Ben Forta

Sams Teach Yourself Oracle® PL/SQL in 10 Minutes

FREE SAMPLE CHAPTER

SHARE WITH OTHERS

Ben Forta

Sams Teach Yourself Oracle® PL/SQL in 10 Minutes

SAMS | 800 East 96th Street, Indianapolis, Indiana 46240

Sams Teach Yourself Oracle® PL/SQL in 10 Minutes

ISBN-13: 978-0-672-32866-4
ISBN-10: 0-672-32866-6

Library of Congress Control Number: 2015910491

Printed in the United States of America

First Printing September 2015

Acquisitions Editor
Mark Taber

Managing Editor
Kristy Hart

Senior Project Editor
Betsy Gratner

Copy Editor
Paula Lowell

Indexer
Lisa Stumpf

Proofreader
Sarah Kearns

Editorial Assistant
Vanessa Evans

Cover Designer
Mark Shirar

Senior Compositor
Gloria Schurick

Contents

About the Author

Ben Forta has three decades of experience in the computer industry in product design and development, support, training, and marketing. As Adobe Inc.'s Senior Director of Education Initiatives, he spends a considerable amount of time teaching, talking, and writing about Adobe products, coding and application development, creativity, and digital literacy and provides feedback to help shape the future direction of Adobe products.

Ben is the author of more than 40 books, including the world's best-selling title on SQL, as well as titles on topics as diverse as Regular Expressions, mobile development, and Adobe ColdFusion. More than 750,000 copies of his books are in print in English, and titles have been translated into fifteen languages. Many of these titles are used as textbooks in colleges and universities worldwide.

Education is Ben's passion. Between writing, lecturing, and in-classroom experience, Ben has dedicated his professional and personal lives to teaching, inspiring, and sharing his love for technology and creativity. He is immensely grateful to have had the opportunity to share with millions worldwide.

Ben is also a successful entrepreneur with experience creating, building, and selling start-ups. He is a sought-after public speaker, a writer, and a blogger, and he presents on education and development topics worldwide.

Acknowledgments

It's been sixteen years since the publication of my first book on SQL, *Sams Teach Yourself SQL in 10 Minutes*. That book was met with such positive feedback that it has been updated three times, has spawned four spin-off titles (the most recent being the book you are reading right now), and has been translated more than a dozen times. In all of its various flavors and iterations, this little book has helped hundreds of thousands learn the basics of SQL. So, first and foremost, thanks to all of you who have trusted me and this book over the years; your support is both incredibly humbling and a source of great pride.

I am blessed with some very vocal and opinionated readers who regularly share ideas, comments, suggestions, and occasionally criticism. These books continue to improve directly in response to that feedback, so thanks, and please keep it coming.

Thanks to the numerous schools and colleges the world over who have made this series part of their curriculum. Seeing students use my writing as part of their studies never ceases to thrill.

And finally, thanks to my partners at Pearson with whom I've now published more than 40 titles, and without whose support none would have seen the light of day. In particular, thanks to Betsy Gratner for shepherding this book through the process, Paula Lowell for her editing help, and Mark Taber for once again patiently and encouragingly supporting whatever I toss his way.

Ben Forta

We Want to Hear from You!

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

We welcome your comments. You can email or write to let us know what you did or didn't like about this book—as well as what we can do to make our books better.

Please note that we cannot help you with technical problems related to the topic of this book.

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

Email: feedback@samspublishing.com

Mail: Sams Publishing
ATTN: Reader Feedback
800 East 96th Street
Indianapolis, IN 46240 USA

Reader Services

Visit our website and register this book at informit.com/register for convenient access to any updates, downloads, or errata that might be available for this book.

