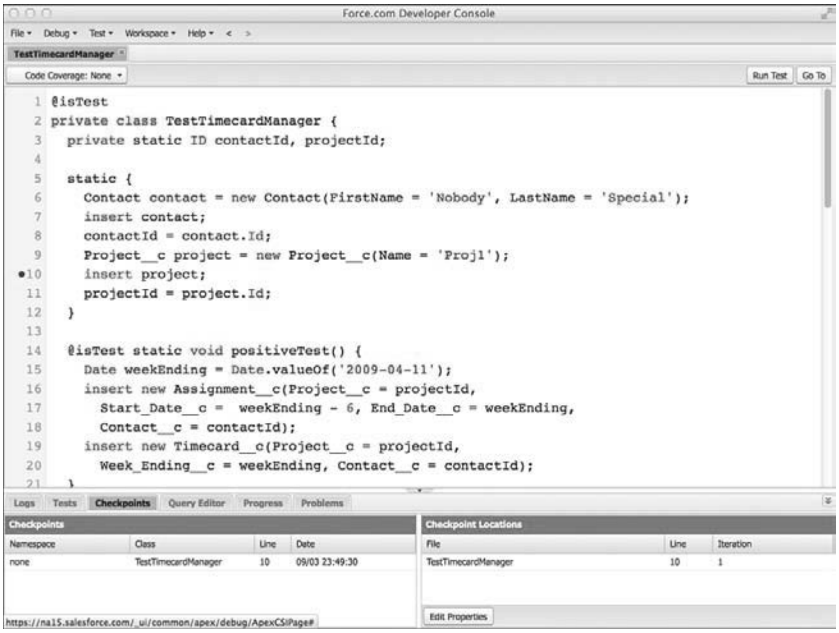
Debugging Apex Using Developer Console

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.


Force.com Developer Console
File ▾ Debug ▾ Test ▾ Workspace ▾ Help ▾ < >
TestTimecardManager
Code Coverage: None ▾
Run Test
Go To
Logs
Tests
Checkpoints
Query Editor
Progress
Problems
Edit Properties
https://na15.salesforce.com/_ui/common/apex/debug/ApexCSIPage#

```
  1 @isTest
  2 private class TestTimecardManager {
  3   private static ID contactId, projectId;
  4 
  5   static {
  6     Contact contact = new Contact(FirstName = 'Nobody', LastName = 'Special');
  7     insert contact;
  8     contactId = contact.Id;
  9     Project__c project = new Project__c(Name = 'Proj1');
 •10     insert project;
 11     projectId = project.Id;
 12   }
 13 
 14   @isTest static void positiveTest() {
 15     Date weekEnding = Date.valueOf('2009-04-11');
 16     insert new Assignment__c(Project__c = projectId,
 17       Start_Date__c =  weekEnding - 6, End_Date__c = weekEnding,
 18       Contact__c = contactId);
 19     insert new Timecard__c(Project__c = projectId,
 20       Week_Ending__c = weekEnding, Contact__c = contactId);
 21   }
```

Checkpoints

Namespace	Class	Line	Date
none	TestTimecardManager	10	09/03 23:49:30

Checkpoint Locations

File	Line	Iteration
TestTimecardManager	10	1

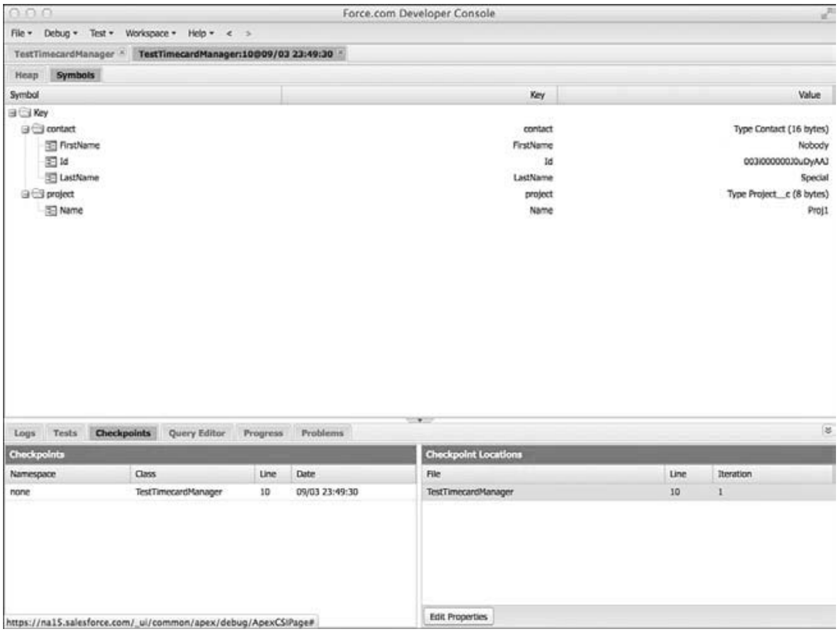
Figure 4.6 Setting a heap dump location

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.


