This page intentionally left blank
Oracle PL/SQL by Example

Sixth Edition

Copyright © 2023 by Pearson Education, Inc.

Trademarks
All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Pearson cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Warning and Disclaimer
Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an “as is” basis. The authors and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book.

Special Sales
For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382-3419.

For government sales inquiries, please contact governmentsales@pearsoned.com.

For questions about sales outside the U.S., please contact intlcs@pearson.com.

All rights reserved. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, request forms, and the appropriate contacts within the Pearson Education Global Rights & Permissions Department, please visit www.pearson.com/permissions. No patent liability is assumed with respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions. Nor is any liability assumed for damages resulting from the use of the information contained herein.

ISBN-10: 0-13-806283-8

Library of Congress Control Number: 2023933441

ScoutAutomatedPrintCode
Pearson’s Commitment to Diversity, Equity, and Inclusion

Pearson is dedicated to creating bias-free content that reflects the diversity of all learners. We embrace the many dimensions of diversity, including but not limited to race, ethnicity, gender, socioeconomic status, ability, age, sexual orientation, and religious or political beliefs.

Education is a powerful force for equity and change in our world. It has the potential to deliver opportunities that improve lives and enable economic mobility. As we work with authors to create content for every product and service, we acknowledge our responsibility to demonstrate inclusivity and incorporate diverse scholarship so that everyone can achieve their potential through learning. As the world’s leading learning company, we have a duty to help drive change and live up to our purpose to help more people create a better life for themselves and to create a better world.

Our ambition is to purposefully contribute to a world where

- Everyone has an equitable and lifelong opportunity to succeed through learning
- Our educational products and services are inclusive and represent the rich diversity of learners
- Our educational content accurately reflects the histories and experiences of the learners we serve
- Our educational content prompts deeper discussions with learners and motivates them to expand their own learning (and worldview)

While we work hard to present unbiased content, we want to hear from you about any concerns or needs with this Pearson product so that we can investigate and address them.

Please contact us with concerns about any potential bias at https://www.pearson.com/report-bias.html.
Figure Credits

Cover: wanpatsorn/Shutterstock
Figure 1-4 through Figure 1-17, Figure 13-1 through Figure 13-5: Oracle, Inc.
## Contents at a Glance

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preface</td>
<td></td>
<td>xvii</td>
</tr>
<tr>
<td>Chapter 1</td>
<td>PL/SQL Concepts</td>
<td>1</td>
</tr>
<tr>
<td>Chapter 2</td>
<td>PL/SQL Language Fundamentals</td>
<td>27</td>
</tr>
<tr>
<td>Chapter 3</td>
<td>SQL in PL/SQL</td>
<td>39</td>
</tr>
<tr>
<td>Chapter 4</td>
<td>Conditional Control: IF Statements</td>
<td>49</td>
</tr>
<tr>
<td>Chapter 5</td>
<td>Conditional Control: CASE Statements</td>
<td>65</td>
</tr>
<tr>
<td>Chapter 6</td>
<td>Iterative Control: Part I</td>
<td>85</td>
</tr>
<tr>
<td>Chapter 7</td>
<td>Iterative Control: Part II</td>
<td>111</td>
</tr>
<tr>
<td>Chapter 8</td>
<td>Error Handling and Built-in Exceptions</td>
<td>125</td>
</tr>
<tr>
<td>Chapter 9</td>
<td>Exceptions</td>
<td>135</td>
</tr>
<tr>
<td>Chapter 10</td>
<td>Exceptions: Advanced Concepts</td>
<td>151</td>
</tr>
<tr>
<td>Chapter 11</td>
<td>Introduction to Cursors</td>
<td>161</td>
</tr>
<tr>
<td>Chapter 12</td>
<td>Advanced Cursors</td>
<td>181</td>
</tr>
<tr>
<td>Chapter 13</td>
<td>Triggers</td>
<td>201</td>
</tr>
<tr>
<td>Chapter 14</td>
<td>Mutating Tables and Compound Triggers</td>
<td>221</td>
</tr>
<tr>
<td>Chapter 15</td>
<td>Collections</td>
<td>229</td>
</tr>
<tr>
<td>Chapter 16</td>
<td>Records</td>
<td>259</td>
</tr>
<tr>
<td>Chapter 17</td>
<td>Native Dynamic SQL</td>
<td>273</td>
</tr>
<tr>
<td>Chapter 18</td>
<td>Bulk SQL</td>
<td>289</td>
</tr>
<tr>
<td>Chapter 19</td>
<td>Procedures</td>
<td>319</td>
</tr>
<tr>
<td>Chapter 20</td>
<td>Functions</td>
<td>331</td>
</tr>
<tr>
<td>Chapter 21</td>
<td>Packages</td>
<td>341</td>
</tr>
<tr>
<td>Chapter 22</td>
<td>Stored Code Advanced Concepts</td>
<td>357</td>
</tr>
<tr>
<td>Chapter 23</td>
<td>Object Types in Oracle</td>
<td>379</td>
</tr>
<tr>
<td>Chapter 24</td>
<td>Storing Object Types in Tables</td>
<td>399</td>
</tr>
<tr>
<td>Chapter 25</td>
<td>Dynamic SQL with the <code>DBMS_SQL</code> Package</td>
<td>411</td>
</tr>
<tr>
<td>Appendix A</td>
<td>PL/SQL Formatting Guide</td>
<td>421</td>
</tr>
<tr>
<td>Appendix B</td>
<td>Student Database Schema</td>
<td>425</td>
</tr>
<tr>
<td>Index</td>
<td></td>
<td>433</td>
</tr>
</tbody>
</table>
## Contents

Preface .......................... xvii
Introduction to PL/SQL New Features in Oracle 21c .................. xxiii
  PL/SQL Extended Iterators ................. xxiv
  PL/SQL Qualified Expressions Enhancements .......... xxiv
  SQL Macros ........................ xxiv
  New JSON Data Type .................. xxiv
  New Pragma SUPPRESSES_WARNING_6009 ........... xxv
  PL/SQL Type Attributes in Non-Persistable User-Defined Types .... xxvi
  PL/SQL Function Enhanced Result Cache ........ xxvii

### Chapter 1
PL/SQL Concepts .................... 1
  Lab 1.1: PL/SQL Architecture ........ 2
    PL/SQL Architecture .............. 2
    PL/SQL Block Structure .......... 5
    How PL/SQL Gets Executed ........ 9
<table>
<thead>
<tr>
<th>Chapter 8</th>
<th>Error Handling and Built-in Exceptions</th>
<th>125</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lab 8.1</td>
<td>Handling Errors</td>
<td>125</td>
</tr>
<tr>
<td>Lab 8.2</td>
<td>Built-in Exceptions</td>
<td>128</td>
</tr>
<tr>
<td>Summary</td>
<td></td>
<td>133</td>
</tr>
<tr>
<td>Chapter 9</td>
<td>Exceptions</td>
<td>135</td>
</tr>
<tr>
<td>Lab 9.1</td>
<td>Exception Scope</td>
<td>135</td>
</tr>
<tr>
<td>Lab 9.2</td>
<td>User-Defined Exceptions</td>
<td>139</td>
</tr>
<tr>
<td>Lab 9.3</td>
<td>Exception Propagation</td>
<td>143</td>
</tr>
<tr>
<td>Re-raising</td>
<td>Exception</td>
<td>148</td>
</tr>
<tr>
<td>Summary</td>
<td></td>
<td>149</td>
</tr>
<tr>
<td>Chapter 10</td>
<td>Exceptions: Advanced Concepts</td>
<td>151</td>
</tr>
<tr>
<td>Lab 10.1</td>
<td>RAISE_APPLICATION_ERROR</td>
<td>151</td>
</tr>
<tr>
<td>Lab 10.2</td>
<td>EXCEPTION_INIT Pragma</td>
<td>155</td>
</tr>
<tr>
<td>Lab 10.3</td>
<td>SQLCODE and SQLERRM</td>
<td>157</td>
</tr>
<tr>
<td>Summary</td>
<td></td>
<td>160</td>
</tr>
<tr>
<td>Chapter 11</td>
<td>Introduction to Cursors</td>
<td>161</td>
</tr>
<tr>
<td>Lab 11.1</td>
<td>Types of Cursors</td>
<td>162</td>
</tr>
<tr>
<td>Implicit Cursor</td>
<td></td>
<td>162</td>
</tr>
<tr>
<td>Explicit Cursor</td>
<td></td>
<td>164</td>
</tr>
<tr>
<td>Lab 11.2</td>
<td>Table-Based and Cursor-Based Records</td>
<td>171</td>
</tr>
<tr>
<td>Table-Based</td>
<td>Records</td>
<td>172</td>
</tr>
<tr>
<td>Cursor-Based</td>
<td>Records</td>
<td>174</td>
</tr>
<tr>
<td>Lab 11.3</td>
<td>Cursor FOR Loops</td>
<td>175</td>
</tr>
<tr>
<td>Lab 11.4</td>
<td>Nested Cursors</td>
<td>177</td>
</tr>
<tr>
<td>Summary</td>
<td></td>
<td>179</td>
</tr>
<tr>
<td>Chapter 12</td>
<td>Advanced Cursors</td>
<td>181</td>
</tr>
<tr>
<td>Lab 12.1</td>
<td>Parameterized Cursors</td>
<td>181</td>
</tr>
<tr>
<td>Lab 12.2</td>
<td>Cursor Variables and Expressions</td>
<td>186</td>
</tr>
<tr>
<td>Cursor Variables</td>
<td></td>
<td>187</td>
</tr>
<tr>
<td>Cursor Expressions</td>
<td></td>
<td>193</td>
</tr>
<tr>
<td>Lab 12.3</td>
<td>FOR UPDATE Cursors</td>
<td>196</td>
</tr>
<tr>
<td>Summary</td>
<td></td>
<td>199</td>
</tr>
</tbody>
</table>
# Chapter 13  Triggers

Lab 13.1: What Triggers Are  
Database Trigger  
**BEFORE** Triggers  
**AFTER** Triggers  
Autonomous Transaction  
Lab 13.2: Types of Triggers  
Row and Statement Triggers  
**INSTEAD OF** Triggers  
Summary

# Chapter 14  Mutating Tables and Compound Triggers

Lab 14.1: Mutating Tables  
Lab 14.2: Compound Triggers  
Summary

# Chapter 15  Collections

Lab 15.1: PL/SQL Tables  
Associative Arrays  
Nested Tables  
Collection Methods  
Lab 15.2: Varrays  
Lab 15.3: Multidimensional Collections  
Lab 15.4: Collection Iteration Controls and Qualified Expressions  
Collection Iteration Controls  
Qualified Expressions  
Summary

# Chapter 16  Records

Lab 16.1: User-Defined Records  
User-Defined Records  
Qualified Expressions with Records  
Record Compatibility  
Lab 16.2: Nested Records  
Lab 16.3: Collections of Records  
Summary
### Chapter 17  Native Dynamic SQL

Lab 17.1: EXECUTE IMMEDIATE Statements  
EXECUTE IMMEDIATE Statement  
Lab 17.2: OPEN FOR, FETCH, and CLOSE Statements  
Summary  

### Chapter 18  Bulk SQL

Lab 18.1: FORALL Statements  
FORALL Statements  
SAVE EXCEPTIONS Option  
INDICES OF Option  
VALUES OF Option  
Lab 18.2: The BULK COLLECT Clause  
Lab 18.3: Binding Collections in SQL Statements  
Binding Collections with EXECUTE IMMEDIATE Statements  
Binding Collections with OPEN FOR, FETCH, and CLOSE Statements  
Summary  

### Chapter 19  Procedures

Lab 19.1: Creating Nested Procedures  
Nested Procedures  
Parameter Modes  
Forward Declaration  
Lab 19.2: Creating Stand-Alone Procedures  
Summary  

### Chapter 20  Functions

Lab 20.1: Creating Nested Functions  
Lab 20.2: Creating Stand-Alone Functions  
Summary  