Introduction

Oracle Database (or Oracle RDBMS) is so prevalent and well established that most users simple refer to it as "Oracle" (ignoring the fact that Oracle, the company, produces other software, and even hardware). Oracle Database (I'll do what most do and just call it "Oracle" to simplify things) has been around since the 1970s, making it one of the earliest database management systems. Oracle is one of the most used database management systems (DBMS) in the world. In fact, most surveys rank it as #1 in database use and popularity worldwide, especially among corporate users, and over the years it has proven itself to be a solid, reliable, fast, and trusted solution to all sorts of data storage needs.

That's the good news. The not-so-good news is that getting started with Oracle can be tricky, especially when compared to some of the alternative DBMSs. Oracle's power, capabilities, security, and more are an important part of why it is so trusted. But that makes installation, configuration, and even the tooling a little more complex, too. On top of that, Oracle's implementation of the SQL language, called PL/SQL, tends to differ subtly from other SQL implementations, and this can make using Oracle just a bit trickier.

What Is This Book?

This book is based on my best-selling *Sams Teach Yourself SQL in 10 Minutes*. That book has become one of the most used SQL tutorials in the world, with an emphasis on teaching what you really need to know—methodically, systematically, and simply. However, as popular and as successful as that book is, it does have some limitations:

- In covering all the major DBMSs, coverage of DBMS-specific features and functionality had to be kept to a minimum.
- To simplify the SQL taught, the lowest common denominator had to be found—SQL statements that would (as much as possible) work with all major DBMSs. This requirement necessitated that better DBMS-specific solutions not be covered.

- Although basic SQL tends to be rather portable between DBMSs, more advanced SQL most definitely is not. As such, that book could not cover advanced topics, such as triggers, cursors, stored procedures, access control, transactions, and more, in any real detail.

And that is where this book comes in. *Sams Teach Yourself Oracle PL/SQL in 10 Minutes* builds on the proven tutorials and structure of *Sams Teach Yourself SQL in Ten Minutes*, without getting bogged down with anything but Oracle and PL/SQL. Starting with simple data retrieval and working toward more complex topics, including the use of joins, subqueries, regular expressions, full text-based searches, stored procedures, cursors, triggers, table constraints, and much more. You'll learn what you need to know methodically, systematically, and simply—in highly focused lessons designed to make you immediately and effortlessly productive.

Who Is This Book For?

This book is for you if

- You are new to SQL.
- You are just getting started with Oracle PL/SQL and want to hit the ground running.
- You want to quickly learn how to get the most out of Oracle and PL/SQL.
- You want to learn how to use Oracle in your own application development.
- You want to be productive quickly and easily using Oracle without having to call someone for help.

It is worth noting that this book is not intended for all readers. If you are an experienced SQL user, then you might find the content in this book to be too elementary. However, if the preceding list describes you and your needs relative to Oracle, you'll find *Sams Teach Yourself Oracle PL/SQL in 10 Minutes* to be the fastest and easiest way to get up to speed with Oracle.

Companion Website

This book has a companion website at `forta.com/books/0672328666`. Visit the site to

- Access table creation and population scripts for creating the example tables used throughout this book
- Visit the online support forum
- Access online errata (if one might be required)
- Find other books that might be of interest to you

Conventions Used in This Book

This book uses different typefaces to differentiate between code and regular English, and also to help you identify important concepts.

Text that you type and text that should appear on your screen appears in `monospace` type. `It looks like this to mimic the way text looks on your screen.`

Placeholders for variables and expressions appear in *`monospace italic`* font. You should replace the placeholder with the specific value it represents.

This arrow (➥) at the beginning of a line of code means that a single line of code is too long to fit on the printed page. Continue typing all the characters after the ➥ as if they were part of the preceding line.

> NOTE
>
> A Note presents interesting pieces of information related to the surrounding discussion.

> TIP
>
> A Tip offers advice or teaches an easier way to do something.

CAUTION

A Caution advises you about potential problems and helps you steer clear of disaster.

New Term sidebars provide clear definitions of new, essential terms.

Input ▼

The Input icon identifies code that you can type in yourself. It usually appears by a listing.

