



Oracle[®] PL/SQL

by Example

Fifth Edition

- ▶ Updated for Oracle 12c
- ▶ Hundreds of examples, questions, and answers
- ▶ Real-life labs
- ▶ No Oracle PL/SQL experience necessary
- ▶ Build PL/SQL Applications—NOW

BENJAMIN ROSENZWEIG • ELENA RAKHIMOV

Oracle® PL/SQL by Example

Fifth Edition

This page intentionally left blank



Oracle® PL/SQL by Example

Fifth Edition

Benjamin Rosenzweig
Elena Rakhimov

◆◆ Addison-Wesley

Boston • Columbus • New York • San Francisco • Amsterdam • Cape Town
Dubai • London • Madrid • Milan • Munich • Paris • Montreal • Toronto • Delhi
Mexico City • São Paulo • Sydney • Hong Kong • Seoul • Singapore • Taipei • Tokyo

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

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

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.

Visit us on the Web: informit.com

Library of Congress Cataloging-in-Publication Data

Rosenzweig, Benjamin.

Oracle PL/SQ[®] by example / Benjamin Rosenzweig, Elena Rakhimov.—Fifth edition.

pages cm

Includes index.

ISBN 978-0-13-379678-0 (pbk. : alk. paper)—ISBN 0-13-379678-7 (pbk. : alk. paper)

1. PL/SQL (Computer program language) 2. Oracle (Computer file) 3. Relational databases.

I. Rakhimov, Elena Silvestrova. II. Title.

QA76.73.P258R68 2015

005.75'6—dc23

2014045792

Copyright © 2015 Pearson Education, Inc.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, request forms and the appropriate contacts within the Pearson Education Global Rights & Permissions Department, please visit www.pearson.com/permissions/.

ISBN-13: 978-0-13-379678-0

ISBN-10: 0-13-379678-7

*To my parents, Rosie and Sandy Rosenzweig,
for their love and support
—Benjamin Rosenzweig*

*To my family, for their excitement and encouragement
—Elena Rakhimov*

This page intentionally left blank



Contents

Preface	xvii
Acknowledgments	xxi
About the Authors	xxiii
Introduction to PL/SQL New Features in Oracle 12c	xxv
Invoker's Rights Functions Can Be Result-Cached	xxvi
More PL/SQL-Only Data Types Can Cross the PL/ SQL-to-SQL Interface Clause	xxvii
ACCESSIBLE BY Clause	xxvii
FETCH FIRST Clause	xxviii
Roles Can Be Granted to PL/SQL Packages and Stand-Alone Subprograms	xxix
More Data Types Have the Same Maximum Size in SQL and PL/SQL	xxx
Database Triggers on Pluggable Databases	xxx
LIBRARY Can Be Defined as a DIRECTORY Object and with a CREDENTIAL Clause	xxx
Implicit Statement Results	xxx
BEQUEATH CURRENT_USER Views	xxxii

	INHERIT PRIVILEGES and INHERIT ANY PRIVILEGES Privileges	xxxii
	Invisible Columns	xxxiii
	Objects, Not Types, Are Editioned or Noneditioned	xxxiv
	PL/SQL Functions That Run Faster in SQL	xxxiv
	Predefined Inquiry Directives \$\$PLSQL_UNIT_OWNER and \$\$PLSQL_UNIT_TYPE	xxxvi
	Compilation Parameter PLSQL_DEBUG Is Deprecated	xxxvii
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	8
	Lab 1.2: PL/SQL Development Environment	9
	Getting Started with SQL Developer	10
	Getting Started with SQL*Plus	11
	Executing PL/SQL Scripts	14
	Lab 1.3: PL/SQL: The Basics	18
	DBMS_OUTPUT.PUT_LINE Statement	18
	Substitution Variable Feature	19
	Summary	25
Chapter 2	PL/SQL Language Fundamentals	27
	Lab 2.1: PL/SQL Programming Fundamentals	28
	PL/SQL Language Components	28
	PL/SQL Variables	29
	PL/SQL Reserved Words	32
	Identifiers in PL/SQL	33
	Anchored Data Types	34
	Declare and Initialize Variables	36
	Scope of a Block, Nested Blocks, and Labels	39
	Summary	41

Chapter 3	SQL in PL/SQL	43
	Lab 3.1: DML Statements in PL/SQL	44
	Initialize Variables with <code>SELECT INTO</code>	44
	Using the <code>SELECT INTO</code> Syntax for Variable Initialization	45
	Using DML in a PL/SQL Block	47
	Using a Sequence in a PL/SQL Block	48
	Lab 3.2: Transaction Control in PL/SQL	49
	Using <code>COMMIT</code> , <code>ROLLBACK</code> , and <code>SAVEPOINT</code>	49
	Putting Together DML and Transaction Control	53
	Summary	55
Chapter 4	Conditional Control: <code>IF</code> Statements	57
	Lab 4.1: <code>IF</code> Statements	58
	<code>IF-THEN</code> Statements	58
	<code>IF-THEN-ELSE</code> Statement	60
	Lab 4.2: <code>ELSIF</code> Statements	63
	Lab 4.3: Nested <code>IF</code> Statements	67
	Summary	70
Chapter 5	Conditional Control: <code>CASE</code> Statements	71
	Lab 5.1: <code>CASE</code> Statements	71
	<code>CASE</code> Statements	72
	Searched <code>CASE</code> Statements	74
	Lab 5.2: <code>CASE</code> Expressions	80
	Lab 5.3: <code>NULLIF</code> and <code>COALESCE</code> Functions	84
	<code>NULLIF</code> Function	84
	<code>COALESCE</code> Function	87
	Summary	89
Chapter 6	Iterative Control: Part I	91
	Lab 6.1: Simple Loops	92
	<code>EXIT</code> Statement	93
	<code>EXIT WHEN</code> Statement	97

Lab 6.2: WHILE Loops	98
Using WHILE Loops	98
Premature Termination of the WHILE Loop	101
Lab 6.3: Numeric FOR Loops	104
Using the IN Option in the Loop	105
Using the REVERSE Option in the Loop	107
Premature Termination of the Numeric FOR Loop	108
Summary	109
Chapter 7 Iterative Control: Part II	111
Lab 7.1: CONTINUE Statement	111
Using CONTINUE Statement	112
CONTINUE WHEN Statement	115
Lab 7.2: Nested Loops	118
Using Nested Loops	118
Using Loop Labels	120
Summary	122
Chapter 8 Error Handling and Built-in Exceptions	123
Lab 8.1: Handling Errors	124
Lab 8.2: Built-in Exceptions	126
Summary	132
Chapter 9 Exceptions	133
Lab 9.1: Exception Scope	133
Lab 9.2: User-Defined Exceptions	137
Lab 9.3: Exception Propagation	141
Re-raising Exceptions	146
Summary	147
Chapter 10 Exceptions: Advanced Concepts	149
Lab 10.1: RAISE_APPLICATION_ERROR	149
Lab 10.2: EXCEPTION_INIT Pragma	153
Lab 10.3: SQLCODE and SQLERRM	155
Summary	158