### Chapter 21  Packages

Lab 21.1: Creating Packages  
Creating a Package Specification  
Creating a Package Body
<table>
<thead>
<tr>
<th>Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lab 21.2: Package Instantiation and Initialization</td>
<td>348</td>
</tr>
<tr>
<td>Package Instantiation and Initialization</td>
<td>349</td>
</tr>
<tr>
<td>Package State</td>
<td>351</td>
</tr>
<tr>
<td>Lab 21.3: SERIALLY_REUSABLE Packages</td>
<td>351</td>
</tr>
<tr>
<td>Summary</td>
<td>356</td>
</tr>
<tr>
<td><strong>Chapter 22</strong> Stored Code Advanced Concepts</td>
<td>357</td>
</tr>
<tr>
<td>Lab 22.1: Subprogram Overloading</td>
<td>357</td>
</tr>
<tr>
<td>Lab 22.2: Result-Cached Functions</td>
<td>363</td>
</tr>
<tr>
<td>Lab 22.3: Invoking PL/SQL Functions from SQL Statements</td>
<td>366</td>
</tr>
<tr>
<td>Invoking Functions in SQL Statements</td>
<td>367</td>
</tr>
<tr>
<td>Using Pipelined Table Functions</td>
<td>368</td>
</tr>
<tr>
<td>Using SQL Macros</td>
<td>370</td>
</tr>
<tr>
<td>Summary</td>
<td>375</td>
</tr>
<tr>
<td><strong>Chapter 23</strong> Object Types in Oracle</td>
<td>379</td>
</tr>
<tr>
<td>Lab 23.1: Object Types</td>
<td>380</td>
</tr>
<tr>
<td>Creating Object Types</td>
<td>381</td>
</tr>
<tr>
<td>Using Object Types with Collections</td>
<td>385</td>
</tr>
<tr>
<td>Lab 23.2: Object Type Methods</td>
<td>388</td>
</tr>
<tr>
<td>Using Constructor Methods</td>
<td>389</td>
</tr>
<tr>
<td>Using Member Methods</td>
<td>392</td>
</tr>
<tr>
<td>Using Static Methods</td>
<td>393</td>
</tr>
<tr>
<td>Comparing Objects</td>
<td>393</td>
</tr>
<tr>
<td>Summary</td>
<td>398</td>
</tr>
<tr>
<td><strong>Chapter 24</strong> Storing Object Types in Tables</td>
<td>399</td>
</tr>
<tr>
<td>Lab 24.1: Storing Object Types in Relational Tables</td>
<td>400</td>
</tr>
<tr>
<td>Lab 24.2: Storing Object Types in Object Tables</td>
<td>403</td>
</tr>
<tr>
<td>Lab 24.3: Type Evolution</td>
<td>405</td>
</tr>
<tr>
<td>Summary</td>
<td>410</td>
</tr>
<tr>
<td><strong>Chapter 25</strong> Dynamic SQL with the DBMS_SQL Package</td>
<td>411</td>
</tr>
<tr>
<td>Lab 25.1: Generating Dynamic SQL with the DBMS_SQL Package</td>
<td>412</td>
</tr>
<tr>
<td>Summary</td>
<td>420</td>
</tr>
<tr>
<td>Contents</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td><strong>Appendix A</strong>  PL/SQL Formatting Guide</td>
<td>421</td>
</tr>
<tr>
<td>Case</td>
<td>421</td>
</tr>
<tr>
<td>Whitespace</td>
<td>421</td>
</tr>
<tr>
<td>Naming Conventions</td>
<td>422</td>
</tr>
<tr>
<td>Comments</td>
<td>423</td>
</tr>
<tr>
<td><strong>Appendix B</strong>  Student Database Schema</td>
<td>425</td>
</tr>
<tr>
<td>Table and Column Descriptions</td>
<td>425</td>
</tr>
<tr>
<td><strong>Index</strong></td>
<td>433</td>
</tr>
</tbody>
</table>
Preface

Oracle® PL/SQL by Example, Sixth Edition, presents the Oracle PL/SQL programming language in a unique and highly effective format. It challenges you to learn Oracle PL/SQL by using it rather than by simply reading about it.

Just as a grammar workbook would teach you about nouns and verbs by first showing you examples and then asking you to write sentences, Oracle® PL/SQL by Example teaches you about loops, cursors, procedures, triggers, and so on by first showing you examples and then asking you to create these objects yourself.

Who This Book Is For

This book is intended for anyone who needs a quick but detailed introduction to programming with Oracle's PL/SQL language. The ideal readers are those with some relational database experience, with some Oracle experience, specifically with SQL, SQL*Plus, and SQL Developer, but with little or no experience with PL/SQL or with most other programming languages.

The content of this book is based primarily on the material that was taught in an Introduction to PL/SQL class at Columbia University's Computer Technology and Applications (CTA) program in New York City. The student body was rather diverse, in that there were some students who had years of experience with information technology (IT) and programming, but no experience with Oracle PL/SQL, and then there were those with absolutely no experience in IT or programming. The content of the book, like the class, is balanced to meet the needs of both extremes.
How This Book Is Organized

The intent of this workbook is to teach you about Oracle PL/SQL by explaining a programming concept or a particular PL/SQL feature and then illustrate it further by means of examples. Oftentimes, as the topic is discussed more in depth, these examples would be changed to illustrate newly covered material. In addition, most of the chapters of this book have “Additional Exercises” sections available through the companion website. These exercises allow you to test the depth of your understanding of the new material.

The basic structure of each chapter is as follows:

- Objectives
- Introduction
- Lab
- Lab...
- Summary

The “Objectives” section lists topics covered in the chapter. Basically, a single objective corresponds to a single lab.

The “Introduction” offers a short overview of the concepts and features covered in the chapter.

Each lab covers a single objective listed in the Objectives section of the chapter. In some instances, the objective is divided even further into the smaller individual topics in the lab. Then each topic is explained and illustrated with the help of examples and corresponding outputs. Note that as much as possible, each example is provided in its entirety so that a complete code sample is readily available. These examples are also available through the companion website.

At the end of each chapter, you will find a “Summary” section, which provides a brief conclusion of the material discussed in the chapter.

About the Companion Website

The companion website is located at www.informit.com/title/9780138062835. Here you will find these very important things:

- Files required to create and install the STUDENT schema.
- Files that contain example scripts used in the book chapters.
- “Additional Exercises” chapters where you are asked to create scripts based on the requirement provided. These exercises are meant to help you test the depth of your understanding.
By the Way
You need to visit the companion website, download the STUDENT schema, and install it in your database prior to using this book if you would like the ability to execute the scripts provided in the chapters and on the site.

What You Will Need

There are software programs as well as knowledge requirements necessary to complete the labs in this book. Note that some features covered throughout the book are applicable to Oracle 21c only. However, you will be able to run a great majority of the examples by using the following products:

- Oracle 18c or higher
- SQL Developer or SQL*Plus 18c or higher
- Access to the Internet

You can use either Oracle Personal Edition or Oracle Enterprise Edition to execute the examples in this book. If you use Oracle Enterprise Edition, it can be running on a remote server or locally on your own machine. It is recommended that you use Oracle 21c or Oracle 18c to perform all or most of the examples in this book. When a feature will work in the latest version of Oracle database only, the book will state so explicitly. Additionally, you should have access to and be familiar with SQL Developer or SQL*Plus.

You have several options for how to edit and run scripts in SQL Developer or SQL*Plus. There are also many third-party programs to edit and debug PL/SQL code. Both SQL Developer and SQL*Plus are used throughout this book because they are two Oracle-provided tools and come as part of the Oracle installation.

By the Way
Chapter 1 has a lab titled “PL/SQL Development Environment” that describes how to get started with SQL Developer and SQL*Plus. However, most of the examples used in the book were executed in SQL Developer.

About the Sample Schema

The STUDENT schema contains tables and other objects meant to keep information about a registration and enrollment system for a fictitious university. Ten tables in the system store data about students, courses, instructors, and so on. In addition to storing contact information (addresses and telephone numbers) for students and instructors, and descriptive information about courses (costs and prerequisites), the schema also keeps track of the sections for courses and the sections in which students are enrolled.
The SECTION table is one of the most important tables in the schema because it stores data about the individual sections that have been created for each course. Each section record also stores information about where and when the section will meet and which instructor will teach the section. The SECTION table is related to the COURSE and INSTRUCTOR tables.

The ENROLLMENT table is just as important because it keeps track of students who are enrolled in sections. Each enrollment record also stores information about the student’s grade and enrollment date. The ENROLLMENT table is related to the STUDENT and SECTION tables.

The STUDENT schema also has several other tables that manage grading for each student in each section.

The detailed structure of the STUDENT schema is described in Appendix B, “Student Database Schema.”
Elena Rakhimov: My contribution to this book reflects the help and advice of many people. I am especially indebted to Tonya Simpson and Chris Zahn for their meticulous editing skills and to Michael Rinomhota and Dan Hotka for their invaluable technical expertise. Many thanks to Malobika Chakraborty, and many others at Pearson who diligently worked to bring this book to market. Most importantly, to my family, whose excitement, enthusiasm, inspiration, and support encouraged me to work hard to the very end and were exceeded only by their love.
About the Author

Elena Rakhimov has more than 20 years of experience in software architecture and development in a wide spectrum of enterprise and business environments ranging from nonprofit organizations to Wall Street to her current position with a prominent consulting company. Her determination to stay “hands-on” notwithstanding, Elena managed to excel in the academic arena, having taught relational database programming at Columbia University’s highly esteemed Computer Technology and Applications program. She was educated in database analysis and design at Columbia University and in applied mathematics at Baku State University in Azerbaijan.
Introduction to PL/SQL
New Features in Oracle 21c

Oracle 21c has introduced several new features and improvements for PL/SQL. This introduction briefly describes features not covered in this book and points you to specific chapters for features that are within the scope of this book. The list of features described here is also available in the “Changes in This Release for Oracle Database PL/SQL Language Reference” section of the PL/SQL Language Reference manual offered as part of Oracle help available online.

The new PL/SQL features and enhancements are as follows:

- PL/SQL Extended Iterators
- PL/SQL Qualified Expressions Enhancements
- SQL Macros
- New JSON Data Type
- New Pragma SUPRESSES_WARNING_6009
- PL/SQL Type Attributes in Non-Persistable User-Defined Types
- PL/SQL Function Enhanced Result Cache
PL/SQL Extended Iterators

In this release, Oracle has extended functionality of the numeric FOR loop. For example, you can combine multiple iteration boundaries in the comma-delimited list in a single loop. Prior to Oracle 21c, you would need to specify a distinct FOR loop for a specific iteration boundary. This functionality is covered in greater detail in Lab 6.3, “Numeric FOR Loops,” and in Lab 15.4, “Collection Iteration Controls and Qualified Expressions.”

PL/SQL Qualified Expressions Enhancements

Qualified expressions were introduced in Oracle 18c and further improved in Oracle 21c. Essentially, starting with Oracle 18c, you can populate a record or a collection data type with values provided by an expression constructor. Qualified expressions are covered in Lab 15.4, “Collection Iteration Controls and Qualified Expressions,” and in Chapter 16, “Records.”

SQL Macros

Starting with Oracle 21c, you can create a PL/SQL function and mark it as a SQL macro. This capability is especially useful when a PL/SQL function is used in a SQL statement. Every time a PL/SQL function is called from a SQL statement, there is a context switch between SQL and PL/SQL engines. This context switch adds a certain processing overhead. However, after a function is flagged as a SQL macro, the context switch is eliminated. SQL macros are discussed in Lab 22.3, “Invoking PL/SQL Functions from SQL Statements.”

New JSON Data Type

JSON (JavaScript Object Notation) is a new data type available in SQL and PL/SQL. Prior to Oracle 21c, JSON data could be stored as VARCHAR2 or CLOB data types. With Oracle21c, a JSON data type may be used when creating a column in a table, in SQL queries, and in PL/SQL programs. Consider the following example with a table that contains JSON data type column.

For Example  Table with JSON Column

```sql
CREATE TABLE json_test
(id   NUMBER,
 ,json_doc JSON);

-- Insert sample data
INSERT INTO json_test
```
VALUES (1,
    | "Doc"   : 1,
    | "DocName": "Sample JSON Doc 1",
    | "DocAuthor": "John Smith"
  );

INSERT INTO json_test
VALUES (2,
    | "Doc"  : 2,
    | "DocName": "Sample JSON Doc 2",
    | "DocAuthor": "Mary Brown"
  );

Note that the string '{...}' in the INSERT statement is converted to a JSON data type. JSON data may be queried as illustrated by the next example.