Output ▼

The Output icon highlights the output produced by running Oracle PL/SQL code. It usually appears after a listing.

Analysis ▼

The Analysis icon alerts you to the author's line-by-line analysis of input or output.

LESSON 4

Retrieving Data

In this lesson, you'll learn how to use the `SELECT` *statement to retrieve one or more columns of data from a table.*

The `SELECT` Statement

> NOTE: **Sample Tables Required**
>
> From this point on, all lessons use the sample database tables. If you have yet to install these, please refer to Lesson 3, "Working with Oracle," before proceeding.

As explained in Lesson 1, "Understanding SQL," SQL statements are made up of plain English terms. These terms are called *keywords*, and every SQL statement is made up of one or more keywords. The SQL statement you'll probably use most frequently is the `SELECT` statement. Its purpose is to retrieve information from one or more tables.

To use `SELECT` to retrieve table data, you must, at a minimum, specify two pieces of information—what you want to select, and from where you want to select it.

Retrieving Individual Columns

We'll start with a simple SQL `SELECT` statement, as follows:

Input ▼

```
SELECT prod_name
FROM products;
```

> TIP: **Type Then Execute**
>
> By now it should be obvious, but I'll remind you one last time. Type the SQL code in the Oracle SQL Developer Worksheet screen, and then click the Run Script button to execute it. Results appear in a screen below the Worksheet. If you need more room, you can drag and resize all the screens.

Analysis ▼

The previous statement uses the `SELECT` statement to retrieve a single column called `prod_name` from the `products` table. The desired column name is specified right after the `SELECT` keyword, and the `FROM` keyword specifies the name of the table from which to retrieve the data. The following shows the output from this statement:

Output ▼

```
+----------------+
| prod_name      |
+----------------+
| .5 ton anvil   |
| 1 ton anvil    |
| 2 ton anvil    |
| Oil can        |
| Fuses          |
| Sling          |
| TNT (1 stick)  |
| TNT (5 sticks) |
| Bird seed      |
| Carrots        |
| Safe           |
| Detonator      |
| JetPack 1000   |
| JetPack 2000   |
+----------------+
```

> NOTE: **Unsorted Data**
>
> If you tried this query yourself, you might have discovered that the data displayed in a different order than shown here. If this is the case, don't worry—it is working exactly as it is supposed to. If

query results are not explicitly sorted (we'll get to that in the next lesson), data will be returned in no order of any significance. It might be the order in which the data was added to the table, but it might not. As long as your query returned the same number of rows, then it is working.

A simple `SELECT` statement like the one just shown returns all the rows in a table. Data is not filtered (so as to retrieve a subset of the results), nor is it sorted. We'll discuss these topics in the next few lessons.

NOTE: **Terminating Statements**

Multiple SQL statements must be separated by semicolons (the `;` character). Oracle (like most DBMSs) does not require that a semicolon be specified after single statements. That said, most SQL developers get in the habit of always terminating their SQL statements with semicolons, even when they are not needed.

NOTE: **SQL Statements and Case**

Note that SQL statements are not case sensitive, so `SELECT` is the same as `select`, which is the same as `Select`. Many SQL developers find that using uppercase for all SQL keywords and lowercase for column and table names makes code easier to read and debug.

However, be aware that while the SQL language is not case sensitive, identifiers (the names of databases, tables, and columns) might be. As a best practice, pick a case convention, and use it consistently.

TIP: **Use of White Space**

All extra white space within a SQL statement is ignored when that statement is processed. You can specify SQL statements on one long line or break them up over many lines. Most SQL developers find that breaking up statements over multiple lines makes them easier to read and debug.

Retrieving Multiple Columns

To retrieve multiple columns from a table, you use the same SELECT statement. The only difference is that you must specify multiple column names after the SELECT keyword, and separate each column by a comma.

> TIP: **Take Care with Commas**
>
> When selecting multiple columns, be sure to specify a comma between each column name, but not after the last column name. Doing so generates an error.