Chapter 11	Introduction to Cursors	159
Lab 11.1:	Types of Cursors	159
Making Use of an Implicit Cursor		160
Making Use of an Explicit Cursor		161
Lab 11.2:	Cursor Loop	165
Processing an Explicit Cursor		165
Making Use of a User-Defined Record		168
Making Use of Cursor Attributes		170
Lab 11.3:	Cursor FOR LOOPS	175
Making Use of Cursor FOR LOOPS		175
Lab 11.4:	Nested Cursors	177
Processing Nested Cursors		177
Summary		181
Chapter 12	Advanced Cursors	183
Lab 12.1:	Parameterized Cursors	183
Cursors with Parameters		184
Lab 12.2:	Complex Nested Cursors	185
Lab 12.3:	FOR UPDATE and WHERE CURRENT Cursors	187
FOR UPDATE Cursor		187
FOR UPDATE OF in a Cursor		189
WHERE CURRENT OF in a Cursor		189
Summary		190
Chapter 13	Triggers	191
Lab 13.1:	What Triggers Are	191
Database Trigger		192
BEFORE Triggers		195
AFTER Triggers		201
Autonomous Transaction		203
Lab 13.2:	Types of Triggers	205
Row and Statement Triggers		205
INSTEAD OF Triggers		206
Summary		211

Chapter 14	Mutating Tables and Compound Triggers	213
Lab 14.1:	Mutating Tables	213
What Is a Mutating Table?		214
Resolving Mutating Table Issues		215
Lab 14.2:	Compound Triggers	217
What Is a Compound Trigger?		218
Resolving Mutating Table Issues with Compound Triggers		220
Summary		223
Chapter 15	Collections	225
Lab 15.1:	PL/SQL Tables	226
Associative Arrays		226
Nested Tables		229
Collection Methods		232
Lab 15.2:	Varrays	235
Lab 15.3:	Multilevel Collections	240
Summary		242
Chapter 16	Records	243
Lab 16.1:	Record Types	243
Table-Based and Cursor-Based Records		244
User-Defined Records		246
Record Compatibility		248
Lab 16.2:	Nested Records	250
Lab 16.3:	Collections of Records	253
Summary		257
Chapter 17	Native Dynamic SQL	259
Lab 17.1:	EXECUTE IMMEDIATE Statements	260
Using the EXECUTE IMMEDIATE Statement		261
How to Avoid Common ORA Errors When Using EXECUTE IMMEDIATE		262
Lab 17.2:	OPEN-FOR, FETCH, and CLOSE Statements	271
Opening Cursor		272

Fetching from a Cursor	272
Closing a Cursor	273
Summary	280
Chapter 18 Bulk SQL	281
Lab 18.1: FORALL Statements	282
Using FORALL Statements	282
SAVE EXCEPTIONS Option	285
INDICES OF Option	288
VALUES OF Option	289
Lab 18.2: The BULK COLLECT Clause	291
Lab 18.3: Binding Collections in SQL Statements	299
Binding Collections with EXECUTE IMMEDIATE Statements	299
Binding Collections with OPEN-FOR, FETCH, and CLOSE Statements	306
Summary	309
Chapter 19 Procedures	311
Benefits of Modular Code	312
Block Structure	312
Anonymous Blocks	312
Lab 19.1: Creating Procedures	312
Putting Procedure Creation Syntax into Practice	313
Querying the Data Dictionary for Information on Procedures	314
Lab 19.2: Passing Parameters IN and OUT of Procedures	315
Using IN and OUT Parameters with Procedures	316
Summary	319
Chapter 20 Functions	321
Lab 20.1: Creating Functions	321
Creating Stored Functions	322
Making Use of Functions	325

Lab 20.2: Using Functions in SQL Statements	327
Invoking Functions in SQL Statements	327
Writing Complex Functions	328
Lab 20.3: Optimizing Function Execution in SQL	329
Defining a Function Using the WITH Clause	329
Creating a Function with the UDF Pragma	330
Summary	331
Chapter 21 Packages	333
Lab 21.1: Creating Packages	334
Creating Package Specifications	335
Creating Package Bodies	337
Calling Stored Packages	339
Creating Private Objects	341
Lab 21.2: Cursor Variables	344
Lab 21.3: Extending the Package	353
Extending the Package with Additional Procedures	353
Lab 21.4: Package Instantiation and Initialization	366
Creating Package Variables During Initialization	367
Lab 21.5: <code>SERIALLY_REUSABLE</code> Packages	368
Using the <code>SERIALLY_REUSABLE</code> Pragma	368
Summary	371
Chapter 22 Stored Code	373
Lab 22.1: Gathering Information about Stored Code	373
Getting Stored Code Information from the Data Dictionary	374
Overloading Modules	378
Summary	382
Chapter 23 Object Types in Oracle	385
Lab 23.1: Object Types	386
Creating Object Types	386
Using Object Types with Collections	391

Lab 23.2: Object Type Methods	394
Constructor Methods	395
Member Methods	398
Static Methods	398
Comparing Objects	399
Summary	404
Chapter 24 Oracle-Supplied Packages	405
Lab 24.1: Extending Functionality with Oracle-Supplied Packages	406
Accessing Files within PL/SQL with UTL_FILE	406
Scheduling Jobs with DBMS_JOB	410
Generating an Explain Plan with DBMS_XPLAN	414
Generating Implicit Statement Results with DBMS_SQL	417
Lab 24.2: Error Reporting with Oracle-Supplied Packages	419
Using the DBMS_UTILITY Package for Error Reporting	419
Using the UTL_CALL_STACK Package for Error Reporting	424
Summary	429
Chapter 25 Optimizing PL/SQL	431
Lab 25.1: PL/SQL Tuning Tools	432
PL/SQL Profiler API	432
Trace API	433
PL/SQL Hierarchical Profiler	436
Lab 25.2: PL/SQL Optimization Levels	438
Lab 25.3: Subprogram Inlining	444
Summary	453
Appendix A PL/SQL Formatting Guide	455
Case	455
White Space	455
Naming Conventions	456
Comments	457
Other Suggestions	457

Appendix B	Student Database Schema	461
	Table and Column Descriptions	461
Index		469



Preface

Oracle® PL/SQL by Example, Fifth 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 cursors, loops, 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. The

additional exercises available through the companion website can be used as labs and homework assignments to accompany the lectures in such a PL/SQL course.

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 such 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.

At the end of each chapter you will find a Summary section, which provides a brief conclusion of the material discussed in the chapter. In addition, the By the Way portion will state whether a particular chapter has an Additional Exercises section available on the companion website.

About the Companion Website

The companion Website is located at informit.com/title/0133796787. Here you will find three 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, which have two parts:
 - A Questions and Answers part where you are asked about the material presented in a particular chapter along with suggested answers to these questions. Oftentimes, you are asked to modify a script based on some requirements and explain the difference in the output caused by these modifications. Note that this part is also organized into Labs similar to its corresponding chapter in the book.
 - A Try it Yourself part where you are asked to create scripts based on the requirements provided. This part is different from the Questions and Answers part in that there are no scripts supplied with the questions. Instead, you will need to create scripts in their entirety.

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 12c only. However, you will be able to run a great majority of the examples and complete Additional Exercises and Try it Yourself sections by using the following products:

- Oracle 11g or higher
- SQL Developer or SQL*Plus 11g 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 11g or Oracle 12c in order to perform all or a majority of the examples in this book. When a feature will only work in the latest version of Oracle database, the book will state so explicitly. Additionally, you should have access to and be familiar with SQL Developer or SQL*Plus.

You have a number of options for how to edit and run scripts in SQL Developer or from 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, since these 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, a great majority 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. There are ten tables in the system that 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 particular courses, and the sections in which students have 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 equally important because it keeps track of which students have enrolled in which 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 a number of 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.