**For Example  Querying JSON Data**

```sql
SELECT json_doc
FROM json_test;
```

```
+-------------------+---------------+
| JSON_DOC           |
+-------------------+---------------+
| {"Doc":1,"DocName":"Sample JSON Doc 1","DocAuthor":"John Smith"} |
| {"Doc":2,"DocName":"Sample JSON Doc 2","DocAuthor":"Mary Brown"} |
+-------------------+---------------+
```

---

```
-- Select Doc Name and Doc Author from json_doc
SELECT j.json_doc.DocName, j.json_doc.DocAuthor
FROM json_test j;
```

```
+---------------+---------------+
| DOCNAME        | DOCAUTHOR     |
|---------------+---------------|
| "Sample JSON Doc 1" | "John Smith" |
| "Sample JSON Doc 2" | "Mary Brown"  |
+---------------+---------------+
```

Take a closer look at the second SELECT statement. When you are referencing individual elements of JSON data, a table alias is required. Without a table alias, the second SELECT statement would cause the following error:

```
ERROR at line 1:
ORA-00904: "JSON_DOC"."DOCAUTHOR": invalid identifier
```

As mentioned earlier, JSON data may be used in PL/SQL. There are various built-in functions such as JSON_EXISTS and JSON_EQUAL, and JSON object types such as JSON_OBJECT_T.

Note that the JSON data type is outside the scope of this book, and detailed information on it may be found in the Oracle's JSON Developer's Guide available online.

**New Pragma SUPPRESSES_WARNING_6009**

New pragma SUPPRESSES_WARNING_6009 suppresses PL/SQL warning PLW-06009. This warning occurs when an exception handler does not utilize RAISE or RAISE_APPLICATION_ERROR statements. This warning applies to
stand-alone and package procedures and functions as well as methods in type definitions. The \texttt{SUPPRESSES\_WARNING\_6009} pragma is not covered in this book, and additional information on it may be found in the \textit{PL/SQL Language Reference} manual offered as part of Oracle help available online.

\section*{PL/SQL Type Attributes in Non-Persistable User-Defined Types}

Starting with Oracle 18c, you can define a user-defined type as persistable or not persistable. Persistable is a default option, and after such an object type is created, it may be referenced in PL/SQL programs, SQL statements, and in the DDL statements.

When a user-defined type is defined as non-persistable, it may be referenced in the PL/SQL code and SQL statements only. Referencing it in a DDL statement such as \texttt{CREATE TABLE} causes an error.

\textbf{For Example} \hspace{1em} \textit{Creating a Non-Persistable Object}

\begin{verbatim}
CREATE TYPE non_persist_type_obj AS OBJECT
  (city  VARCHAR2(30)
   ,state VARCHAR2(2)
   ,zip   VARCHAR2(5))
NOT PERSISTABLE;
/
Type NON_PERSIST_TYPE_OBJ compiled
CREATE TABLE test_obj
  (id      NUMBER
   ,zip_obj non_persist_type_obj);
ORA-22384: cannot create a column or table of a non-persistable type

However, this non-persistable object type may be used in the PL/SQL code as illustrated in this example.

\textbf{For Example} \hspace{1em} \textit{Using a Non-Persistable Object Type in PL/SQL Code}

\begin{verbatim}
DECLARE
  v_zip_obj non_persist_type_obj :=
    non_persist_type_obj('New York', 'NY', null);
BEGIN
  DBMS_OUTPUT.PUT_LINE ('City: '||v_zip_obj.city);
  DBMS_OUTPUT.PUT_LINE ('State: '|| v_zip_obj.state);
END;
/

City: New York
State: NY
\end{verbatim}
Starting with Oracle 21c, non-persistable user-defined types have been enhanced to handle attributes of PL/SQL data types such as BOOLEAN or PLS_INTEGER.

For Example  Creating a Non-Persistable Object

```sql
DROP TYPE non_persist_type_obj;
/
Type NON_PERSIST_TYPE_OBJ dropped.
CREATE TYPE non_persist_type_obj AS OBJECT
  (city     VARCHAR2(30)
  ,state    VARCHAR2(2)
  ,zip      VARCHAR2(5)
  ,is_valid BOOLEAN)
NOT PERSISTABLE;
/
Type NON_PERSIST_TYPE_OBJ compiled
DECLARE
  v_zip_obj non_persist_type_obj :=
    non_persist_type_obj('New York', 'NY', null, true);
BEGIN
  DBMS_OUTPUT.PUT_LINE ('City: '||v_zip_obj.city);
  DBMS_OUTPUT.PUT_LINE ('State: '|| v_zip_obj.state);
  IF v_zip_obj.is_valid
  THEN
    DBMS_OUTPUT.PUT_LINE ('Valid');
  ELSE
    DBMS_OUTPUT.PUT_LINE ('Not valid');
  END IF;
END;
/
City: New York
State: NY
Valid
```

Additional details on PL/SQL data types in non-persistable user-defined types may be found in the PL/SQL Language Reference manual offered as part of Oracle help available online.

PL/SQL Function Enhanced Result Cache

With Oracle 21c, result cache functionality has been expanded to provide better control of the cached result set, increase the number of use cases for such functions, and further improve database performance and reduce the overall workload. Result-cached function is covered in Lab 22.2, “Result-Cached Functions.”
PL/SQL stands for “Procedural Language Extension to SQL.” Because of its tight integration with SQL, PL/SQL supports the great majority of the SQL features, such as SQL data manipulation, data types, operators, functions, and transaction control statements. As an extension to SQL, PL/SQL combines SQL with programming structures and subroutines available in any high-level language.

PL/SQL is used for both server-side and client-side development. For example, database triggers (code that is attached to tables discussed in Chapter 13, “Triggers,” and Chapter 14, “Mutating Tables and Compound Triggers”) on the server side and the logic behind an Oracle form on the client side can be written using PL/SQL. In addition, PL/SQL can be used to develop web and mobile applications in both conventional and cloud environments when used in conjunction with a wide variety of Oracle development tools.
Lab 1.1: PL/SQL Architecture

After this lab, you will be able to

- Describe PL/SQL Architecture
- Discuss PL/SQL Block Structure
- Understand How PL/SQL Gets Executed

Many Oracle applications are built using multiple tiers, also known as \(\text{N}\)-tier architecture, where each tier represents a separate logical and physical layer. For example, a three-tier architecture would consist of three tiers: a data management tier, an application processing tier, and a presentation tier. In this architecture, the Oracle database resides in the data management tier, and the programs that make requests against this database reside in either the presentation tier or the application processing tier. Such programs can be written in many programming languages, including PL/SQL. An example of a simplified three-tier architecture is shown in Figure 1.1.

PL/SQL Architecture

Although PL/SQL is just like any other programming language, its main distinction is that it is not a stand-alone programming language. Rather, PL/SQL is a part of the Oracle RDBMS as well as various Oracle development tools such as Oracle Application Express (APEX) and Oracle Forms and Reports, components of Oracle Fusion Middleware. As a result, PL/SQL may reside in any layer of the multitier architecture.

No matter which layer PL/SQL resides in, any PL/SQL block or subroutine is processed by the PL/SQL engine, which is a special component of various Oracle products. As a result, it is easy to move PL/SQL modules between various tiers. The PL/SQL engine processes and executes any PL/SQL statements and sends any SQL statements to the SQL statement processor. The SQL statement processor is always located on the Oracle server. Figure 1.2 illustrates the PL/SQL engine residing on the Oracle server.
Lab 1.1: PL/SQL Architecture

Figure 1.1 Three-Tier Architecture

**Presentation Tier**
This tier enables the user to interact with an application. It collects and sends user requests for further processing to the application processing tier.

**Application Processing Tier (aka Middle Tier)**
This tier processes requests received from the user and services results of those requests back to the presentation tier. In addition, it evaluates business rules, validates user requests, and interacts with the data management tier.

**Data Management Tier**
This tier is essentially the database server. The data is stored and retrieved from the database server based on the requests received from the application processing tier.

*Purchase Oracle PL/SQL Book by Example:*
Add purchase item to the shopping cart
Proceed to checkout

*Validate user request:*
Verify customer information
Request customer data

*Query*  
*Query Results*  

Process query received from the application processing tier
When the PL/SQL engine is located on the server, the whole PL/SQL block is passed to the PL/SQL engine on the Oracle server. The PL/SQL engine processes the block according to the scheme depicted in Figure 1.2.

When the PL/SQL engine is located on the client, as it is in Oracle development tools, the PL/SQL processing is done on the client side. All SQL statements that are embedded within the PL/SQL block are sent to the Oracle server for further processing. When a PL/SQL block contains no SQL statements, the entire block is executed on the client side.

Using PL/SQL has several advantages. For example, when you issue a SELECT statement in SQL*Plus or SQL Developer against the STUDENT table, it retrieves a list of students. The SELECT statement you issued at the client computer is sent to the database server to be executed. The results of this execution are then returned to the client. In turn, rows are displayed on your client machine.

Now, assume that you need to issue multiple SELECT statements. Each SELECT statement is a request against the database and is sent to the Oracle server. The results of each SELECT statement are sent back to the client. Each time a SELECT statement is executed, network traffic is generated. Hence, multiple SELECT statements will result in multiple round-trip transmissions, adding significantly to the network traffic.

When these SELECT statements are combined into a PL/SQL program, they are sent to the server as a single unit. The SELECT statements in this PL/SQL program are executed at the server. The server sends the results of these SELECT statements back to the client, also as a single unit. Therefore, a PL/SQL program
encompassing multiple `SELECT` statements can be executed at the server and have all the results returned to the client in the same round trip. This process is obviously more efficient than having each `SELECT` statement execute independently. This model is illustrated in Figure 1.3.

Figure 1.3 compares two applications. The first application uses four independent SQL statements that generate eight trips on the network. The second application combines SQL statements into a single PL/SQL block, which is then sent to the PL/SQL engine. The engine sends SQL statements to the SQL statement processor and checks the syntax of the PL/SQL statements. As you can see, only two trips are generated on the network with the second application.

In addition, applications written in PL/SQL are portable. They can run in any environment that Oracle products can run in. Because PL/SQL does not change from one environment to the next, different tools can use PL/SQL programs.

**PL/SQL Block Structure**

A block is the most basic unit in PL/SQL. All PL/SQL programs are combined into blocks. These blocks can also be nested within one another. Usually, PL/SQL blocks combine statements that represent a single logical task. Therefore, different tasks within a single program can be separated into blocks. With this structure, it is easier to understand and maintain the logic of the program.
PL/SQL blocks can be divided into two groups: named and anonymous. Named PL/SQL blocks are used when creating subroutines. These subroutines, which include procedures, functions, and packages, can be stored in the database and referenced by their names later. In addition, subroutines such as procedures and functions can be defined within the anonymous PL/SQL block. These subroutines exist as long as the block is executing and cannot be referenced outside the block. In other words, subroutines defined in one PL/SQL block cannot be called by another PL/SQL block or referenced by their names later. Subroutines are discussed in Chapters 19 through 21. Anonymous PL/SQL blocks, as you have probably guessed, do not have names. As a result, they cannot be stored in the database or referenced later.

PL/SQL blocks contain three sections: a declaration section, an executable section, and an exception-handling section. The executable section is the only mandatory section of the block; both the declaration and exception-handling sections are optional. As a result, a PL/SQL block has the structure illustrated in Listing 1.1.

Listing 1.1  PL/SQL Block Structure

```plaintext
DECLARE
  Declaration statements
BEGIN
  Executable statements
EXCEPTION
  Exception-handling statements
END;
```

**Declaration Section**

The declaration section is the first section of the PL/SQL block. It contains definitions of PL/SQL identifiers such as variables, constants, cursors, and so on. PL/SQL identifiers are covered in detail throughout this book.

**For Example**

```plaintext
DECLARE
  v_first_name VARCHAR2(35);
  v_last_name  VARCHAR2(35);
```

This example shows the declaration section of an anonymous PL/SQL block. It begins with the keyword `DECLARE` and contains two variable declarations. The names of the variables, `v_first_name` and `v_last_name`, are followed by their data types and sizes. Notice that a semicolon terminates each declaration.
**Executable Section**

The executable section is the next section of the PL/SQL block. It contains executable statements that allow you to manipulate the variables that have been declared in the declaration section.

**For Example**

```plsql
BEGIN
    SELECT first_name, last_name
    INTO v_first_name, v_last_name
    FROM student
    WHERE student_id = 123;
    DBMS_OUTPUT.PUT_LINE ('Student name: '||v_first_name||' '||v_last_name);
END;
```

This example shows the executable section of the PL/SQL block. It begins with the keyword `BEGIN` and contains a `SELECT INTO` statement from the `STUDENT` table. The first and last names for student ID 123 are selected into two variables: `v_first_name` and `v_last_name`. Chapter 3, “SQL in PL/SQL,” contains a detailed explanation of the `SELECT INTO` statement. Next, the values of the variables, `v_first_name` and `v_last_name`, are displayed on the screen with the help of the `DBMS_OUTPUT.PUT_LINE` statement. This statement is covered later in this chapter in greater detail. The end of the executable section of this block is marked by the keyword `END`.