The following SELECT statement retrieves three columns from the products table:

Input ▼

```
SELECT prod_id, prod_name, prod_price
FROM products;
```

Analysis ▼

Just as in the prior example, this statement uses the SELECT statement to retrieve data from the products table. In this example, three column names are specified, each separated by a comma. The output from this statement is as follows:

Output ▼

```
+---------+----------------+------------+
| prod_id | prod_name      | prod_price |
+---------+----------------+------------+
| ANV01   | .5 ton anvil   |       5.99 |
| ANV02   | 1 ton anvil    |       9.99 |
| ANV03   | 2 ton anvil    |      14.99 |
| OL1     | Oil can        |       8.99 |
| FU1     | Fuses          |       3.42 |
| SLING   | Sling          |       4.49 |
| TNT1    | TNT (1 stick)  |        2.5 |
| TNT2    | TNT (5 sticks) |         10 |
| FB      | Bird seed      |         10 |
```

```
| FC      | Carrots        |        2.5 |
| SAFE    | Safe           |         50 |
| DTNTR   | Detonator      |         13 |
| JP1000  | JetPack 1000   |         35 |
| JP2000  | JetPack 2000   |         55 |
+---------+----------------+------------+
```

> NOTE: **Presentation of Data**
>
> SQL statements typically return raw, unformatted data. Data formatting is a presentation issue, not a retrieval issue. Therefore, presentation (for example, alignment and displaying the price values as currency amounts with the currency symbol and commas) is typically specified in the application that displays the data. Actual raw retrieved data (without application-provided formatting) is rarely displayed as is.

Retrieving All Columns

In addition to being able to specify desired columns (one or more, as shown previously), you can also use `SELECT` statements to request all columns without having to list them individually. This is done using the asterisk (`*`) wildcard character in lieu of actual column names, as follows:

Input ▼

```
SELECT *
FROM products;
```

Analysis ▼

When you specify a wildcard (`*`), all the columns in the table are returned. The columns are in the order in which the columns appear in the table definition. However, you cannot rely on this because changes to table schemas (adding and removing columns, for example) could cause ordering changes.

> CAUTION: **Using Wildcards**
>
> As a rule, you are better off not using the * wildcard unless you really do need every column in the table. Even though use of wildcards might save you the time and effort needed to list the desired columns explicitly, retrieving unnecessary columns usually slows down the performance of your retrieval and your application.

> TIP: **Retrieving Unknown Columns**
>
> There is one big advantage to using wildcards. As you do not explicitly specify column names (because the asterisk retrieves every column), it is possible to retrieve columns whose names are unknown.

Retrieving Distinct Rows

As you have seen, `SELECT` returns all matched rows. But what if you did not want every occurrence of every value? For example, suppose you wanted the vendor ID of all vendors with products in your `products` table:

Input ▼

```
SELECT vend_id
FROM products;
```

Output ▼

```
+---------+
| vend_id |
+---------+
|    1001 |
|    1001 |
|    1001 |
|    1002 |
|    1002 |
|    1003 |
|    1003 |
|    1003 |
|    1003 |
```

```
|    1003 |
|    1003 |
|    1003 |
|    1005 |
|    1005 |
+---------+
```

The `SELECT` statement returned 14 rows (even though only 4 vendors are in that list) because 14 products are listed in the `products` table. So how could you retrieve a list of distinct values?

The solution is to use the `DISTINCT` keyword which, as its name implies, instructs Oracle to only return distinct values:

Input ▼

```
SELECT DISTINCT vend_id
FROM products;
```

Analysis ▼

`SELECT DISTINCT vend_id` tells Oracle to only return distinct (unique) `vend_id` rows, and so only 4 rows are returned, as shown in the following output. If you use it, you must place the `DISTINCT` keyword directly in front of the column names:

Output ▼

```
+---------+
| vend_id |
+---------+
|    1001 |
|    1002 |
|    1003 |
|    1005 |
+---------+
```

> CAUTION: **Can't Be Partially** `DISTINCT`
>
> The `DISTINCT` keyword applies to all columns, not just the one it precedes. If you were to specify `SELECT DISTINCT vend_id, prod_price`, all rows would be retrieved unless *both* of the specified columns were distinct.