Acknowledgments

Ben Rosenzweig: I would like to thank my coauthor Elena Rakhimov for being a wonderful and knowledgeable colleague to work with. I would also like to thank Douglas Scherer for giving me the opportunity to work on this book as well as for providing constant support and assistance through the entire writing process. I am indebted to the team at Prentice Hall, which includes Greg Doench, Michelle Housley, and especially Songlin Qiu for her detailed edits. Finally, I would like to thank the many friends and family, especially Edward Clarin and Edward Knopping, for helping me through the long process of putting the whole book together, which included many late nights and weekends.

Elena Rakhimov: My contribution to this book reflects the help and advice of many people. I am particularly indebted to my coauthor Ben Rosenzweig for making this project a rewarding and enjoyable experience. Many thanks to Greg Doench, Michelle Housley, and especially Songlin Qiu for her meticulous editing skills, and many others at Prentice Hall who diligently worked to bring this book to market. Thanks to Michael Rinomhota for his invaluable expertise in setting up the Oracle environment and Dan Hotka for his valuable comments and suggestions. 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.

This page intentionally left blank



About the Authors

Benjamin Rosenzweig is a Senior Project Manager at Misys Financial Software, where he has worked since 2002. Prior to that he was a principal consultant for more than three years at Oracle Corporation in the Custom Development Department. His computer experience ranges from creating an electronic Tibetan–English Dictionary in Kathmandu, Nepal, to supporting presentation centers at Goldman Sachs and managing a trading system at TIAA-CREF. Benjamin has been an instructor at the Columbia University Computer Technology and Application program in New York City since 1998. In 2002 he was awarded the “Outstanding Teaching Award” from the Chair and Director of the CTA program. He holds a B.A. from Reed College and a certificate in database development and design from Columbia University. His previous books with Prentice Hall are *Oracle Forms Developer: The Complete Video Course* (2000), and *Oracle Web Application Programming for PL/SQL Developers* (2003).

Elena Rakhimov has over 20 years of experience in database architecture and development in a wide spectrum of enterprise and business environments ranging from non-profit organizations to Wall Street to her current position with a prominent software company where she heads up the database team. 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. She currently resides in Vancouver, Canada.

This page intentionally left blank



Introduction to PL/SQL

New Features in Oracle 12c

Oracle 12c has introduced a number of 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’s online help.

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

- Invoker’s rights functions can be result-cached
- More PL/SQL-only data types can cross the PL/SQL-to-SQL interface clause
- `ACCESSIBLE BY` clause
- `FETCH FIRST` clause
- Roles can be granted to PL/SQL packages and stand-alone subprograms
- More data types have the same maximum size in SQL and PL/SQL
- Database triggers on pluggable databases
- `LIBRARY` can be defined as `DIRECTORY` object and with `CREDENTIAL` clause
- Implicit statement results
- `BEQUEATH CURRENT_USER` views
- `INHERIT PRIVILEGES` and `INHERIT ANY PRIVILEGES` privileges
- Invisible columns
- Objects, not types, are editioned or noneditioned

- PL/SQL functions that run faster in SQL
- Predefined inquiry directives `$$PLSQL_UNIT_OWNER` and `$$PLSQL_UNIT_TYPE`
- Compilation parameter `PLSQL_DEBUG` is deprecated

Invoker's Rights Functions Can Be Result-Cached

When a stored subprogram is created in Oracle products, it may be created as either a *definer rights* (DR) unit or an *invoker rights* (IR) unit. A DR unit would execute with the permissions of its owner, whereas an IR unit would execute with the permissions of a user who invoked that particular unit. By default, a stored subprogram is created as a DR unit unless explicitly specified otherwise. Whether a particular unit is considered a DR or IR unit is controlled by the `AUTHID` property, which may be set to either `DEFINER` (default) or `CURRENT_USER`.

Prior to Oracle 12c, functions created with the invoker rights clause (`AUTHID CURRENT_USER`) could not be result-cached. To create a function as an IR unit, the `AUTHID` clause must be added to the function specification.

A result-cached function is a function whose parameter values and result are stored in the cache. As a consequence, when such a function is invoked with the same parameter values, its result is retrieved from the cache instead of being computed again. To enable a function for result-caching, the `RESULT_CACHE` clause must be added to the function specification. This is demonstrated by the following example (the invoker rights clause and result-caching are highlighted in bold).

For Example *Result-Caching Functions Created with Invoker's Rights*

```
CREATE OR REPLACE FUNCTION get_student_rec (p_student_id IN NUMBER)
RETURN STUDENT%ROWTYPE
AUTHID CURRENT_USER
RESULT_CACHE RELIES_ON (student)
IS
    v_student_rec STUDENT%ROWTYPE;
BEGIN
    SELECT *
        INTO v_student_rec
        FROM student
        WHERE student_id = p_student_id;

    RETURN v_student_rec;
EXCEPTION
    WHEN no_data_found
    THEN
        RETURN NULL;
END get_student_rec;
/

-- Execute newly created function
DECLARE
    v_student_rec STUDENT%ROWTYPE;
```

```
BEGIN
  v_student_rec := get_student_rec (p_student_id => 230);
END;
```

Note that if the student record for student ID 230 is in the result cache already, then the function will return the student record from the result cache. In the opposite case, the student record will be selected from the `STUDENT` table and added to the cache for future use. Because the result cache of the function relies on the `STUDENT` table, any changes applied and committed on the `STUDENT` table will invalidate all cached results for the `get_student_rec` function.

More PL/SQL-Only Data Types Can Cross the PL/SQL-to-SQL Interface Clause

In this release, Oracle has extended support of PL/SQL-only data types to dynamic SQL and client programs (OCI or JDBC). For example, you can bind collections variables when using the `EXECUTE IMMEDIATE` statement or the `OPEN FOR`, `FETCH`, and `CLOSE` statements. This topic is covered in greater detail in Lab 18.3, *Binding Collections in SQL Statements*, in Chapter 18.

ACCESSIBLE BY Clause

An optional `ACCESSIBLE BY` clause enables you to specify a list of PL/SQL units that may access the PL/SQL unit being created or modified. The `ACCESSIBLE BY` clause is typically added to the module header—for example, to the function or procedure header. Each unit listed in the `ACCESSIBLE BY` clause is called an *accessor*, and the clause itself is also called a *white list*. This is demonstrated in the following example (the `ACCESSIBLE BY` clause is shown in bold).

For Example *Procedure Created with the ACCESSIBLE BY Clause*

```
CREATE OR REPLACE PROCEDURE test_proc1
ACCESSIBLE BY (TEST_PROC2)
AS
BEGIN
  DBMS_OUTPUT.PUT_LINE ('TEST_PROC1');
END test_proc1;
/

CREATE OR REPLACE PROCEDURE test_proc2
AS
BEGIN
  DBMS_OUTPUT.PUT_LINE ('TEST_PROC2');
  test_proc1;
END test_proc2;
/
```

```
-- Execute TEST_PROC2
BEGIN
    test_proc2;
END;
/

TEST_PROC2
TEST_PROC1

-- Execute TEST_PROC1 directly
BEGIN
    test_procl;
END;
/

ORA-06550: line 2, column 4:
PLS-00904: insufficient privilege to access object TEST_PROC1
ORA-06550: line 2, column 4:
PL/SQL: Statement ignored
```

In this example, there are two procedures, `test_procl` and `test_proc2`, and `test_procl` is created with the `ACCESSIBLE BY` clause. As a consequence, `test_procl` may be accessed by `test_proc2` only. This is demonstrated by two anonymous PL/SQL blocks. The first block executes `test_proc2` successfully. The second block attempts to execute `test_procl` directly and, as a result, causes an error.

Note that both procedures were created within a single schema (`STUDENT`), and that both PL/SQL blocks were executed in the single session by the schema owner (`STUDENT`).