**By the Way**

The executable section of any PL/SQL block always begins with the keyword `BEGIN` and ends with the keyword `END`.

**Exception-Handling Section**

Two types of errors may occur when a PL/SQL block is executed: compilation or syntax errors and runtime errors. Compilation errors are detected by the PL/SQL compiler when there is a misspelled reserved word or a missing semicolon at the end of the statement.

**For Example**

```plsql
BEGIN
    DBMS_OUTPUT.PUT_LINE ('This is a test')
END;
```

This example contains a syntax error: the `DBMS_OUTPUT.PUT_LINE` statement is not terminated by a semicolon.
Runtime errors occur while the program is running and cannot be detected by the PL/SQL compiler. These types of errors are detected or handled by the exception-handling section of the PL/SQL block. It contains a series of statements that are executed when a runtime error occurs within the block.

When a runtime error occurs, control is passed to the exception-handling section of the block. The error is then evaluated, and a specific exception is raised or executed. This is best illustrated by the following example. All changes are shown in bold.

**For Example**

```sql
BEGIN
    SELECT first_name, last_name
    INTO v_first_name, v_last_name
    FROM student
    WHERE student_id = 123;

    DBMS_OUTPUT.PUT_LINE ('Student name: '||v_first_name||' '||v_last_name);
EXCEPTION
    WHEN NO_DATA_FOUND
    THEN
        DBMS_OUTPUT.PUT_LINE ('There is no student with student id 123');
END;
```

This example shows the exception-handling section of the PL/SQL block. It begins with the keyword **EXCEPTION**. The **WHEN clause** evaluates which exception must be raised. In this example, there is only one exception, called **NO_DATA_FOUND**, and it is raised when the **SELECT** statement does not return any rows. If there is no record for student ID 123 in the **STUDENT** table, control will be passed to the exception-handling section and the **DBMS_OUTPUT.PUT_LINE** statement will be executed. Chapter 8, “Error Handling and Built-In Exceptions,” Chapter 9, “Exceptions,” and Chapter 10, “Exceptions: Advanced Concepts,” contain detailed explanations of the exception-handling section.

You have seen examples of the declaration section, executable section, and exception-handling section. These examples may be combined into a single PL/SQL block.

**For Example  ch01_1a.sql**

```sql
DECLARE
    v_first_name VARCHAR2(35);
    v_last_name  VARCHAR2(35);
BEGIN
    SELECT first_name, last_name
    INTO v_first_name, v_last_name
    FROM student
    WHERE student_id = 123;

    DBMS_OUTPUT.PUT_LINE ('Student name: '||v_first_name||' '||v_last_name);
END;
```
EXCEPTION
WHEN NO_DATA_FOUND
THEN
  DBMS_OUTPUT.PUT_LINE ('There is no student with student id 123');
END;

How PL/SQL Gets Executed

Every time an anonymous PL/SQL block is executed, the code is sent to the PL/SQL engine, where it is compiled. A named PL/SQL block is compiled only at the time of its creation or if it has been changed. The compilation process includes syntax and semantic checking, as well as code generation.

Syntax checking involves checking PL/SQL code for syntax or compilation errors. As stated previously, a syntax error occurs when a statement does not exactly correspond to the syntax of the programming language. A misspelled keyword, a missing semicolon at the end of the statement, and an undeclared variable are all examples of syntax errors. After syntax errors are corrected, the compiler can generate a parse tree.

By the Way
A parse tree is a tree-like structure that represents the language rules of a computer language.

Semantic checking involves further processing on the parse tree. It determines whether database objects such as table names and column names referenced in the SELECT statements are valid and whether you have privileges to access them. At the same time, the compiler can assign a storage address to program variables that are used to hold data. This process, which is called binding, allows Oracle software to reference storage addresses when the program is run.

Code generation creates code for the PL/SQL block in interpreted or native mode. Code created in interpreted mode is called p-code. P-code is a list of instructions to the PL/SQL engine that are interpreted at runtime. Code created in a native mode is a processor-dependent system code that is called native code. Because native code does not need to be interpreted at runtime, it usually runs slightly faster.

The mode in which the PL/SQL engine generates code is determined by the PLSQL_CODE_TYPE database initialization parameter. By default, its value is set to INTERPRETED. This parameter is typically set by the database administrators.

For named blocks, both p-code and native code are stored in the database and are used the next time the program is executed. When the process of compilation has completed successfully, the status of a named PL/SQL block is set to VALID, and it is also stored in the database. If the compilation process was not successful, the status of the named PL/SQL block is set to INVALID.
Watch Out!
Successful compilation of the named PL/SQL block on one occasion does not guarantee successful execution of this block in the future. If, at the time of execution, any one of the stored objects referenced by the block is not present in the database or not accessible to the block, execution will fail. At such time, the status of the named PL/SQL block will be changed to INVALID.

Lab 1.2: PL/SQL Development Environment

After this lab, you will be able to
- Get Started with SQL Developer
- Get Started with SQL*Plus
- Execute PL/SQL Scripts

SQL Developer and SQL*Plus are two Oracle-provided tools that you can use to develop and run PL/SQL scripts. SQL*Plus is an old-style command-line utility tool that has been part of the Oracle platform since its infancy. It is included in the Oracle installation on every platform. SQL Developer is a free graphical tool used for database development and administration. It is available either as a part of the Oracle installation or via download from Oracle's website.

Due to its graphical interface, SQL Developer is a much easier environment to use than SQL*Plus. It allows you to browse database objects; run SQL statements; and create, debug, and run PL/SQL statements. In addition, it supports syntax highlighting and formatting templates that become very useful when you are developing and debugging complex PL/SQL modules.

Even though SQL*Plus and SQL Developer are two very different tools, their underlying functionality and their interactions with the database are very similar. At runtime, the SQL and PL/SQL statements are sent to the database. After they are processed, the results are sent back from the database and displayed on the screen.

The examples used in this chapter are executed in both tools to illustrate some of the interface differences when appropriate. Note that the primary focus of this book is learning PL/SQL; thus, these tools are covered only to the degree that is required to run PL/SQL examples provided by this book.

Getting Started with SQL Developer
Whether SQL Developer has been installed as part of the Oracle installation or as a separate module, you need to create at least one connection to the database server. You can accomplish this task by clicking the Plus icon located in the
upper-left corner of the Connections tab. Clicking this icon activates the New/Select Database Connection dialog box, as shown in Figure 1.4.

In Figure 1.4, you need to provide a connection name (ORCLPDB_STUDENT), username (student), and password (learn).

In the same dialog box, you need to provide database connection information such as the hostname (typically, the IP address of the machine or the machine name where the database server resides), the default port where that database listens for the connection requests (usually 1521), and the SID (system ID) or service name that identifies a particular database. Both the SID and service name would depend on the names you picked up for your installation of Oracle. The default SID for the pluggable database is usually set to ORCLPDB.

**Watch Out!**

If you have not created the **STUDENT** schema yet, you will not be able to create this connection successfully. To create the **STUDENT** schema, refer to the installation instructions provided on the companion website.

After the connection has been successfully created, you can connect to the database by double-clicking the ORCLPDB_STUDENT. By expanding the ORCLPDB_STUDENT (clicking the plus sign located to the left of it), you can browse various database objects available in the **STUDENT** schema. For example, Figure 1.5 shows list of tables available in the **STUDENT** schema. At this point you can start typing SQL or PL/SQL commands in the Worksheet window.
To disconnect from the STUDENT schema, you need to right-click the ORCLPDB_STUDENT and click the Disconnect option. This action is illustrated in Figure 1.6.
Getting Started with SQL*Plus

You can access SQL*Plus via the Programs menu or by typing `sqlplus` in the command prompt window. When you open SQL*Plus, you are prompted to enter the username and password (`student` or `student@orclpdb` and `learn`, respectively).

By the Way

In SQL*Plus, the password is not displayed on the screen, even as masked text.

Watch Out!

You need to have an entry for the pluggable database container (PDB) in your `TNSNAMES.ORA` file to be able to connect to the `STUDENT` schema via SQL*Plus. It should look similar to

```sql
ORCLPDB =
    (DESCRIPTION =
        (ADDRESS = (PROTOCOL = TCP)(HOST = localhost)(PORT = 1521))
        (CONNECT_DATA =
            (SERVER = DEDICATED)
            (SERVICE_NAME = orclpdb)
        )
    )
```

After successful login, you are able to enter your commands at the `SQL>` prompt. This prompt is illustrated in Figure 1.7.

![SQL*Plus](image)

**Figure 1.7** Connecting to the Database in SQL*Plus

To terminate your connection to the database, type either `EXIT` or `QUIT` command and press Enter.
Did You Know?
Terminating the database connection in either SQL Developer or SQL*Plus terminates only your own client connection. In a multiuser environment, there may be potentially hundreds of client connections to the database server at any time. As these connections terminate and new ones are initiated, the database server continues to run and send various query results back to its clients.

Executing PL/SQL Scripts
As mentioned earlier, at runtime SQL and PL/SQL statements are sent from the client machine to the database. After they are processed, the results are sent back from the database to the client and are displayed on the screen. However, there are some differences between entering SQL and PL/SQL statements.

Consider the following example of a SQL statement.

For Example

```sql
SELECT first_name, last_name
FROM student
WHERE student_id = 102;
```

If this statement is executed in SQL Developer, the semicolon is optional. To execute this statement, you need to click the triangle button in the ORCLPDB_STUDENT SQL Worksheet or press the F9 key on your keyboard. The results of this query are then displayed in the Query Result window, as shown in Figure 1.8. Note that the statement does not have a semicolon.

When the same `SELECT` statement is executed in SQL*Plus, the semicolon is required. It signals SQL*Plus that the statement is terminated. As soon as you press the Enter key, the query is sent to the database and the results are displayed on the screen, as shown in Figure 1.9.
Lab 1.2: PL/SQL Development Environment

Figure 1.8 Executing a Query in SQL Developer

Figure 1.9 Executing a Query in SQL*Plus
Now, consider the example of the PL/SQL block used in the previous lab.

For Example  
\textit{ch01\_1a.sql}

```
DECLARE
  v\_first\_name VARCHAR2(35);
  v\_last\_name  VARCHAR2(35);
BEGIN
  SELECT first\_name, last\_name
  INTO v\_first\_name, v\_last\_name
  FROM student
  WHERE student\_id = 123;

  DBMS\_OUTPUT\_PUT\_LINE ('Student name: ||v\_first\_name|| ' ||
  v\_last\_name);
EXCEPTION
  WHEN NO\_DATA\_FOUND
  THEN
    DBMS\_OUTPUT\_PUT\_LINE ('There is no student with student id 123');
END;
```

Note that each individual statement in this script is terminated by a semicolon. Each variable declaration, the SELECT INTO statement, both DBMS\_OUTPUT\_PUT\_LINE statements, and the END keyword are all terminated by the semicolon. This syntax is necessary because in PL/SQL the semicolon marks termination of an individual statement within a block. In other words, the semicolon is not a block terminator.

Because SQL Developer is a graphical tool, it does not require a special block terminator. The preceding example can be executed in SQL Developer by clicking the triangle button in the ORCLPDB\_STUDENT SQL Worksheet or pressing the F9 key on your keyboard, as shown in Figure 1.10.

The block terminator becomes necessary when the same example is executed in SQL*Plus. Because it is a command-line tool, SQL*Plus requires a textual way of knowing when the block has terminated and is ready for execution. The forward slash (/) is interpreted by SQL*Plus as a block terminator. After you press the Enter key, the PL/SQL block is sent to the database, and the results are displayed on the screen. This is shown in Figure 1.11 on the left.

If you omit /, SQL*Plus will not execute the PL/SQL script. Instead, it will simply add a blank line to the script when you press the Enter key. This is shown in Figure 1.11 on the right, where lines 16 through 20 are blank.
Lab 1.2: PL/SQL Development Environment

Triangle button to execute the script

Figure 1.10 Executing a PL/SQL Block in SQL Developer

Figure 1.11 Executing a PL/SQL Block in SQL*Plus with a Block Terminator and Without a Block Terminator
Lab 1.3: PL/SQL: The Basics

After this lab, you will be able to

- Use the DBMS_OUTPUT.PUT_LINE Statement
- Use the Substitution Variable Feature

We noted earlier that PL/SQL is not a stand-alone programming language; rather, it exists only as a tool within the Oracle environment. As a result, it does not really have any capabilities to accept input from a user. The lack of this ability is compensated with the special feature of the SQL Developer and SQL*Plus tools called a substitution variable.