Force.com Developer Console
File Debug Test Workspace Help
TestTimecardManager
TestTimecardManager:10@09/03 23:49:30
Heap
Symbols
Logs
Tests
Checkpoints
Query Editor
Progress
Problems
Edit Properties
https://na15.salesforce.com/_ui/common/apex/debug/ApexCSIPage#

Symbol	Key	Value
Key		
contact	contact	Type Contact (16 bytes)
FirstName	FirstName	Nobody
Id	Id	003i000000J0uDyAAJ
LastName	LastName	Special
project	project	Type Project__c (8 bytes)
Name	Name	Proj1

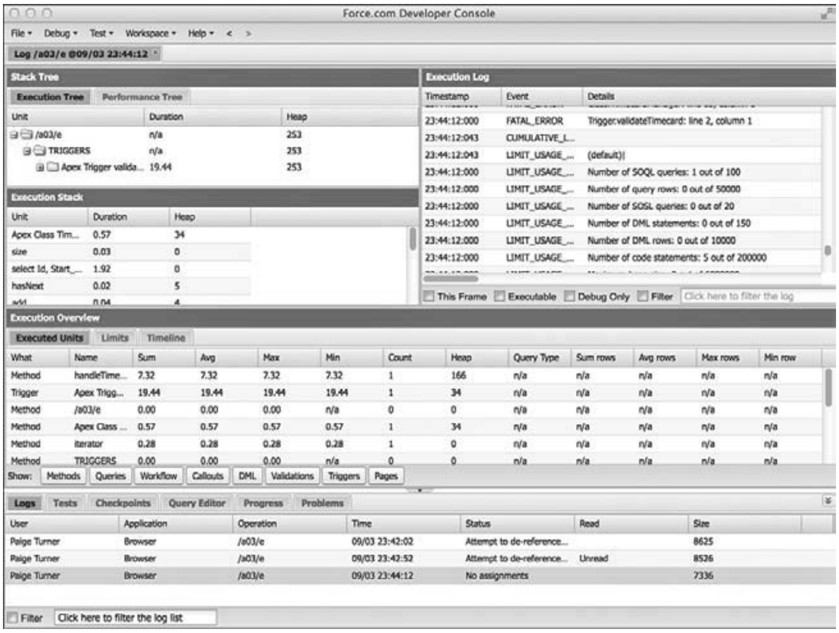
Checkpoints

Namespace	Class	Line	Date
none	TestTimecardManager	10	09/03 23:49:30

Checkpoint Locations

File	Line	Iteration
TestTimecardManager	10	1

Figure 4.7 Examining a heap dump


Force.com Developer Console
File Debug Test Workspace Help
Log /a03/e @09/03 23:44:12
Stack Tree
Execution Tree
Performance Tree
Execution Stack
Execution Log
This Frame
Executable
Debug Only
Filter
Click here to filter the log
Execution Overview
Executed Units
Limits
Timeline
Show: Methods Queries Workflow Callouts DML Validations Triggers Pages
Logs
Tests
Checkpoints
Query Editor
Progress
Problems
Filter
Click here to filter the log list

Stack Tree

Unit	Duration	Heap
/a03/e	n/a	253
TRIGGERS	n/a	253
Apex Trigger valida...	19.44	253

Execution Stack

Unit	Duration	Heap
Apex Class Tim...	0.57	34
size	0.03	0
select Id, Start_...	1.92	0
hasNext	0.02	5
add	0.04	4

Execution Log

Timestamp	Event	Details
23:44:12:000	FATAL_ERROR	Trigger.validateTimecard: line 2, column 1
23:44:12:043	CUMULATIVE_L...	
23:44:12:043	LIMIT_USAGE_...	(default)|
23:44:12:000	LIMIT_USAGE_...	Number of SOQL queries: 1 out of 100
23:44:12:000	LIMIT_USAGE_...	Number of query rows: 0 out of 50000
23:44:12:000	LIMIT_USAGE_...	Number of SOSL queries: 0 out of 20
23:44:12:000	LIMIT_USAGE_...	Number of DML statements: 0 out of 150
23:44:12:000	LIMIT_USAGE_...	Number of DML rows: 0 out of 10000
23:44:12:000	LIMIT_USAGE_...	Number of code statements: 5 out of 200000