FETCH FIRST Clause

The `FETCH FIRST` clause is a new optional feature that is typically used with the “Top-N” queries as illustrated by the following example. The `ENROLLMENT` table used in this example contains student registration data. Each student is identified by a unique student ID and may be registered for multiple courses. The `FETCH FIRST` clause is shown in bold.

For Example *Using `FETCH FIRST` Clause with “Top-N” Query*

```
-- Sample student IDs from the ENROLLMENT table
SELECT student_id
FROM enrollment;

STUDENT_ID
-----
102
102
103
104
105
```

```

106
106
107
108
109
109
110
110
...

-- "Top-N" query returns student IDs for the 5 students that registered for the most
-- courses
SELECT student_id, COUNT(*) courses
  FROM enrollment
 GROUP BY student_id
 ORDER BY courses desc
FETCH FIRST 5 ROWS ONLY;

```

STUDENT_ID	COURSES
214	4
124	4
232	3
215	3
184	3

Note that **FETCH FIRST** clause may also be used in conjunction with the **BULK COLLECT INTO** clause as demonstrated here. The **FETCH FIRST** clause is shown in bold.

For Example *Using FETCH FIRST Clause with BULK COLLECT INTO Clause*

```

DECLARE
  TYPE student_name_tab IS TABLE OF VARCHAR2(100) INDEX BY PLS_INTEGER;

  student_names student_name_tab;
BEGIN
  -- Fetching first 20 student names only
  SELECT first_name||' '||last_name
     BULK COLLECT INTO student_names
   FROM student
  FETCH FIRST 20 ROWS ONLY;

  DBMS_OUTPUT.PUT_LINE ('There are '||student_names.COUNT||' students');
END;
/
There are 20 students

```

Roles Can Be Granted to PL/SQL Packages and Stand-Alone Subprograms

Starting with Oracle 12c, you are able to grant roles to PL/SQL packages and stand-alone subprograms. Note that granting a role to a PL/SQL package or stand-alone subprogram does not alter its compilation. Instead, it affects how privileges required by the SQL statements that are issued by the PL/SQL unit at run time are checked.

Consider the following example where the READ role is granted to the function `get_student_name`.

For Example *Granting READ Role to the `get_student_name` Function*

```
GRANT READ TO FUNCTION get_student_name;
```

More Data Types Have the Same Maximum Size in SQL and PL/SQL

Prior to Oracle 12c, some data types had different maximum sizes in SQL and in PL/SQL. For example, in SQL the maximum size of NVARCHAR2 was 4000 bytes, whereas in PL/SQL it was 32,767 bytes. Starting with Oracle 12c, the maximum sizes of the VARCHAR2, NVARCHAR2, and RAW data types have been extended to 32,767 for both SQL and PL/SQL. To see these maximum sizes in SQL, the initialization parameter MAX_STRING_SIZE must be set to EXTENDED.

Database Triggers on Pluggable Databases

The pluggable database (PDB) is one of the components of Oracle's multitenant architecture. Typically it is a portable collection of schemas and other database objects. Starting with Oracle 12c, you are able to create event triggers on PDBs. Detailed information on triggers is provided in Chapters 13 and 14. Note that PDBs are outside the scope of this book, but detailed information on them may be found in Oracle's online Administration Guide.

LIBRARY Can Be Defined as a DIRECTORY Object and with a CREDENTIAL Clause

A LIBRARY is a schema object associated with a shared library of an operating system. It is created with the help of the CREATE OR REPLACE LIBRARY statement. A DIRECTORY is also an object that maps an alias to an actual directory on the server file system. The DIRECTORY object is covered very briefly in Chapter 25 as part of the install processes for the PL/SQL Profiler API and PL/SQL Hierarchical Profiler. In the Oracle 12c release, a LIBRARY object may be defined as a DIRECTORY object with an optional CREDENTIAL clause as shown here.

For Example *Creating `LIBRARY` as `DIRECTORY` Object*

```
CREATE OR REPLACE LIBRARY my_lib AS 'plsql_code' IN my_dir;
```

In this example, the `LIBRARY` object `my_lib` is created as a `DIRECTORY` object. The `'plsql_code'` is the name of the dynamic link library (DDL) in the `DIRECTORY` object `my_dir`. Note that for this library to be created successfully, the `DIRECTORY` object `my_dir` must be created beforehand. More information on `LIBRARY` and `DIRECTORY` objects can be found in Oracle's online Database PL/SQL Language Reference.

Implicit Statement Results

Prior to Oracle release 12c, result sets of SQL queries were returned explicitly from the stored PL/SQL subprograms via `REF CURSOR` out parameters. As a result, the invoker program had to bind to the `REF CURSOR` parameters and fetch the result sets explicitly as well.

Starting with this release, the `REF CURSOR` out parameters can be replaced by two procedures of the `DBMS_SQL` package, `RETURN_RESULT` and `GET_NEXT_RESULT`. These procedures enable stored PL/SQL subprograms to return result sets of SQL queries implicitly, as illustrated in the following example (the reference to the `RETURN_RESULT` procedure is highlighted in bold):

For Example *Using `DBMS_SQL.RETURN_RESULT` Procedure*

```
CREATE OR REPLACE PROCEDURE test_return_result
AS
    v_cur SYS_REFCURSOR;
BEGIN
    OPEN v_cur
    FOR
        SELECT first_name, last_name
        FROM instructor
        FETCH FIRST ROW ONLY;

    DBMS_SQL.RETURN_RESULT (v_cur);
END test_return_result;
/

BEGIN
    test_return_result;
END;
/
```

In this example, the `test_return_result` procedure returns the instructor's first and last names to the client application implicitly. Note that the cursor `SELECT` statement employs a `FETCH FIRST ROW ONLY` clause, which was introduced in Oracle 12c as well. To get the result set from the procedure `test_return_result` successfully, the client application must likewise be upgraded to Oracle 12c. Otherwise, the following error message is returned:

```
ORA-29481: Implicit results cannot be returned to client.  
ORA-06512: at "SYS.DBMS_SQL", line 2785  
ORA-06512: at "SYS.DBMS_SQL", line 2779  
ORA-06512: at "STUDENT.TEST_RETURN_RESULT", line 10  
ORA-06512: at line 2
```

BEQUEATH CURRENT_USER Views

Prior to Oracle 12c, a view could be created only as a definer rights unit. Starting with release 12c, a view may be created as an invoker's rights unit as well (this is similar to the `AUTHID` property of a stored subprogram). For views, however, this behavior is achieved by specifying a `BEQUEATH DEFINER` (default) or `BEQUEATH CURRENT_USER` clause at the time of its creation as illustrated by the following example (the `BEQUEATH CURRENT_USER` clause is shown in bold):

For Example *Creating View with `BEQUEATH CURRENT_USER` Clause*

```
CREATE OR REPLACE VIEW my_view  
BEQUEATH CURRENT_USER  
AS  
  SELECT table_name, status, partitioned  
     FROM user_tables;
```

In this example, `my_view` is created as an IR unit. Note that adding this property to the view does not affect its primary usage. Rather, similarly to the `AUTHID` property, it determines which set of permissions will be applied at the time when the data is selected from this view.

INHERIT PRIVILEGES and INHERIT ANY PRIVILEGES Privileges

Starting with Oracle 12c, an invoker's rights unit will execute with the invoker's permissions only if the owner of the unit has `INHERIT PRIVILEGES` or `INHERIT ANY PRIVILEGES` privileges. For example, before Oracle 12c, suppose `user1` created a function `F1` as an invoker's rights unit and granted execute privilege on it to `user2`, who happened to have more privileges than `user1`. Then when `user2` ran function

F1, the function would run with the permissions of user2, potentially performing operations for which user1 might not have had permissions. This is no longer the case with Oracle 12c. As stated previously, such behavior must be explicitly specified via `INHERIT PRIVILEGES` or `INHERIT ANY PRIVILEGES` privileges.