Similarly, it is often helpful to provide the user with some pertinent information after the execution of a PL/SQL block, and this is accomplished with the help of the DBMS_OUTPUT.PUT_LINE statement. Note that unlike the substitution variable, this statement is part of the PL/SQL language.

DBMS_OUTPUT.PUT_LINE Statement

In the previous section of this chapter, you saw how the DBMS_OUTPUT.PUT_LINE statement may be used in a script to display information on the screen. The DBMS_OUTPUT.PUT_LINE is a call to the procedure PUT_LINE. This procedure is a part of the DBMS_OUTPUT package that is owned by the Oracle user SYS.

The DBMS_OUTPUT.PUT_LINE statement writes information to the buffer for storage. After a program has completed, the information from the buffer is displayed on the screen. The size of the buffer can be set between 2000 and 1 million bytes.

To see the results of the DBMS_OUTPUT.PUT_LINE statement on the screen, you need to enable it. In SQL Developer, you do so by selecting the View menu option and then choosing the Dbms Output option, as shown in Figure 1.12.

After the Dbms Output window appears in SQL Developer, you must click the plus button, as shown in Figure 1.13.
Lab 1.3: PL/SQL: The Basics

Figure 1.12 Enabling DBMS_OUTPUT in SQL Developer: Step 1

Figure 1.13 Enabling DBMS_OUTPUT in SQL Developer: Step 2
After you click the plus button, you will be prompted with the name of the connection for which you want to enable the statement. You need to select ORCLPDB_STUDENT and click OK. The result of this operation is shown in Figure 1.14.

To enable the DBMS_OUTPUT statement in SQL*Plus, you enter one of the following statements before the PL/SQL block:

```
SET SERVEROUTPUT ON;
```

or

```
SET SERVEROUTPUT ON SIZE 5000;
```

The first SET statement enables the DBMS_OUTPUT.PUT_LINE statement, with the default value for the buffer size being used. The second SET statement not only enables the DBMS_OUTPUT.PUT_LINE statement but also changes the buffer size from its default value to 5000 bytes.

*Figure 1.14 Enabling DBMS_OUTPUT in SQL Developer: Step 3*
Similarly, if you do not want information to be displayed on the screen by the DBMS_OUTPUT.PUT_LINE statement, you can issue the following SET command prior to the PL/SQL block:

```
SET SERVEROUTPUT OFF;
```

### Substitution Variable Feature

Substitution variables are a special type of variable that enables PL/SQL to accept input from a user at a runtime. These variables cannot be used to output values, however, because no memory is allocated for them. Substitution variables are replaced with the values provided by the user before the PL/SQL block is sent to the database. The variable names are usually prefixed by the ampersand (&) or double ampersand (&&) character.

Consider the following example.

**For Example**  
*ch01_1b.sql*

```sql
DECLARE
  v_student_id NUMBER := &sv_student_id;
  v_first_name VARCHAR2(35);
  v_last_name  VARCHAR2(35);
BEGIN
  SELECT first_name, last_name
  INTO v_first_name, v_last_name
  FROM student
  WHERE student_id = v_student_id;
  DBMS_OUTPUT.PUT_LINE ('Student name: '||v_first_name||' '||v_last_name);
EXCEPTION
  WHEN NO_DATA_FOUND
  THEN
    DBMS_OUTPUT.PUT_LINE ('There is no such student');
END;
```

When this example is executed, the user is asked to provide a value for the student ID. The student's name is then retrieved from the STUDENT table if there is a record with the given student ID. If there is no record with the given student ID, the message from the exception-handling section is displayed on the screen.

In SQL Developer, the substitution variable feature operates as shown in Figure 1.15.

After the value for the substitution variable is provided, the results of the execution are displayed in the Script Output window, as shown in Figure 1.16.
Figure 1.15 Using a Substitution Variable in SQL Developer

Figure 1.16 Using a Substitution Variable in SQL Developer: Script Output Window
In Figure 1.16, the substitution of the variable is shown in the Script Output window, and the result of the execution is shown in the Dbms Output window.

In SQL*Plus, the substitution variable feature operates as shown in Figure 1.17. Note that SQL*Plus does not list the complete PL/SQL block in its results, but rather displays the substitution operation only.

The preceding example uses a single ampersand for the substitution variable. When a single ampersand is used throughout the PL/SQL block, the user is asked to provide a value for each occurrence of the substitution variable.

**For Example  ch01_2a.sql**

```sql
BEGIN
  DBMS_OUTPUT.PUT_LINE ('Today is '||'&sv_day');
  DBMS_OUTPUT.PUT_LINE ('Tomorrow will be '||'&sv_day');
END;
```

When executing this example in either SQL Developer or SQL*Plus, you are prompted twice to provide the value for the substitution variable. This example produces the following output:

Today is Monday
Tomorrow will be Tuesday

![Figure 1.17 Using a Substitution Variable in SQL*Plus](image-url)
Did You Know?

When a substitution variable is used in the script, the output produced by the program contains the statements that show how the substitution was done. If you do not want to see these lines displayed in the output produced by the script, use the `SET` command option before you run the script:

```
SET VERIFY OFF;
```

This command is supported by both SQL Developer and SQL*Plus.

As demonstrated earlier, when the same substitution variable is used with a single ampersand, the user is prompted to provide a value for each occurrence of this variable in the script. To avoid this task, you can prefix the first occurrence of the substitution variable by the double ampersand (`&&`) character, as highlighted in bold in the following example.

**For Example  ch01_2b.sql**

```sql
BEGIN
    DBMS_OUTPUT.PUT_LINE ('Today is '||'&&sv_day');
    DBMS_OUTPUT.PUT_LINE ('Tomorrow will be '||'&sv_day');
END;
```

In this example, the substitution variable `sv_day` is prefixed by a double ampersand in the first `DBMS_OUTPUT.PUT_LINE` statement. As a result, this version of the example produces different output:

```
Today is Monday
Tomorrow will be Monday
```

From the output shown, it is clear that the user is asked only once to provide the value for the substitution variable `sv_day`. In turn, both `DBMS_OUTPUT.PUT_LINE` statements use the value of Monday entered by the user.

When a substitution variable is assigned to the string (text) data type, it is a good practice to enclose it with single quotes. You cannot always guarantee that a user will provide text information in single quotes. This practice, which will make your program less error prone, is illustrated in the following code fragment.

**For Example**

```sql
DECLARE
    v_course_no VARCHAR2(5) := '&sv_course_no';
```
As mentioned earlier, substitution variables are usually prefixed by the amper- 
sand (&) or double ampersand (&&) characters; these are the default characters 
that denote substitution variables. A special `SET` command option available in SQL 
Developer and SQL*Plus also allows you to change the default character to any 
other character or disable the substitution variable feature. This `SET` command 
has the following syntax:

<table>
<thead>
<tr>
<th>SET DEFINE character</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>SET DEFINE ON</code></td>
</tr>
<tr>
<td><code>SET DEFINE OFF</code></td>
</tr>
</tbody>
</table>

The first `SET` command option changes the prefix of the substitution variable 
from an ampersand to another character. Note, however, that this character cannot 
be alphanumeric or whitespace. The second (`ON` option) and third (`OFF` option) 
control whether SQL*Plus will look for substitution variables. In addition, the `ON` 
option changes the value of the character back to the ampersand.

**Summary**

In this chapter, you learned about PL/SQL architecture and how it may be used in 
a multitier environment. You also learned how PL/SQL can interact with users via 
substitution variables and the `DBMS_OUTPUT.PUT_LINE` statement. Finally, you 
learned about two PL/SQL development tools—SQL Developer and SQL*Plus. The 
examples shown in this chapter were executed in both tools to illustrate the differ-
ences between them. The main difference between the two is that SQL Developer 
has a graphical user interface and SQL*Plus has a command-line interface. The 
PL/SQL examples used throughout this book may be executed in either tool with 
the same results. Depending on your preference, you may choose one tool over the 
other. However, it is a good idea to become familiar with both, as these tools are 
part of almost every Oracle database installation.

**By the Way**

The companion website provides additional exercises and suggested answers for 
this chapter, with discussion related to how those answers resulted. The main 
purpose of these exercises is to help you test the depth of your understanding by 
utilizing all the skills that you have acquired throughout this chapter.
This page intentionally left blank
# Index

## Symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;</td>
<td>(ampersand), 21, 25</td>
</tr>
<tr>
<td>:=</td>
<td>(assignment operator), 35, 37</td>
</tr>
<tr>
<td>{ }</td>
<td>(braces), 372</td>
</tr>
<tr>
<td>--</td>
<td>comment notation, 28, 423</td>
</tr>
<tr>
<td>/*</td>
<td>comment notation, 28, 423</td>
</tr>
<tr>
<td>&gt;</td>
<td>(comparison operator), 393</td>
</tr>
<tr>
<td>&quot;</td>
<td>(double quotes), 31</td>
</tr>
<tr>
<td>/</td>
<td>(forward slash), 28, 354, 423</td>
</tr>
<tr>
<td>%FOUND</td>
<td>attribute, 167</td>
</tr>
<tr>
<td>%ISOPEN</td>
<td>attribute, 167, 171</td>
</tr>
<tr>
<td>%NOTFOUND</td>
<td>attribute, 167, 190</td>
</tr>
<tr>
<td>( )</td>
<td>(parentheses), 61</td>
</tr>
<tr>
<td>?</td>
<td>(question mark), 61</td>
</tr>
<tr>
<td>..</td>
<td>(range operator), 98</td>
</tr>
<tr>
<td>%ROWCOUNT</td>
<td>attribute, 167, 191</td>
</tr>
<tr>
<td>%ROWTYPE</td>
<td>attribute, 172, 174</td>
</tr>
</tbody>
</table>

## A

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>actual parameters</td>
<td>319–321</td>
</tr>
<tr>
<td>AFTER STATEMENT</td>
<td>sections, compound triggers, 223–227</td>
</tr>
<tr>
<td>AFTER triggers</td>
<td>210–211</td>
</tr>
<tr>
<td>aggregate qualified expressions</td>
<td>with index iterator association, 254–255</td>
</tr>
<tr>
<td>WITH section</td>
<td>with iterator association, 253–254</td>
</tr>
<tr>
<td>WITH trigger</td>
<td>with sequence iterator association, 255–257</td>
</tr>
<tr>
<td>ALTER TRIGGER</td>
<td>statement, 204</td>
</tr>
<tr>
<td>ALTER TYPE</td>
<td>statement, 405–410</td>
</tr>
<tr>
<td>ampersand (&amp;)</td>
<td>21, 25</td>
</tr>
<tr>
<td>anchored data types</td>
<td>33–35</td>
</tr>
<tr>
<td>anchored declarations</td>
<td>33</td>
</tr>
<tr>
<td>AND logical operator</td>
<td>61</td>
</tr>
</tbody>
</table>

## Architecture, PL/SQL

- See also functions; packages; procedures; triggers
- block structure
  - nested blocks, 36–38
  - overview of, 5–9, 128
  - sequences in, 43–44
- client-server architecture, 4–5
- code execution, 9–10
- DBMS_OUTPUT.PUT_LINE statement, 18–21
- development environment
  - overview of, 10
  - SQL Developer, 10–12
- engine, 2
- overview of, 2–5
- substitution variables, 21–25
- three-tier architecture, 1–2