Using Fully Qualified Table Names

The SQL examples used thus far have referred to columns by just the column names. Referring to columns using fully qualified names (using both the table and column names) is also possible. Look at this example:

Input ▼

```
SELECT products.prod_name
FROM products;
```

This SQL statement is functionally identical to the first one used in this lesson, but here a fully qualified column name is specified.

Table names, too, may be fully qualified, as shown here:

Input ▼

```
SELECT products.prod_name
FROM crashcourse.products;
```

Once again, this statement is functionally identical to the one just used (assuming, of course, that the `products` table is indeed in the `crashcourse` database).

Situations exist where fully qualified names are required, as we will see in later lessons. For now, it is worth noting this syntax so you'll know what it is if you run across it.

Using Comments

As you have seen, SQL statements are instructions that Oracle processes. But what if you wanted to include text that you do not want processed and executed? Why would you ever want to do this? Here are a few reasons:

- The SQL statements we've been using here are all very short and very simple. But, as your SQL statement grows (in length and complexity), you'll want to include descriptive comments (for your own future reference or for whoever has to work on the project next). You need to embed these comments in the SQL scripts, but they are obviously not intended for Oracle

processing. (For an example of this, see the `create.sql` and `populate.sql` files you used in Lesson 3.)

- The same is true for headers at the top of a SQL file, perhaps containing the programmer contact information and a description and notes. (You also see this use case in the `create.sql` and `populate.sql` files.)
- Another important use for comments is to temporarily stop SQL code from being executed. If you were working with a long SQL statement and wanted to test just part of it, you could *comment out* some of the code so that Oracle saw it as comments and ignored it.

Oracle supports two forms of comment syntax. We'll start with inline comments:

Input ▼

```
SELECT prod_name   -- this is a comment
FROM products;
```

Analysis ▼

You may embed comments inline using `--` (two hyphens). Anything after the `--` is considered comment text, making this a good option for describing columns in a `CREATE TABLE` statement, for example.

Here is another form of inline comment:

Input ▼

```
-- This is a comment
SELECT prod_name
FROM products;
```

Analysis ▼

A `--` at the start of a line makes the entire line a comment. You can see this format comment used in the accompanying `create.sql` and `populate.sql` scripts.

You can also create multi-line comments, and comments that stop and start anywhere within the script:

Input ▼

```
/* SELECT prod_name, vend_id
FROM products; */

SELECT prod_name
FROM products;
```

Analysis ▼

`/*` starts a comment, and `*/` ends it. Anything between `/*` and `*/` is comment text. This type of comment is often used to *comment out* code, as shown in this example. Here, two `SELECT` statements are defined, but the first won't execute because it has been commented out.

> TIP: **Oracle SQL Developer Color Coding**
>
> You might have noticed that Oracle SQL Developer color codes your PL/SQL. SQL statements are usually displayed in blue, identifiers (like table and column names) are in black, and so on. Color coding makes it easier to read your code and to find mistakes; if you've mistyped a PL/SQL statement, it'll probably appear in the wrong color. Oracle SQL Developer also color codes any comments (inline or multi-line) and displays them in a light gray. This makes it easy to locate comments and commented-out code (and can also help you find code that you no longer want commented out).

Summary

In this lesson, you learned how to use the SQL `SELECT` statement to retrieve a single table column, multiple table columns, and all table columns. You also learned about commenting and saw various ways that you can use comments. In the next lesson, you'll learn how to sort the retrieved data.

This page intentionally left blank

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W–X–Y–Z