Invisible Columns

Starting with Oracle 12c, it is possible to define and manipulate invisible columns. In PL/SQL, records defined as `%ROWTYPE` are aware of such columns, as illustrated by the following example (references to the invisible columns are shown in bold):

For Example *%ROWTYPE Records and Invisible Columns*

```
-- Make NUMERIC_GRADE column invisible
ALTER TABLE grade MODIFY (numeric_grade INVISIBLE);
/
table GRADE altered

DECLARE
  v_grade_rec grade%ROWTYPE;
BEGIN
  SELECT *
    INTO v_grade_rec
    FROM grade
    FETCH FIRST ROW ONLY;

  DBMS_OUTPUT.PUT_LINE ('student ID: ' || v_grade_rec.student_id);
  DBMS_OUTPUT.PUT_LINE ('section ID: ' || v_grade_rec.section_id);
  -- Referencing invisible column causes an error
  DBMS_OUTPUT.PUT_LINE ('grade:      ' || v_grade_rec.numeric_grade);
END;
/
ORA-06550: line 12, column 54:
PLS-00302: component 'NUMERIC_GRADE' must be declared
ORA-06550: line 12, column 4:
PL/SQL: Statement ignored

-- Make NUMERIC_GRADE column visible
ALTER TABLE grade MODIFY (numeric_grade VISIBLE);
/
table GRADE altered

DECLARE
  v_grade_rec grade%ROWTYPE;
BEGIN
  SELECT *
    INTO v_grade_rec
    FROM grade
    FETCH FIRST ROW ONLY;

  DBMS_OUTPUT.PUT_LINE ('student ID: ' || v_grade_rec.student_id);
  DBMS_OUTPUT.PUT_LINE ('section ID: ' || v_grade_rec.section_id);
  -- This time the script executes successfully
  DBMS_OUTPUT.PUT_LINE ('grade:      ' || v_grade_rec.numeric_grade);
END;
/
```

```
student ID: 123
section ID: 87
grade:      99
```

As you can gather from this example, the first run of the anonymous PL/SQL block did not complete due to the reference to the invisible column. Once the `NUMERIC_GRADE` column has been set to visible again, the script is able to complete successfully.

Objects, Not Types, Are Editioned or Noneditioned

An edition is a component of the edition-based redefinition feature that allows you to make a copy of an object—for example, a PL/SQL package—and make changes to it without affecting or invalidating other objects that may be dependent on it. With introduction of this feature, objects created in the database may be defined as editioned or noneditioned. For an object to be editioned, its object type must be editionable and it must have the `EDITIONABLE` property. Similarly, for an object to be noneditioned, its object type must be noneditioned or it must have the `NONEDITIONABLE` property.

Starting with Oracle 12c, you are able to specify whether a schema object is editionable or noneditionable in the `CREATE OR REPLACE` and `ALTER` statements. In this new release, a user (schema) that has been enabled for editions is able to own a noneditioned object even if its type is editionable in the database but noneditionable in the schema itself or if this object has `NONEDITIONABLE` property.

PL/SQL Functions That Run Faster in SQL

Starting with Oracle 12c, you can create user-defined functions that may run faster when they are invoked in the SQL statements. This may be accomplished as follows:

- User-defined function declared in the `WITH` clause of a `SELECT` statement
- User-defined function created with the UDF pragma

Consider the following example, where the `format_name` function is created in the `WITH` clause of the `SELECT` statement. This newly created function returns the formatted student name.

For Example *Creating a User-Defined Function in the `WITH` Clause*

```
WITH
  FUNCTION format_name (p_salutation IN VARCHAR2
                        ,p_first_name IN VARCHAR2
                        ,p_last_name  IN VARCHAR2)
```

```

RETURN VARCHAR2
IS
BEGIN
  IF p_salutation IS NULL
  THEN
    RETURN p_first_name || ' ' || p_last_name;
  ELSE
    RETURN p_salutation || ' ' || p_first_name || ' ' || p_last_name;
  END IF;
END;

SELECT format_name (salutation, first_name, last_name) student_name
FROM student
FETCH FIRST 10 ROWS ONLY;

STUDENT_NAME
-----
Mr. George Kocka
Ms. Janet Jung
Ms. Kathleen Mulroy
Mr. Joel Brendler
Mr. Michael Carcia
Mr. Gerry Tripp
Mr. Rommel Frost
Mr. Roger Snow
Ms. Z.A. Scrittorale
Mr. Joseph Yourish

```

Next, consider another example where the `format_name` function is created with the UDF pragma.

For Example *Creating a User-Defined Function in the UDF Pragma*

```

CREATE OR REPLACE FUNCTION format_name (p_salutation IN VARCHAR2
                                         ,p_first_name IN VARCHAR2
                                         ,p_last_name IN VARCHAR2)

RETURN VARCHAR2
AS
  PRAGMA UDF;
BEGIN
  IF p_salutation IS NULL
  THEN
    RETURN p_first_name || ' ' || p_last_name;
  ELSE
    RETURN p_salutation || ' ' || p_first_name || ' ' || p_last_name;
  END IF;
END;

/

SELECT format_name (salutation, first_name, last_name) student_name
FROM student
FETCH FIRST 10 ROWS ONLY;

STUDENT_NAME
-----
Mr. George Kocka
Ms. Janet Jung
Ms. Kathleen Mulroy
Mr. Joel Brendler
Mr. Michael Carcia
Mr. Gerry Tripp
Mr. Rommel Frost
Mr. Roger Snow
Ms. Z.A. Scrittorale
Mr. Joseph Yourish

```

Predefined Inquiry Directives `$$PLSQL_UNIT_OWNER` and `$$PLSQL_UNIT_TYPE`

In PL/SQL, there are a number of predefined inquiry directives, as described in the following table (`$$PLSQL_UNIT_OWNER` and `$$PLSQL_UNIT_TYPE` are highlighted in bold):

Name	Description
<code>\$\$PLSQL_LINE</code>	The number of the code line where it appears in the PL/SQL subroutine.
<code>\$\$PLSQL_UNIT</code>	The name of the PL/SQL subroutine. For the anonymous PL/SQL blocks, it is set to NULL.
<code>\$\$PLSQL_UNIT_OWNER</code>	A new directive added in release 12c. This is the name of the owner (schema) of the PL/SQL subroutine. For anonymous PL/SQL blocks, it is set to NULL.
<code>\$\$PLSQL_UNIT_TYPE</code>	A new directive added in release 12c. This is the type of the PL/SQL subroutine—for example, FUNCTION, PROCEDURE, or PACKAGE BODY.
<code>\$\$plsql_compilation_parameter</code>	A set of PL/SQL compilation parameters, some of which are <code>PLSQL_CODE_TYPE</code> , which specifies the compilation mode for PL/SQL subroutines, and others of which are <code>PLSQL_OPTIMIZE_LEVEL</code> (covered in Chapter 25).

The following example demonstrates how directives may be used.