## Anonymous blocks, 6
arrays
  associative
    creating, 231–233
    nested tables and
c  varrays (variable-size
arrays)
    associative arrays
    and nested tables
    compared to, 244
c  creating, 240–244
c  nested, 245
assignment operator (:=),
  35, 37
associative arrays
  creating, 231–233
  nested tables and
c  varrays compared to, 244
attributes, 380
autonomous transactions,
  211–213
AUTONOMOUS_ TRANSACTION
  pragma, 211–213
B
BEFORE EACH ROW
  section, compound
  triggers, 223–227
BEFORE STATEMENT
  section, compound
  triggers, 223–227
BEFORE triggers, 205–210
BEGIN keyword, 7
bind arguments,
  EXECUTE
    IMMEDIATE
    statement, 275–282
BIND_ARRAY function
  (DBMS_SQL), 412
BIND_VARIABLE function
  (DBMS_SQL), 412
BIND_VARIABLE_PKG
  function (DBMS_SQL), 412
  binding, 9
    with CLOSE statement,
    314–317
    with EXECUTE
      IMMEDIATE
      statement, 308–314
    with FETCH statement,
    314–317
    with OPEN FOR
      statement, 314–317
  block structure. See also
    functions; packages;
    procedures; triggers
    nested blocks, 36–38
    overview of, 5–9, 128
    sequences in, 43–44
  body
    of packages, 343–348
    of triggers, 204–205
  Boolean expressions,
    terminating loops
    with, 95
    braces ({}), 372
  built-in exceptions,
    128–133
    exception handling,
    130–133
    list of, 129–130
  RAISE_APPLICATION_ ERROR procedure,
    151–155
BULK COLLECT
  clause, 299–307
  with INSERT, UPDATE, and
  DELETE statements,
    303–307
  with SELECT
    statement, 299–303
  collections, binding in
    SQL statements
      with CLOSE
      statement, 314–317
      with EXECUTE
        IMMEDIATE
        statement, 308–314
      with FETCH
        statement, 314–317
      with OPEN FOR
        statement, 314–317
FORALL statements
  INDICES OF option,
    291, 296–297
  overview of, 290
  SAVE EXCEPTIONS
    option, 294–296
    structure and use of,
    290–293
  VALUES OF option,
    291, 297–299
    overview of, 289
C
c_course cursor, 184–186
c_name cursor, 269, 284
c_zip cursor, 182–184, 287
calc function, 334–335
calc_sum procedure,
  320–321
CASCADE option, 405
case
  formatting guidelines
    for, 421
  literals, 33
CASE expressions, 74–78
CASE statements
functions embedded in, 339
overview of, 66
searched
CASE versus searched CASE statements, 70–74
structure of, 68–70
structure of, 66–68
character sets, 27–28
check_total_rows function, 415
city_data_adm package, 363–366
city_tab variable, 388
city_tab_type table type, 387
classes. See also reserved words; statements
BULK COLLECT, 299–307
with INSERT, UPDATE, and DELETE statements, 303–307
with SELECT statement, 299–303
COMPOUND TRIGGER, 223–224
CREATE OR REPLACE TYPE, 381–384
DETERMINISTIC, 336, 374
ELSE, 66, 76
CASE statements, 66, 76
ELSIF, 55–59
IF-THEN-ELSE statement, 52–55
END CASE, 72
FOLLOW/PRECEDES, 203–204
INDEX BY, 231
INTO, 275, 282
ISOLATION LEVEL, 47
NAME, 47
PARALLEL_ENABLE, 336, 374
PIPELINED, 336, 368–370, 374
READ ONLY, 47
READ WRITE, 47
RESULT_CACHE, 363–366, 374
RETURN, 332–333, 370
EXECUTE IMMEDIATE statement, 275
IN OUT parameter, 391
SELF parameter, 389, 391
RETURNING, 304
RETURNING INTO, 41–42, 275
SQL_MACRO, 370–375
USE ROLLBACK SEGMENT, 47
USING
EXECUTE IMMEDIATE statement, 275, 280–282
OPEN FOR statement, 284
WHEN, 66, 76
WHERE CURRENT OF, 198
collection iteration controls, 245–247
collection methods, 236–240
collections
binding in SQL statements
with CLOSE statement, 314–317
with EXECUTE IMMEDIATE statement, 308–314
with FETCH statement, 314–317
with OPEN FOR statement, 314–317
collection methods, 236–240
empty, 236
iteration controls, 247–251
multidimensional, 245–247
object types with, 385–388
overview of, 229
qualified expressions with, 247–251
aggregate with index iterator association, 254–255
aggregate with iterator association, 253–254
aggregate with sequence iterator association, 255–257
empty, 253
simple, 253
syntax for, 251–253
tables
associative arrays, 231–233, 244
definition of, 230
nested, 233–236
collections (continued)
  varrays (variable-size arrays)
  associative arrays
  and nested tables
  compared to, 244
  creating, 240–244
  nested, 245

column objects, storing
  in relational tables, 400–403
COLUMN_VALUE function
  (DBMS_SQL), 412, 417
COLUMN_VALUE_LONG
  function (DBMS_SQL), 412
columns
  naming conventions, 421–422
STUDENT database
  schema
  COURSE table, 425–426
  ENROLLMENT
  table, 427–428
  GRADE table, 430–431
  GRADE_TYPE table, 429
  GRADE_TYPE_WEIGHT table, 430
  INSTRUCTOR table, 428
  SECTION table, 426
  STUDENT table, 427
  ZIPCODE table, 429
comments
  definition of, 28
  formatting guidelines
    for, 423
COMMIT statement, 44–47
  comparing objects
    with map methods, 394–395
    with order methods, 395–398
  comparison operator (>), 393
  compatibility, record, 263–265
  compilation errors, 125–128
  compile-time constants, 351
  COMPOUND TRIGGER
    clause, 223–224
  compound triggers
    creating, 223–224
  resolving mutating table
    issues with, 223–227
    restrictions, 224–225
    structure of, 223–224
  concatenation operator, 32
  conditional control
    CASE expressions, 74–78
    CASE statements
      CASE versus searched CASE
      statements, 70–74
      COALESCE function, 80–82
      NULLIF function, 78–80
    overview of, 66
    searched, 68–74
    structure of, 66–68
  IF statements
    ELSIF, 55–59
    IF-THEN, 50–52
    IF-THEN-ELSE, 52–55
    importance of, 49
    inner versus outer, 60
    logical operators, 61
    nested, 59–60, 62
  NULL conditions, 54–55
  connections, database
    creating with SQL
      Developer, 10–12
      creating with SQL*Plus, 13–14
  constants
    compile-time, 351
    naming conventions, 422
  constructor methods, 389–392
  context switches, 371
  CONTINUE statement, 112–115
  CONTINUE WHEN
    statement, 116–118
  COUNT method, 237–240, 243
  counters, loop, 88
  COURSE table schema, 425–426
  course_rec record, 172–174
  course_sections macro, 373–375
  CREATE FUNCTION
    statement, 336–339
  SQL_MACRO clause, 370–375
  stand-alone procedures,
    creating, 327–330
  CREATE keyword, 202–203
  CREATE OR REPLACE
    TYPE BODY
    statement, 409
  CREATE OR REPLACE
    TYPE clause, 381–384
  CREATE PROCEDURE
    statement, 327–330
  CREATE TABLE
    statement, 278, 403–404
CREATE TRIGGER statement, 213–214
Create Trigger window, 206–210
CREATE TYPE statement, 234, 241
CREATE VIEW privilege, 215
CURRVAL operator, 43
CURSOR keyword, 165
cursor specification section, packages, 343–344
cursor-based records, 174–175
cursors
c._name, 269
cursor expressions, 193–196
cursor loops
cursor FOR loops, 175–177
processing with DBMS_SQL, 418–420
cursor variables, 187–193
cursor-based records, 174–175
definition of, 161
explicit
declaring, 164–165
fetching, 166–171
opening/closing, 165–166
overview of, 164
FOR UPDATE, 196–199
implicit, 162–164
INVALID_CURSOR exception, 191–193
naming conventions, 165
nested, 177–179
parameterized, 181–186
SQL, 162
cv_course variable, 193
cv_student_name variable, 195
D
Data Manipulation Language (DML) statements, 41–42
data types
anchored, 33–35
VARCHAR2, 77
weak, 187
database character sets, 27–28
Database Object-Relational Developer’s Guide, 379, 399
database triggers
AFTER, 210–211
autonomous transactions, 211–213
BEFORE, 205–210
creating, 202–204
definition of, 202–205
destination versus nondestination, 203
enabling/disabling, 202–204
firing, 202, 203–204
INSTEAD OF, 215–219
names, 203
row triggers, 213–214
statement triggers, 213–214
trigger headers, 204–205
triggering events, 202
databases, connecting to/disconnecting from with SQL Developer, 10–12
with SQL*Plus, 13–14
date_time_info_adm package, 348, 351
creating, 349–350
instantiating and initializing, 349–350
package state, 351
DBMS_OUTPUT.PUT_LINE statement, 18–21
DBMS_RESULT_CASH_FLUSH procedure, 366
DBMS_SQL package, generating dynamic SQL with
cursor loop processing with, 418–420
execution flow for, 411–413
multirrow SELECT statement, 416–420
table_adm_pkg package example
creating/replacing, 415–416
testing, 416
DBMS_UTILITY package, 293
declaration, forward, 324–325
declaration section packages, 343–344
PL/SQL blocks, 6
DECLARE keyword, 6
DEFINE_ARRAY function (DBMS_SQL), 412, 419
DEFINE_COLUMN function (DBMS_SQL), 412, 415
DEFINE_COLUMN_LONG function (DBMS_SQL), 412
DELETE method, 237–240, 243
DELETE statement, 41–42
BULK COLLECT clause, 303–307
RETURNING INTO clause, 42
WHERE CURRENT OF clause, 198
delete_table_data function, 415
dels_test procedure, 311
DELETING function, 210
delimiters, 28, 32, 372
DETERMINISTIC clause, 336, 374
development environment
overview of, 10
script execution, 14–16
SQL Developer
database connections, creating/terminating, 10–12
script execution, 10–12
substitution
variables, 21–25
SQL*Plus
database connections, creating/terminating, 13–14
script execution, 13–14
substitution
variables, 21–25
diff function, 332
display procedure,
overloading, 357–358
display_data procedure, 403
display_person procedure, 402–403
display_student_info
procedure, 316–317
display_zipcode_info
method, 393
DML (Data Manipulation Language)
statements, 41–42
dot notation, 346
double quotes ("), 31
double-pipe ( | | ) delimiter, 32
DUP_VALUE_ON_INDEX
exception, 130
dynamic SQL
generating with DBMS_SQL package
cursor loop processing with, 418–420
execution flow for, 411–413
multirow SELECT statement, 416–420
table_adm_pkg package, creating/replacing, 413–415
table_adm_pkg package, testing, 416
native dynamic SQL
CLOSE statement, 283–287
EXECUTE IMMEDIATE statement, 274–283
FETCH statement, 283–287
OPEN FOR statement, 283–287
overview of, 273
e_invalid_id exception, 140–141
EDITIONABLE keyword, 203, 328
ditioning, trigger, 203
ELSE key
CASE statements, 66, 76
ELSIF statement, 55–59
IF-THEN-ELSE statement, 52–55
ELSIF statement, 55–59, 334–335
embedded functions, 335
empty collections, 236
empty qualified expressions, 253
ENABLE/DISABLE option, 203–204
enclosing records, 265
END CASE clause, 72
END keyword, 7
END LOOP statement, 89, 121
e, 271
enroll_adm_pkg
package, creating/replacing, 413–415
statement, 416
ENROLLMENT table
schema, 427–428
error handling
compilation versus runtime errors, 125–128
error messages, creating
RAISE_APPLICATION_ERROR procedure, 151–155
SQLCODE function, 157–160
exceptions
built-in, 128–133
DUP_VALUE_ON_INDEX, 130
exception handlers, 127–128
EXCEPTION_INIT pragma, 155–157
internally defined, 155–157
NO_DATA_FOUN
191–193
ORA-06530, 384
propagating, 143–148
RAISE_APPLICATION_ERROR procedure, 151–155
re-raising, 148–149
scope of, 135–138
SQLCODE function, 157–160
SQLERRM function, 157–160
user-defined, 139–143
VALUE_ERROR, 137
EXECUTE IMMEDIATE statement, 276–282
importance of, 125–128
mutating table issues
description of, 221–223
resolving with compound triggers, 223–227
packages, 347–348
ERROR_CODE field, 294–296
ERROR_INDEX field, 294–296
events, triggering, 202
evolution, type, 405–410
exc_ind_type data type, 298
EXCEPTION keyword, 8, 31, 139
exception variable, 31
EXCEPTION_INIT
pragma, 155–157
internally defined, 155–157
INVALID_CURSOR, 191–193
NO_DATA_FOUND, 136–138, 232, 248
ORA-06530, 384
propagating, 143–148
RAISE_APPLICATION_ERROR procedure, 151–155
re-raising, 148–149
scope of, 135–138
SQLCODE function, 157–160
SQLERRM function, 157–160
user-defined, 139–143
VALUE_ERROR, 137
executable section, PL/SQL blocks, 7
EXECUTE function
(DBMS_SQL), 412, 415, 417
EXECUTE IMMEDIATE statement, 274–283, 308–314
with CLOSE statement, 314–317
error handling, 276–282
with FETCH statement, 314–317
NULL values, passing, 282–283
with OPEN FOR statement, 314–317
structure of, 274–276
EXECUTE_AND_FETCH function (DBMS_SQL), 412
execution of code, 9–10, 14–16
EXISTS method, 237–240
EXIT statement, 13, 87–90
EXIT WHEN statement, 91–92
explicit cursors
declaring, 164–165
fetching, 166–171
opening/closing, 165–166
overview of, 164
expressions
CASE, 74–78
cursor, 193–196
qualified, 262
expressions, qualified
with collections, 251–257
aggregate with index iterator association, 254–255
aggregate with iterator association, 253–254
aggregate with sequence iterator association, 255–257
empty, 253
simple, 253
syntax for, 251–253
with user-defined records, 262
EXTEND method, 236–240
F
FETCH statement, 166–171, 189, 283–287, 314–317
FETCH_ROWS function
(DBMS_SQL), 412, 417
fetching cursors, 166–171
firing order, triggers, 203–204
firing triggers, 202
FIRST method, 237–240, 243
first&last_names variable, 30
instantiation of packages, 349–350
INSTEAD OF triggers, 215–219
INSTRUCTOR table schema, 428
internally defined exceptions, 155–157
INTO clause, EXECUTE IMMEDIATE statement, 275, 282
INVALID_CURSOR exception, 191–193
INVALID_NUMBER error, 137
INVALIDATE option, 405
is_number function, 335
ISOLATION LEVEL clause, 47
%ISOPEN attribute, 167, 171
iterative control
collection iteration controls, 245–247
CONTINUE statement, 112–115
CONTINUE WHEN statement, 116–118
loops
with Boolean expressions, 95
FOR, 97–109, 175–177
infinite, 87
labels, 120–122
loop counters, 88
loop headers, 99
multiple iterations, 107–108
nested, 119–120
simple, 86–92
single expression iteration, 106–107
stepped range iteration, 104–106
terminating prematurely, 108–109
WHILE, 92–97
iterator associations, aggregate qualified expressions with, 253–255
J-K-L
keywords. See reserved words
labels
loop, 120–122
nested, 36–38
LAST method, 237–240, 243
last_name_tab associative array, 231–232
last_name_table, 234–236
last_name_type, 268
leaf-level attributes, 399
LIMIT method, 243
LIMIT option, BULK
COLLECT clause, 301–302
literals, 28, 33
logical operators, 61
LOGIN_DENIED exception, 130
FOR.LOOP statement, 99
loop_counter variable, 98
loops
CONTINUE statement, 112–115
CONTINUE WHEN statement, 116–118
FOR, 97–103
cursor FOR loops, 175–177
IN option, 100–103
IN REVERSE option, 103–104
multiple iterations, 107–108
single expression iteration, 106–107
stepped range iteration, 104–106
terminating prematurely, 108–109
infinite, 87
labels, 120–122
loop counters, 88
loop headers, 99
multiple iterations, 106–107
nested, 119–120
simple
with EXIT statement, 87–90
with EXIT WHEN statement, 91–92
structure of, 86–87
single expression iteration, 106–107
stepped range iteration, 104–106
terminating
with Boolean expressions, 95
with EXIT statement, 87–90
with EXIT WHEN statement, 91–92
terminating prematurely, 108–109
WHILE, 92–97
M
macros, 370–375
map methods, 394–395
mathematical symbols, 29
member methods, 392
MERGE statement, 41–42
messages, error
RAISE_APPLICATION_ERROR procedure, 151–155
messages, error (continued)
  SQLCODE function, 157–160
  SQLERRM function, 157–160
methods
  collection, 236–240
  definition of, 380
  object type
    constructor methods, 389–392
    map methods, 394–395
    member methods, 392
    order methods, 395–398
  overview of, 388–389
  static methods, 393
  public, 382
mixed notation, 324–325
MOD function, 68, 71
modular code, 319
multidimensional
  collections, 245–247
  multiple iterations, 106–107
multirow queries, 283–287
multirow SELECT
  statement, 416–420
mutating tables
  description of, 221–223
  resolving with compound triggers, 223–227