Execution Overview

What	Name	Sum	Avg	Max	Min	Count	Heap	Query Type	Sum rows	Avg rows	Max rows	Min row
Method	handleTime...	7.32	7.32	7.32	7.32	1	166	n/a	n/a	n/a	n/a	n/a
Trigger	Apex Trigg...	19.44	19.44	19.44	19.44	1	34	n/a	n/a	n/a	n/a	n/a
Method	/a03/e	0.00	0.00	0.00	n/a	0	0	n/a	n/a	n/a	n/a	n/a
Method	Apex Class ...	0.57	0.57	0.57	0.57	1	34	n/a	n/a	n/a	n/a	n/a
Method	iterator	0.28	0.28	0.28	0.28	1	0	n/a	n/a	n/a	n/a	n/a
Method	TRIGGERS	0.00	0.00	0.00	n/a	0	0	n/a	n/a	n/a	n/a	n/a

User	Application	Operation	Time	Status	Read	Size
Paige Turner	Browser	/a03/e	09/03 23:42:02	Attempt to de-reference...		8625
Paige Turner	Browser	/a03/e	09/03 23:42:52	Attempt to de-reference...	Unread	8526
Paige Turner	Browser	/a03/e	09/03 23:44:12	No assignments		7336

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.

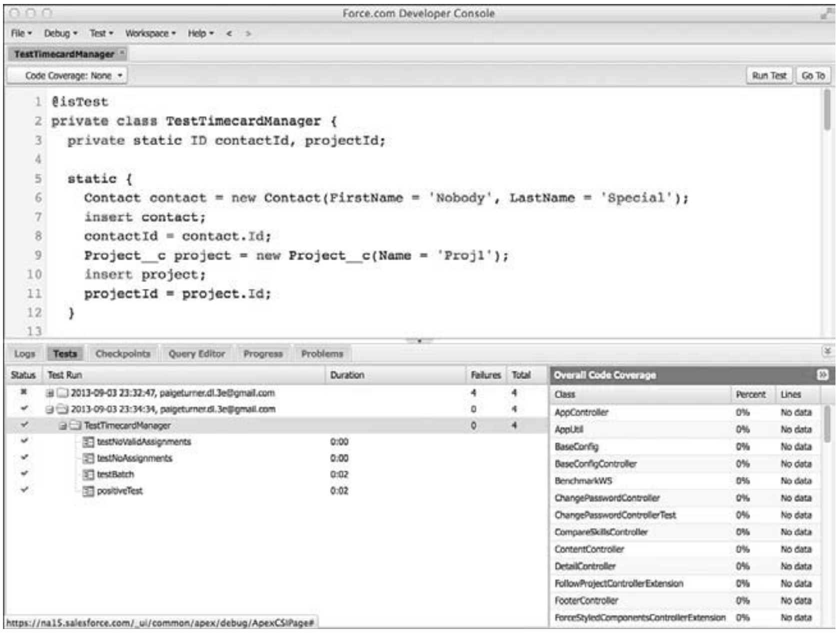
Test Data

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.


Force.com Developer Console
File Debug Test Workspace Help
TestTimecardManager
Code Coverage: None
Run Test
Go To
Logs
Tests
Checkpoints
Query Editor
Progress
Problems
https://na15.salesforce.com/_ui/common/apex/debug/ApexCSIPage#

```
@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;
  }

```

Status	Test Run	Duration	Failures	Total
✗	2013-09-03 23:32:47, paigeturner.dl.3e@gmail.com		4	4
✓	2013-09-03 23:34:34, paigeturner.dl.3e@gmail.com		0	4
✓	TestTimecardManager		0	4
✓	testNoValidAssignments	0:00		
✓	testNoAssignments	0:00		
✓	testBatch	0:02		
✓	positiveTest	0:02		

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.