For Example *Using Predefined Inquiry Directives*

```
CREATE OR REPLACE PROCEDURE test_directives
AS
BEGIN
    DBMS_OUTPUT.PUT_LINE ('Procedure test_directives');
    DBMS_OUTPUT.PUT_LINE ('$$PLSQL_UNIT_OWNER: ' || $$PLSQL_UNIT_OWNER);
    DBMS_OUTPUT.PUT_LINE ('$$PLSQL_UNIT_TYPE: ' || $$PLSQL_UNIT_TYPE);
    DBMS_OUTPUT.PUT_LINE ('$$PLSQL_UNIT: ' || $$PLSQL_UNIT);
    DBMS_OUTPUT.PUT_LINE ('$$PLSQL_LINE: ' || $$PLSQL_LINE);
END;
/

BEGIN
    -- Execute TEST_DERECTIVES procedure
    test_directives;
    DBMS_OUTPUT.PUT_LINE ('Anonymous PL/SQL block');
    DBMS_OUTPUT.PUT_LINE ('$$PLSQL_UNIT_OWNER: ' || $$PLSQL_UNIT_OWNER);
    DBMS_OUTPUT.PUT_LINE ('$$PLSQL_UNIT_TYPE: ' || $$PLSQL_UNIT_TYPE);
```

```
DBMS_OUTPUT.PUT_LINE ('$$PLSQL_UNIT:      ' || $$PLSQL_UNIT);
DBMS_OUTPUT.PUT_LINE ('$$PLSQL_LINE:      ' || $$PLSQL_LINE);
END;
/

Procedure test_directives
$$PLSQL_UNIT_OWNER: STUDENT
$$PLSQL_UNIT_TYPE:  PROCEDURE
$$PLSQL_UNIT:       TEST_DIRECTIVES
$$PLSQL_LINE:       8
Anonymous PL/SQL block
$$PLSQL_UNIT_OWNER:
$$PLSQL_UNIT_TYPE:  ANONYMOUS BLOCK
$$PLSQL_UNIT:
$$PLSQL_LINE:       8
```

Compilation Parameter PLSQL_DEBUG Is Deprecated

Starting with Oracle release 12c, the PLSQL_DEBUG parameter is deprecated. To compile PL/SQL subroutines for debugging, the PLSQL_OPTIMIZE_LEVEL parameter should be set to 1. Chapter 25 covers the PLSQL_OPTIMIZE_LEVEL parameter and various optimization levels supported by the PL/SQL performance optimizer in greater detail.

This page intentionally left blank

PL/SQL Concepts

In this chapter, you will learn about

- | | |
|----------------------------------|---------|
| ■ PL/SQL Architecture | Page 2 |
| ■ PL/SQL Development Environment | Page 9 |
| ■ PL/SQL: The Basics | Page 18 |

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 Chapters 13 and 14) 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 *N*-tier architecture, where each tier represents a separate logical process. 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 three-tier architecture is shown in Figure 1.1.

PL/SQL Architecture

While 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. 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 very 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.

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 PL/SQL block contains no SQL statements, the entire block is executed on the client side.

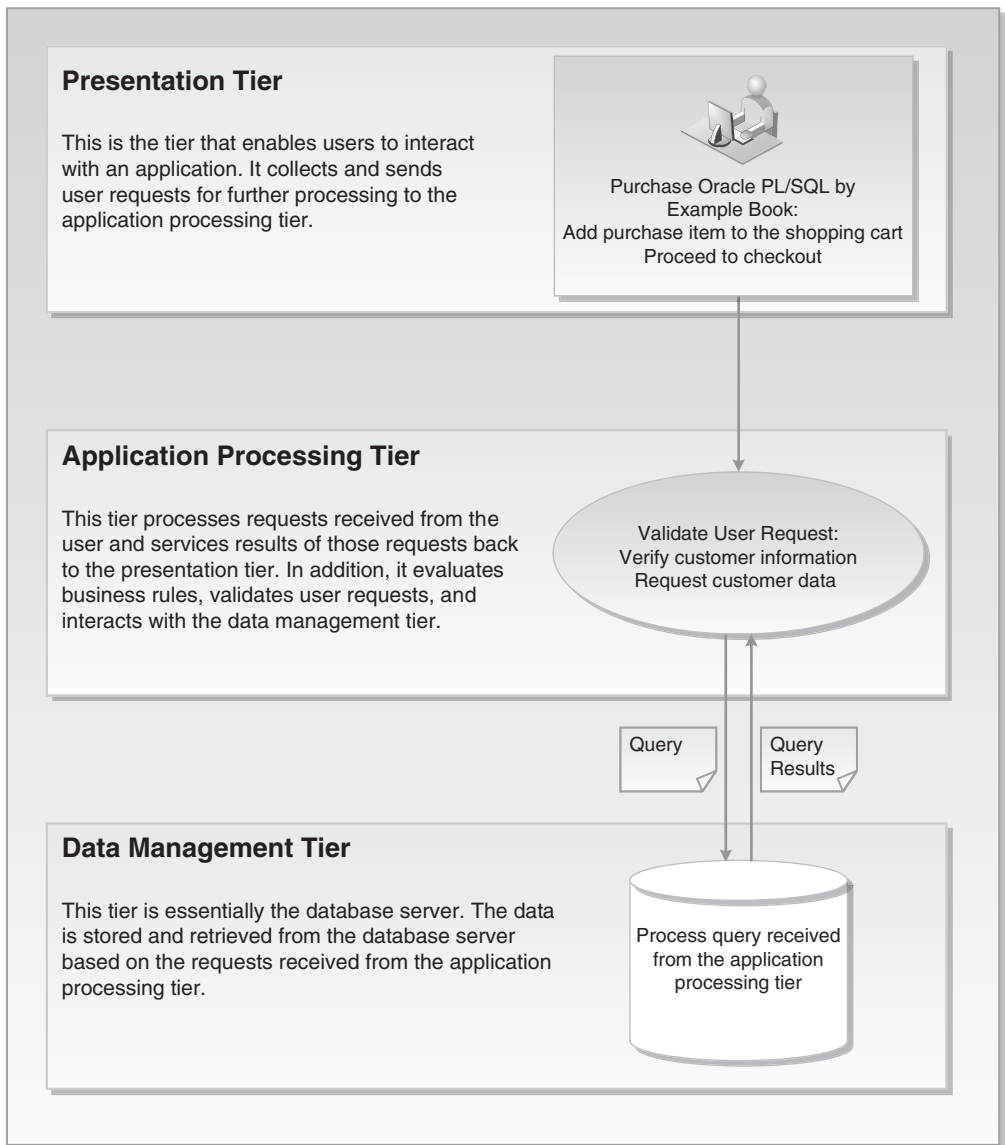


Figure 1.1 Three-Tier Architecture

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.

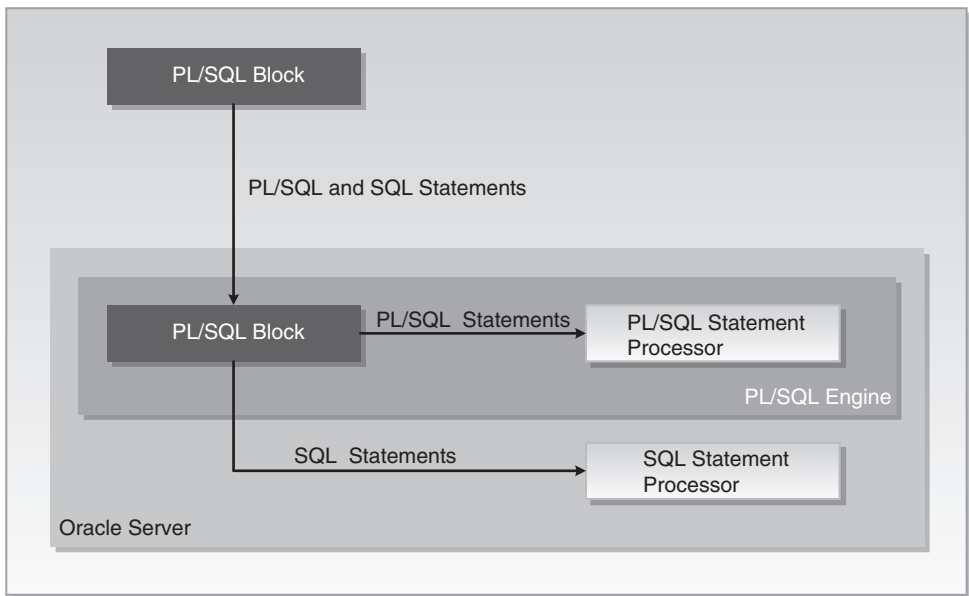


Figure 1.2 The PL/SQL Engine and Oracle Server

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 of the results returned to the client in the same round trip. This is obviously a more efficient process 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.