N
NAME clause, SET
  TRANSACTION statement, 47
name_type record type, 266
named blocks, 6
named notation, 324–325, 330, 390
naming conventions, 421–422
cursors, 165
variables, 29–31
national character sets, 27–28
native character sets, 27–28
native code, 9
native dynamic SQL
  CLOSE statement, 283–287
  EXECUTE IMMEDIATE statement, 274–283
  error handling, 276–282
  NULL values, passing, 282–283
  structure of, 274–276
  FETCH statement, 283–287
  OPEN FOR statement, 283–287
  overview of, 273
nesting
  blocks, 36–38
  cursors, 177–179
  functions, 331–335
  IF statements, 59–60, 62
  labels, 36–38
  loops, 119–120
  procedures
    formal versus actual parameters, 319–321
    forward declaration, 324–325
    parameter modes, 321–325
  records, 265–268
  subprograms
    definition of, 319
    overloading, 357–363
tables
  associative arrays
    and varrays
  compared to, 244
  creating, 233–236
  varrays (variable-size arrays), 245
:NEW pseudorecord, 226
New Trigger option,
  Worksheet window, 206–210
New/Select Database
  Connection dialog box, 10–11
NEXT method, 237–240
NEXTVAL, 43
NOCOPY, 391
NONEDITABLE
  keyword, 203, 328
NOT INCLUDING TABLE
  DATA phrase, 406
NOT NULL constraint, 34, 260–261
notation types, 323–325, 330
%NOTFOUND attribute, 167, 190
NULL values
  CASE statements, 68
  empty collections versus, 236
  with IF statements, 54–55
  object types, 383–384
  passing, 282–283
  procedures and, 323
  NULLIF function, 78–80
  numeric FOR loops, 97–103
  IN option, 100–103
  IN REVERSE option, 103–104
  multiple iterations, 107–108
  single expression iteration, 106–107
  stepped range iteration, 104–106
  structure of, 97–100
  terminating prematurely, 108–109
NVL function, 80
object tables
comparing objects
with map methods, 394–395
with order methods, 395–398
public interfaces, 382
storing objects in, 403–405
type evolution, 405–410
object types
with collections, 385–388
creating, 381–384
NULL values, 383–384
object type methods
constructor methods, 389–392
map methods, 394–395
member methods, 392
order methods, 395–398
overview of, 388–389
static methods, 393
overview of, 379
public objects, 342
result objects, 365
storing in tables
object tables, 403–405
overview of, 399–400
relational tables, 400–403
type evolution, 405–410
structure of, 380
OPEN FOR statement, 189, 283–287, 314–317
OPEN statement, 166
OPEN_CURSOR function (DBMS_SQL), 412, 415
operators
comparison (>, 393
CURRVAL, 43
logical, 61
range (.), 98
OR logical operator, 61
ORA-06530 exception, 384
Oracle Application Express
(APLEX), 2
Oracle Forms and Reports,
2
Oracle Fusion Middleware,
2
order methods, 395–398
OTHERS exception
handler, 132–133
OUT parameter, 322, 391, 395
outer IF statements, 60
overloading subprograms, 357–363
P
p_prereq cursor parameter, 185–186
p_state cursor parameter, 182–184
p_students_tab collection, 346
packages
benefits of, 341–342
creating
package body, 343–348
package specification, 342–343
DBMS_SQL, generating
dynamic SQL with
cursor loop processing
with, 418–420
execution flow for, 411–413
multirow SELECT
statement, 416–420
table_adm_pkg
package, creating/
replacing, 413–415
table_adm_pkg
package, testing, 416
DBMS_UTILITY, 293
error handling, 347–348
initializing, 349–350
instantiating, 349–350
naming conventions, 421–422
package state, 351
package subprograms, 319, 357–363
SERIALLY_REUSABLE, 351–356
stateful/stateless, 351
PARALLEL_ENABLE
clause, 336, 374
param_modes procedure, 322–324
parameterized cursors, 181–186
parameters
for cursors, 181–186
for nested procedures
formal versus actual, 319–321
forward declaration, 324–325
notation types, 323–325
parameter modes, 321–325
parentheses, 61
PARSE function
(DBMS_SQL), 415
parse trees, 9
p-code, 9
PERSON table, 401–403
person_obj_type objects, 400–403
person_rec record, 265–266
person_type record type, 266
pipe (|), 32
PIPE ROW statement, 370
PIPEDLENE clause, 336, 368–370, 374
pipelined table functions, 368–370
PLS_INTEGER type, 231, 298
plsql_block variable, 279
PLSQL_CODE_TYPE parameter, 9
pluggable database container (PDB), 13
populate_student_tab procedure, 316–317
populate_test procedure, 311
populate_test_rec procedure, 313
positional notation, 323–325, 390
pragmas AUTONOMOUS_TRANSACTION, 211–213
EXCEPTION_INIT, 155–157, 212
SERIALLY_REUSABLE, 351–356
print_student_data procedure, 313
naming conventions, 421–422
nested, creating formal versus actual parameters, 319–321
forward declaration, 324–325
parameter modes, 321–325
notation types, 323–325, 330
stand-alone, creating, 327–330
PROGRAM_ERROR exception, 130
propagating exceptions, 143–148
pseudorecords, 205–206
public interfaces, 382
public methods, 382
public objects, 342
punctuation characters, 28
purity rules, 367–368
PUT_LINE statement, 18–21
Q qualified expressions with collections, 251–257
aggregate with index iterator association, 254–255
aggregate with iterator association, 253–254
aggregate with sequence iterator association, 255–257
empty, 253
simple, 253
syntax for, 251–253
with user-defined records, 262
queries executing, 14–16
multirow, 283–287
stored, 215
view, 216
question mark (?), 61
QUIT command, 13
R RAISE statement, 140, 142, 151–155
RAISE_APPLICATION_ERROR procedure, 151–155
range operator (.), 98
READ ONLY clause, SET TRANSACTION statement, 47
READ WRITE clause, SET TRANSACTION statement, 47
records collections of, 268–271
cursor-based, 174–175
enclosing, 265
nested, 265–268
pseudorecords, 205–206
table-based, 172–174
user-defined creating, 260–262
qualified expressions with, 262
record compatibility, 263–265
REF CURSOR type, 187, 194, 284, 315, 317
relational tables, storing objects in, 400–403
REPLACE keyword, 202–203
re-raising exceptions, 148–149
reserved words. See also clauses; statements
BEGIN, 7
CREATE, 202–203
CURSOR, 165
DECLARE, 6, 7
definition of, 28
EDITABLE, 203, 328
ELSE
CASE statements, 66, 76
ELSIF statement, 55–59
IF-THEN-ELSE statement, 52–55
END, 7
EXCEPTION, 8, 31, 139
formatting guidelines for, 421
NONEDITIONABLE, 203, 328
overview of, 31–32
REPLACE, 202–203
result objects, 365
RESULT_CACHE clause, 336, 363–366, 374
result-cached functions, 363–366
RETURN clause, 332–333
EXECUTE IMMEDIATE statement, 275
IN OUT parameter, 391
SELF parameter, 389, 391
return types, 332–333
RETURNING clause, 304
RETURNING INTO clause, 41–42, 275
REVERSE option, loops, 103–104
ROLLBACK statement, 44–47, 198
row objects, storing in object tables, 403–405
row triggers, 213–214
row_text_tab associative array, 298
%ROWCOUNT attribute, 167, 191
%ROWTYPE attribute, 172, 174
rules, purity, 367–368
runtime errors
error handling, 125–128
mutating tables
description of, 221–223
resolving with compound triggers, 223–227
semantic checking, 9
sequence iterator associations,
aggregate qualified expressions with,
255–257
sequences, 43–44
SERIALLY_REUSABLE packages, 351–356
SET command, 25
SET SERVEROUTPUT statement, 20
SET TRANSACTION statement, 47–48
SGA (system global area), 352
show_description function, 336–337, 367–368
show_enrollment function, 338–339
simple loops
with EXIT statement, 87–90
with EXIT WHEN statement, 91–92
nested, 119–120
structure of, 86–87
simple qualified expressions, 253
single expression iteration, 106–107
SQL cursors, 162
SQL Developer
database connections,
creating/terminating, 10–12
script execution, 10–12
substitution variables, 21–25
SQL macros, 370–375
SQL statements. See statements
SQL%BULK_EXCEPTIONS attribute, 294–296
SQL\%FOUND attribute, 163–164
SQL\%ISOPEN attribute, 163–164
SQL\%NOTFOUND attribute, 163–164
SQL\%ROWCOUNT attribute, 163–164
SQL*Plus database connections, creating/terminating, 13–14
script execution, 13–14
substitution variables, 21–25
SQL_MACRO clause, 370–375
SQLCODE function, 157–160
SQLERRM function, 157–160
sqlplus command, 13
stand-alone functions, 336–339
stand-alone subprograms, 319
state_obj_type object type, 387
stateful packages, 351
stateless packages, 351
statement triggers, 213–214
statements, 34, 47–48
ALTER TRIGGER, 204
ALTER TYPE, 405–410
binding collections in with CLOSE
statement, 314–317
with EXECUTE
IMMEDIATE statement, 308–314
with FETCH
statement, 314–317
with OPEN FOR
statement, 314–317
BULK SELECT, 385, 403
CASE
CASE versus searched CASE, 70–74
COALESCE function, 80–82
functions embedded in, 339
NULLIF function, 78–80
overview of, 66
searched, 68–74
structure of, 66–68
CLOSE, 166, 189, 283–287, 314–317
COMMIT, 44–47
CONTINUE, 112–115
CONTINUE WHEN, 116–118
CREATE FUNCTION, 336–339
SQL_MACRO clause, 370–375
stand-alone procedures, creating, 327–330
CREATE OR REPLACE TYPE BODY, 409
CREATE PROCEDURE, 327–330
CREATE TABLE, 278, 403–404
CREATE TRIGGER, 213–214
CREATE TYPE, 234, 241
DBMS_OUTPUT.PUT_LINE, 18–21
DELETE, 41–42
BULK COLLECT clause, 303–307
RETURNING INTO clause, 42
WHERE CURRENT OF clause, 198–199
DML (Data Manipulation Language), 41–42
ELSIF, 334–335
END LOOP, 89, 121
EXECUTE IMMEDIATE, 274–283, 308–314
error handling, 276–282
NULL values, passing, 282–283
structure of, 274–276
EXIT, 13, 87–90
EXIT WHEN, 91–92
FETCH, 166–171, 189, 283–287, 314–317
FOR EACH ROW, 203, 213–214
FORALL INDICES OF option, 291, 296–297
overview of, 290
SAVE EXCEPTIONS option, 294–296
structure and use of, 290–293
VALUES OF option, 291, 297–299
IF, 142
ELSIF, 55–59
IF-THEN, 50–52
IF-THEN-ELSE, 52–55, 393
importance of, 49
inner versus outer, 60
logical operators, 61
nested, 59–60, 62
NULL conditions, 54–55
INSERT, 41–42, 209–210
BULK COLLECT clause, 303–307
person_obj_type objects, 404
invoking functions in, 367–368
FOR.LOOP, 99
MERGE, 41–42
OPEN, 166
OPEN FOR, 189, 283–287, 314–317
PIPE ROW, 370
PRAGMA SERIALLY_REUSABLE, 352
RAISE, 140, 142, 151–155
RETURN, 332–333, 370
ROLLBACK, 44–47, 198
SAVEPOINT, 44–47
SELECT, 4. See also cursors
   BULK COLLECT clause, 299–307
   DBMS_SQL package and, 416–420
dynamic SQL, 283–287
   invoking functions in, 367–368
   person_obj_type objects, 404
   VALUE function, 406
SELECT FOR UPDATE, 196–199
SELECT INTO, 7, 337
   initializing variables with, 40–41
   mutating table issues, 221–227
   object attributes, 383–384
TABLE, 368
TYPE, 187, 231, 241, 260
UPDATING, 41–42
   autonomous transactions, 211–213
   BULK COLLECT clause, 303–307
   mutating table issues, 221–227
person_obj_type objects, 405
SELECT FOR UPDATE and, 196–199
WHERE CURRENT OF clause, 198–199
   static methods, 393
   stepped range iteration, 104–106
   stored code. See also packages functions. See also
   individual functions
   definition of, 331–332
   embedding, 335
   invoking in SQL statements, 367–368
   nested, 331–335
   pipelined table, 368–370
   purity rules, 367–368
   result-cached, 363–366
   return types, 332–333
   SQL macros, 370–375
   stand-alone, 336–339
procedures. See also
   individual procedures
   forward declaration, 326–327
   naming conventions, 421–422
   nested, creating, 319–325
   notation types, 323–325, 330
   stand-alone, creating, 327–330
   subprogram overloading, 357–363
   stored queries, 215
   stored subprograms, 319
STUDENT database schema
   COURSE table, 425–426
database connections, creating/terminating
   with SQL Developer, 10–12
   with SQL*Plus, 13–14
ENROLLMENT table, 427–428
   entity relationship diagram, 432
GRADE table, 430–431
GRADE_CONVERSION table, 431
GRADE_TYPE table, 429
GRADE_TYPE_WEIGHT table, 430
INSTRUCTOR table, 428
PERSON table, 401–403
SECTION table, 426
STUDENT table, 223–227, 427
ZIPCODE table, 429
STUDENT table, 223–227, 427
   student_adm package
   overloaded procedures in, 358–362
   package body, 344–348
   package specification, 342–343
   with record and collection types, 316–317
STUDENT_BI trigger, 205–210
   student_cur cursor variable, 315, 346
   student_cur_var variable, 315
   student_cur_typ cursor type, 315
   student_rec variable, 315
   subprograms, 319–321
Index