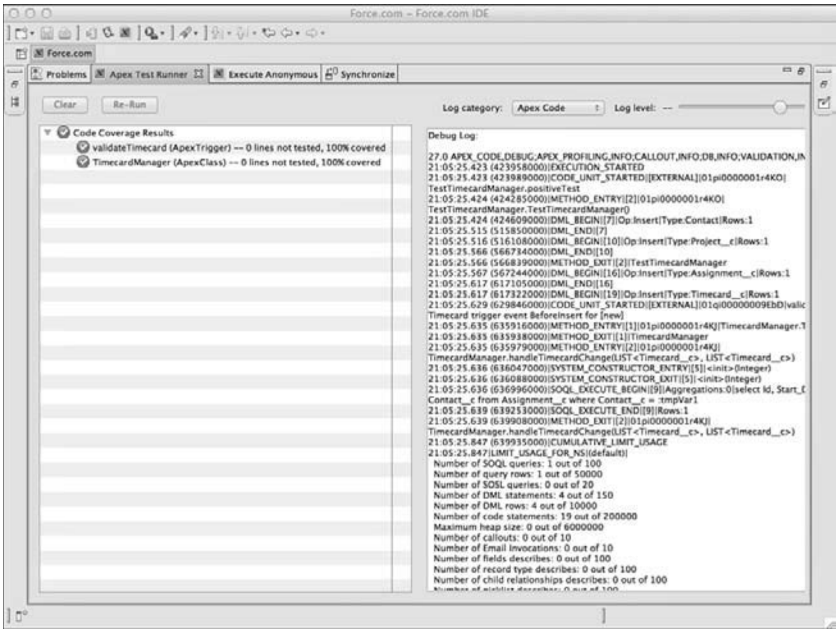
To create unit tests for the timecard validation code using the Force.com IDE, follow the same procedure as that for creating an ordinary Apex class. An optional variation on this process is to select the Test Class template from the Create New Apex Class Wizard. This generates skeleton code for a class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



```
Code Coverage Results
    validateTimecard (ApexTrigger) -- 0 lines not tested, 100% covered
    TimecardManager (ApexClass) -- 0 lines not tested, 100% covered
```

```
Debug Log:

27.0 APEX_CODE,DEBUG;APEX_PROFILING,INFO;CALLOUT,INFO;DB,INFO;VALIDATION,IN
21:05:25.423 (423958000)|EXECUTION_STARTED
21:05:25.423 (423989000)|CODE_UNIT_STARTED|[EXTERNAL]|01pi0000001r4KO|
TestTimecardManager.positiveTest
21:05:25.424 (424285000)|METHOD_ENTRY|[2]|01pi0000001r4KO|
TestTimecardManager.TestTimecardManager()
21:05:25.424 (424609000)|DML_BEGIN|[7]|Op:Insert|Type:Contact|Rows:1
21:05:25.515 (515850000)|DML_END|[7]
21:05:25.516 (516108000)|DML_BEGIN|[10]|Op:Insert|Type:Project__c|Rows:1
21:05:25.566 (566734000)|DML_END|[10]
21:05:25.566 (566839000)|METHOD_EXIT|[2]|TestTimecardManager
21:05:25.567 (567244000)|DML_BEGIN|[16]|Op:Insert|Type:Assignment__c|Rows:1
21:05:25.617 (617105000)|DML_END|[16]
21:05:25.617 (617322000)|DML_BEGIN|[19]|Op:Insert|Type:Timecard__c|Rows:1
21:05:25.629 (629846000)|CODE_UNIT_STARTED|[EXTERNAL]|01qi00000009EbD|valic
Timecard trigger event BeforeInsert for [new]
21:05:25.635 (635916000)|METHOD_ENTRY|[1]|01pi0000001r4KJ|TimecardManager.T
21:05:25.635 (635938000)|METHOD_EXIT|[1]|TimecardManager
21:05:25.635 (635979000)|METHOD_ENTRY|[2]|01pi0000001r4KJ|
TimecardManager.handleTimecardChange(LIST<Timecard__c>, LIST<Timecard__c>)
21:05:25.636 (636047000)|SYSTEM_CONSTRUCTOR_ENTRY|[5]|<init>(Integer)
21:05:25.636 (636088000)|SYSTEM_CONSTRUCTOR_EXIT|[5]|<init>(Integer)
21:05:25.636 (636996000)|SOQL_EXECUTE_BEGIN|[9]|Aggregations:0|select Id, Start_[
Contact__c from Assignment__c where Contact__c = :tmpVar1
21:05:25.639 (639253000)|SOQL_EXECUTE_END|[9]|Rows:1
21:05:25.639 (639908000)|METHOD_EXIT|[2]|01pi0000001r4KJ|
TimecardManager.handleTimecardChange(LIST<Timecard__c>, LIST<Timecard__c>)
21:05:25.847 (639935000)|CUMULATIVE_LIMIT_USAGE
21:05:25.847|LIMIT_USAGE_FOR_NS|(default)|
  Number of SOQL queries: 1 out of 100
  Number of query rows: 1 out of 50000
  Number of SOSL queries: 0 out of 20
  Number of DML statements: 4 out of 150
  Number of DML rows: 4 out of 10000
  Number of code statements: 19 out of 200000
  Maximum heap size: 0 out of 6000000
  Number of callouts: 0 out of 10
  Number of Email Invocations: 0 out of 10
  Number of fields describes: 0 out of 100
  Number of record type describes: 0 out of 100
  Number of child relationships describes: 0 out of 100
  Number of picklist describes: 0 out of 100
```

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