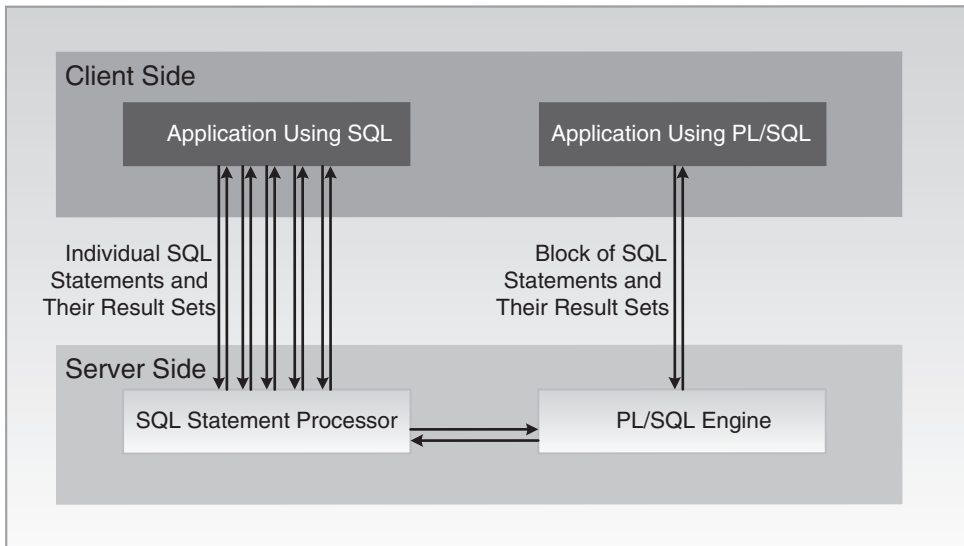


Figure 1.3 PL/SQL in Client–Server Architecture

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 a PL/SQL script.

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*

```
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

```
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

```
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 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 will be 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

```
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.

Once 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

```
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 is passed to the exception-handling section and the `DBMS_OUTPUT.PUT_LINE` statement is executed. Chapters 8, 9, and 10 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*