definition of, 319
forward declaration, 326–327
overloading, 357–363
placement of definition of, 321
procedures. See also individual procedures
forward declaration, 326–327
naming conventions, 421–422
nested, creating, 319–325
stand-alone, creating, 327–330
subscript notation, 231
substitution variables, 21–25
sv_table_name variable, 276
syntax checking, 9
syntax errors, 125–128
SYS_REFCURSOR type, 187
system global area (SGA), 352
system_date variable, 349

T

tab_type table type, 368
table expression, 371
TABLE function, 317, 386
table functions, pipelined, 368–370
table SQL macros, 370–375
TABLE statement, 368
TABLE type, 371
table_adm_pkg package
  creating/replacing with DBMS_SQL, 413–415
testing, 416
table_exists function, 415
table-based records, 172–174

tables
  associative arrays
    creating, 231–233
    nested tables and varrays compared to, 244
  definition of, 230
  mutating
description of, 221–223
resolving with compound triggers, 223–227
naming conventions, 421–422
nested
  associative arrays and varrays compared to, 244
  creating, 233–236
pipelined table functions, 368–370
records
cursor-based, 174–175
table-based, 172–174
storing object types in object tables, 403–405
overview of, 399–400
relational tables, 400–403
type evolution, 405–410
STUDENT database
  schema
  COURSE, 425–426
  ENROLLMENT, 427–428
entity relationship
diagram, 432
GRADE, 430–431
GRADE_CONV, 431
GRADE_TYPE, 429
GRADE_TYPE_WEIGHT, 430
  INSTRUCTOR, 428
  PERSON, 401–403
  SECTION, 426
  STUDENT, 427
  ZIPCODE, 429
terminating loops
  with Boolean expressions, 95
  with EXIT statement, 87–90
  with EXIT WHEN statement, 91–92
terminating prematurely, 108–109
TEST_EXC table, 298
testing table_adm_pkg package, 416
time_rec record, 260–262
time_rec_type record, 260–262
TO_CHAR function, 40, 77, 231, 358
TOO_MANY_ROWS exception, 130, 131
transactions
  autonomous, 211–213
  transaction control
    COMMIT statement, 44–47
    ROLLBACK statement, 44–47
    SAVEPOINT statement, 44–47
    SET TRANSACTION statement, 47–48
triggering events, 202
triggers
  AFTER, 210–211
  autonomous transactions, 211–213
  BEFORE, 205–210
  compound, 223–227
  creating, 223–224
  restrictions, 224–225
  structure of, 223–224
<table>
<thead>
<tr>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>creating, 202–204</td>
</tr>
<tr>
<td>definition of, 202–205</td>
</tr>
<tr>
<td>editioned versus noneditioned, 203</td>
</tr>
<tr>
<td>enabling/disabling, 202–204</td>
</tr>
<tr>
<td>firing, 202, 203–204</td>
</tr>
<tr>
<td>INSTEAD OF, 215–219</td>
</tr>
<tr>
<td>mutating table issues description of, 221–223</td>
</tr>
<tr>
<td>resolution with compound triggers, 223–227</td>
</tr>
<tr>
<td>names, 203</td>
</tr>
<tr>
<td>row, 213–214</td>
</tr>
<tr>
<td>statement, 213–214</td>
</tr>
<tr>
<td>trigger headers, 204–205</td>
</tr>
<tr>
<td>triggering events, 202</td>
</tr>
<tr>
<td>TRIM method, 237–240</td>
</tr>
<tr>
<td>type evolution, 405–410</td>
</tr>
<tr>
<td>TYPE statement, 187, 231, 241, 260</td>
</tr>
<tr>
<td>typemark, 251</td>
</tr>
<tr>
<td>types, object. See object types</td>
</tr>
<tr>
<td>v_area variable, 280</td>
</tr>
<tr>
<td>v_city variable, 195</td>
</tr>
<tr>
<td>v_counter variable, 88–109</td>
</tr>
<tr>
<td>v_date_rec variable, 348</td>
</tr>
<tr>
<td>v_diff variable, 332</td>
</tr>
<tr>
<td>v_dob variable, 30</td>
</tr>
<tr>
<td>v_dyn_sql variable, 311</td>
</tr>
<tr>
<td>v_err_code variable, 153</td>
</tr>
<tr>
<td>v_err_msg variable, 153</td>
</tr>
<tr>
<td>v_final_grade variable, 73</td>
</tr>
<tr>
<td>v_first_name variable, 6–7</td>
</tr>
<tr>
<td>v_index variable, 268</td>
</tr>
<tr>
<td>v_instructor_id variable, 133</td>
</tr>
<tr>
<td>v_last_name variable, 6</td>
</tr>
<tr>
<td>v_letter_grade variable, 73</td>
</tr>
<tr>
<td>v_my_table_name variable, 277</td>
</tr>
<tr>
<td>v_name variable, 30</td>
</tr>
<tr>
<td>v_new_name variable, 34</td>
</tr>
<tr>
<td>v_num variable, 57, 79, 327</td>
</tr>
<tr>
<td>v_num_flag variable, 68, 70, 71, 76</td>
</tr>
<tr>
<td>v_num1 variable, 51–52, 126–127, 321</td>
</tr>
<tr>
<td>v_num2 variable, 52, 126–127, 321</td>
</tr>
<tr>
<td>v_numeric_grade variable, 34</td>
</tr>
<tr>
<td>v_prereq variable, 77, 185–186, 193</td>
</tr>
<tr>
<td>v_remainder variable, 79</td>
</tr>
<tr>
<td>v_result variable, 75, 126, 397</td>
</tr>
<tr>
<td>v_seq_value variable, 43–44</td>
</tr>
<tr>
<td>v_state variable, 195</td>
</tr>
<tr>
<td>v_student_id variable, 132, 136, 136–142, 153</td>
</tr>
<tr>
<td>v_sum variable, 321</td>
</tr>
<tr>
<td>v_table_name variable, 285–287</td>
</tr>
<tr>
<td>v_temp variable, 51–52</td>
</tr>
<tr>
<td>v_total_rows variable, 276</td>
</tr>
<tr>
<td>v_trans_type variable, 211</td>
</tr>
<tr>
<td>v_zip variable, 195, 268, 278</td>
</tr>
<tr>
<td>v_zip_rec variable, 287</td>
</tr>
<tr>
<td>VALUE function, 406</td>
</tr>
<tr>
<td>VALUE_ERROR exception, 130, 137</td>
</tr>
<tr>
<td>VALUES OF option, 247–251, 291, 297–299</td>
</tr>
<tr>
<td>VARCHAR2 data type, 77, 337</td>
</tr>
<tr>
<td>VARIABLE_VALUE function (DBMS_SQL), 412</td>
</tr>
<tr>
<td>VARIABLE_VALUE_PKG function (DBMS_SQL), 412</td>
</tr>
<tr>
<td>variables. See also individual variables</td>
</tr>
<tr>
<td>cursor variables, 187–193</td>
</tr>
</tbody>
</table>
variables (continued)
  declaring, 29
  initializing, 40–41
  naming conventions, 29–31, 421–422
  overview of, 29–31
  scope of, 35
  substitution, 21–25
varrays (variable-size arrays)
  associative arrays
  and nested tables
  compared to, 244
  creating, 240–244
  nested, 245
  view query, 216
views, 215

W
  weak types, 187
  WHEN clause, 66, 76
  WHERE CURRENT OF clause, 198
  WHILE loops, 92–97, 119–120
  whitespace, formatting guidelines for, 28, 421–422

X-Y-Z
  ZERO_DIVIDE exception, 130
  zip_code_obj_type object type, 383
  zip_cur cursor, 195
  zip_info_rec record, 268
  zip_is_good function, 346
  zip_rec_type, 287
  zip_tab variable, 388
  zip_tab_type table type, 386, 387
  zip_type array type, 385
  ZIPCODE table schema, 429
  zipcode_obj_type, 382–383, 389–392, 396