```
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;
```

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. Once 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 run time. 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 run time, 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. Once 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 a fairly recent addition to the Oracle

tool set and 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 run time, the SQL and PL/SQL statements are sent to the database. Once 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

If SQL Developer has been installed as part of the Oracle installation, you can launch it by accessing Start, All Programs, Oracle, Application Development, SQL Developer on Windows 7 and earlier versions. On Windows 8, SQL Developer is invoked by accessing Start, All Apps, Oracle, SQL Developer. Alternatively, you can download and install this tool as a separate module.

Once SQL Developer is installed, you need to create connection to the database server. This can be accomplished by clicking on the Plus icon located in the upper-left corner of the Connections tab. This 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 (StudentConnection), user name (student), and password (learn).

By the Way

Starting with Oracle 11g, the password is case sensitive.

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 is usually set to orcl.

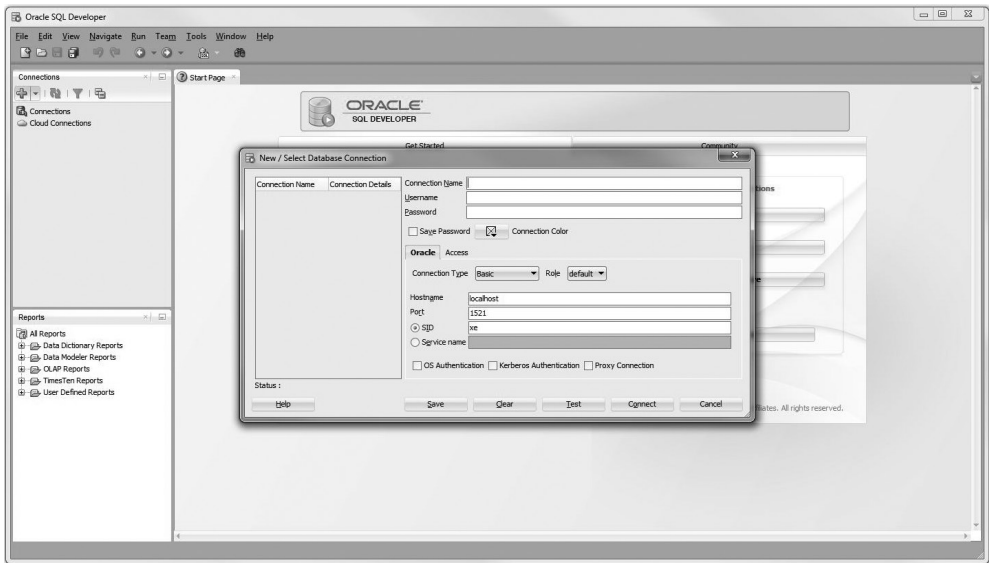


Figure 1.4 Creating a Database Connection in SQL Developer

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.

Once the connection has been successfully created, you can connect to the database by double-clicking on the `StudentConnection`. By expanding the `StudentConnection` (clicking on the plus sign located to the left of it), you are able to 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, shown in Figure 1.5.

To disconnect from the `STUDENT` schema, you need to right-click on the `StudentConnection` and click on the `Disconnect` option. This is illustrated in Figure 1.6.

Getting Started with SQL*Plus

On Windows 7 and earlier versions, you can access SQL*Plus by choosing `Start, All Programs, Oracle, Application Development, SQL*Plus` under the `Start` button. On Windows 8, SQL*Plus is invoked by accessing `Start, All Apps, Oracle, SQL*Plus`.

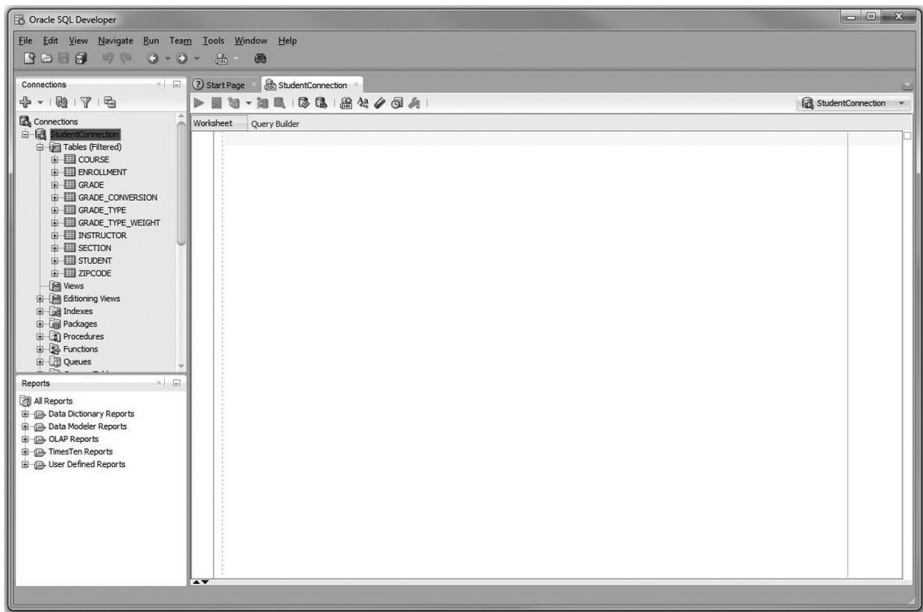


Figure 1.5 List of Tables in the STUDENT Schema

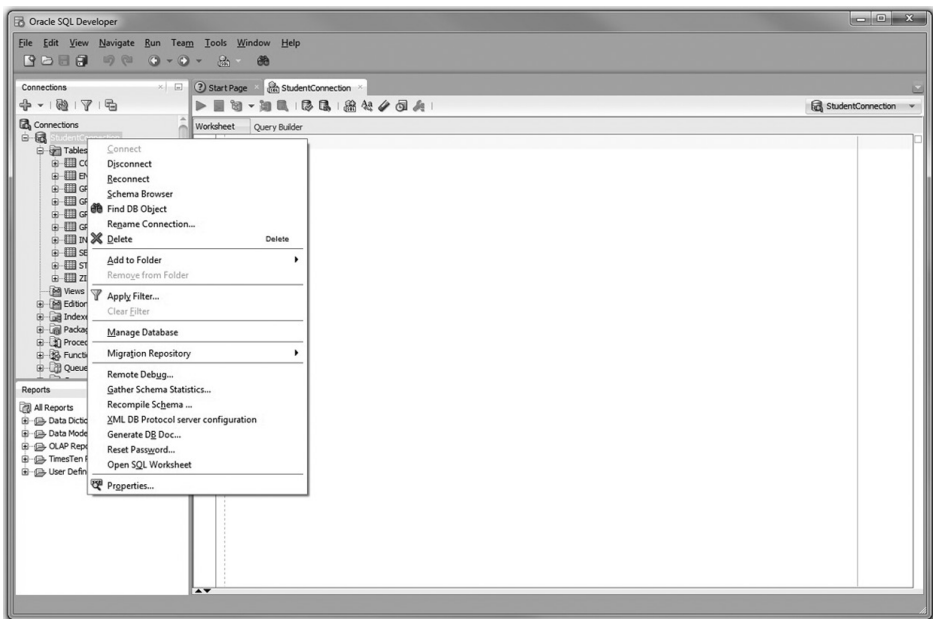


Figure 1.6 Disconnecting from a Database in SQL Developer

When you open SQL*Plus, you are prompted to enter your user name and password (“student” and “learn,” respectively). In addition, you can invoke SQL*Plus by typing `sqlplus` in the command prompt window.

By the Way

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

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

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.

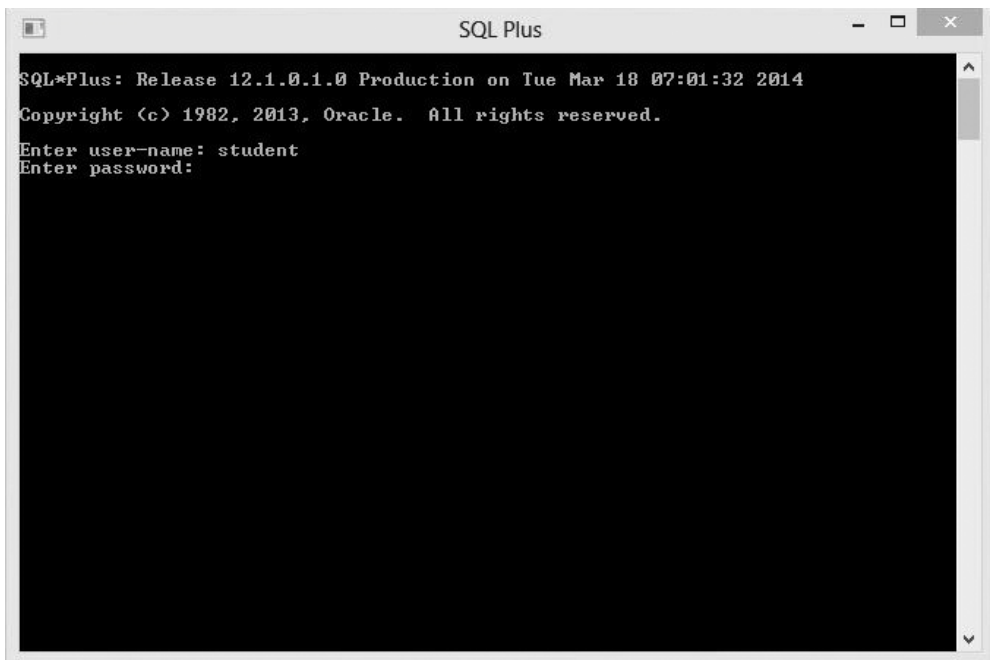


Figure 1.7 Connecting to the Database in SQL*Plus

Executing PL/SQL Scripts

As mentioned earlier, at run time SQL and PL/SQL statements are sent from the client machine to the database. Once 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

```
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 on the triangle button in the Student-Connection 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.

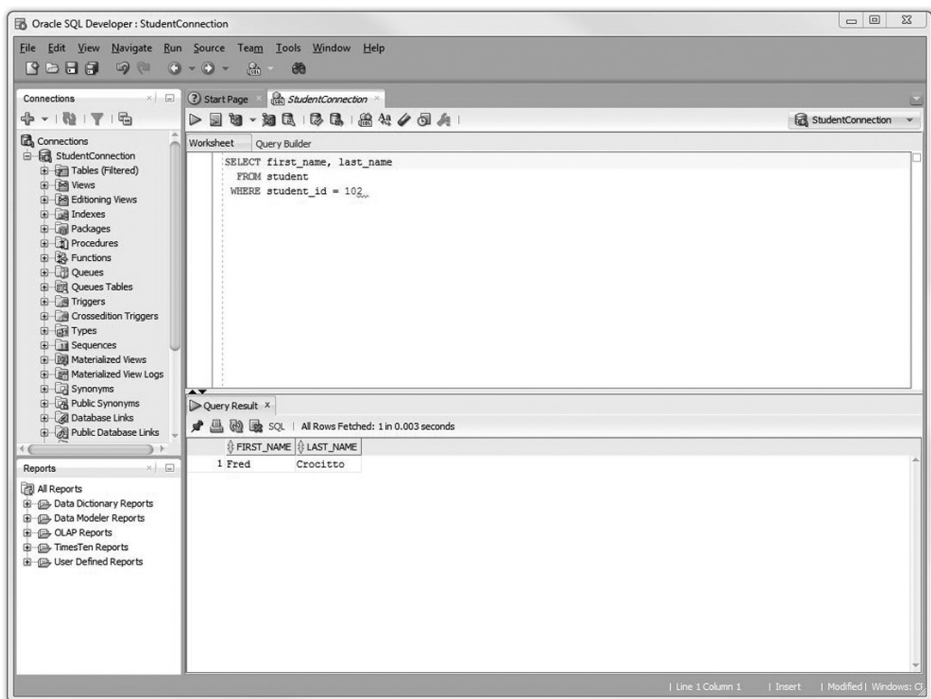


Figure 1.8 Executing a Query in SQL Developer