

ORACLE DATABASE PROGRAMMING

PL/SQL Programming from Zero

Learn Oracle Database Programming Step by Step — From Your First Block to Packages, Triggers, and Real Projects

No experience needed · Learn just enough SQL, then build the complete database logic for a real shop

PROCEDURES · CURSORS · PACKAGES · TRIGGERS

THENEXBOOKS

PL/SQL Programming from Zero

Learn Oracle Database Programming Step by Step — From Your First Block to Packages, Triggers, and Real Projects

First Edition

THENEXBOOKS

PL/SQL Programming from Zero
Learn Oracle Database Programming Step by Step — From Your First Block to Packages, Triggers, and Real Projects

First Edition, 2026

Copyright © 2026 thenexbooks (thenexbooks.gumroad.com). All rights reserved.

No part of this publication may be reproduced, distributed, or transmitted in any form or by any means without the prior written permission of the publisher, except for brief quotations in reviews and certain other non-commercial uses permitted by copyright law. The code in this book may be used freely in your own projects.

The information in this book is provided "as is" without warranty of any kind. While every precaution has been taken in its preparation, the publisher assumes no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein. Database software and tools change over time; consult the official documentation for the versions you use.

This book is an independent publication and is not affiliated with, sponsored by, or endorsed by Oracle Corporation. Oracle, Java, MySQL, and PL/SQL are trademarks or registered trademarks of Oracle and/or its affiliates. Other product names are trademarks of their respective owners; their use here is for identification only. Bluewater Outfitters, the town of Pine Harbor, and all people and data in the examples are fictional.

Preface

Behind many banking systems, airline bookings, hospital records, and online shops sits an Oracle database — and inside many of those databases runs PL/SQL. PL/SQL is Oracle's procedural language for the database. It takes the power of SQL, which describes *what* data you want, and adds what SQL alone cannot do: variables, decisions, loops, error handling, and reusable procedures that live right next to the data they work on.

Learning PL/SQL is a practical skill. It lets you enforce business rules in one trusted place, process thousands of rows efficiently, react automatically when data changes, and build reliable back-end logic that every application connected to the database can share. It is also a friendly first programming language: its syntax reads almost like English, and every program you write works with real, visible data.

This book starts from zero. You do not need to know SQL or have programmed before. We begin by setting up a free Oracle environment and learning just enough SQL, then add one PL/SQL idea at a time — blocks, variables, conditions, loops, exceptions, procedures, cursors, collections, packages, and triggers — and use each one immediately in a realistic project.

That project is the database for **Bluewater Outfitters**, a small outdoor-gear shop. Chapter by chapter you will write the code that places orders, applies loyalty discounts and shipping rules, protects stock levels, logs errors, audits price changes, produces reorder and sales reports, and even returns JSON. By the end you will have built a complete set of tested PL/SQL packages and you will be ready to read and write professional database code.

Who This Book Is For

- **Complete beginners** who want to learn database programming — no SQL or programming experience is assumed.
- **Students** in database courses who need a patient, practical guide to PL/SQL.
- **Developers** who know another language and now work with Oracle databases.
- **Analysts and DBAs** who write SQL every day and want to automate and package their work.

What You Will Learn

- How to set up a free Oracle environment, connect with SQL Developer or SQLcl, and run scripts.
- Just enough SQL to create tables, query data, and manage transactions.
- PL/SQL blocks, variables, data types, conditions, loops, and exception handling.
- Procedures, functions, cursors, records, collections, and fast bulk processing.
- Packages, triggers, dynamic SQL, and working with dates, strings, and numbers.
- Debugging, testing, performance basics, security, logging, reports, scheduling, and JSON.

How This Book Is Organized

Part I — Getting Started (Chapters 1–5) sets up your environment, teaches the SQL you need, and writes your first PL/SQL blocks. **Part II — Control Flow and Program Units** (Chapters 6–10) covers decisions, loops, SQL inside PL/SQL, exceptions, procedures, and functions. **Part III — Working with Data** (Chapters 11–15) introduces cursors, records, collections, and bulk processing, and ends with a complete order-processing project. **Part IV — Building Reusable Code** (Chapters 16–19) covers packages, triggers, dynamic SQL, and built-in functions. **Part V — Professional PL/SQL** (Chapters 20–24) is about debugging, testing, performance, security, and logging. **Part VI — Projects and Next Steps** (Chapters 25–28) builds the inventory and reporting packages, explores JSON and modern features, and points the way forward. The appendices provide a cheat sheet, a reference of built-in functions and packages, checklists, and a glossary.

Every chapter ends with **Try It Yourself** exercises and a **Key Takeaways** box.

Welcome to Bluewater Outfitters

Bluewater Outfitters is a small (and fictional) outdoor-gear shop in the town of Pine Harbor, selling tents, packs, footwear, clothing, and camp-cooking gear in its store and online. Nora Quinn owns the shop, Eli Brooks manages the store and looks after the stock, and Maya Chen runs the online orders and the reports. Their database already holds customers, products, and a few orders. Over the course of this book, you will write the PL/SQL that runs the business behind it.

How to Use This Book

Type the examples yourself and run them against your own copy of the Bluewater schema, which you create in Chapter 3. Keep the setup script handy: if an experiment changes the data in a way you did not intend, you can drop the tables and run the script again to start fresh. When Oracle reports an error, read the message and the line number carefully — they nearly always point straight at the problem, and Chapter 20 shows you how to follow them.

Conventions Used in This Book

> **TIP**
>
> A recommended practice or a shortcut that makes your work easier.

> **NOTE**
>
> Background information, version differences, or a useful aside.

> **WARNING**
>
> A common mistake or something that can cause errors or lost data.

Code appears in `monospace`. SQL and PL/SQL keywords are written in UPPERCASE and names in lowercase. Scripts appear in dark boxes with their file name above them; what the database prints appears in lighter boxes labeled **Output**. Features that need Oracle Database 23ai or later are clearly marked; everything else runs on Oracle Database 19c and later.

Contents

Preface

PART I — GETTING STARTED

1 What Is PL/SQL, and Why Learn It?

2 Setting Up: Oracle Database, SQL Developer, and Your First Connection

3 Just Enough SQL: Tables, Queries, and the Bluewater Schema

4 Your First PL/SQL Block

5 Variables, Data Types, and %TYPE

PART II — CONTROL FLOW AND PROGRAM UNITS

6 Conditional Logic: IF and CASE

7 Loops

8 SQL Inside PL/SQL: SELECT INTO, DML, and Transactions

9 Exception Handling

10 Procedures and Functions

PART III — WORKING WITH DATA

11 Cursors

12 Records and %ROWTYPE

13 Collections

14 Bulk Processing with BULK COLLECT and FORALL

15 Project: Order Processing

PART IV — BUILDING REUSABLE CODE

16 Packages: Specification and Body

17 Triggers

18 Dynamic SQL

19 Working with Dates, Strings, and Numbers

PART V — PROFESSIONAL PL/SQL

20 Debugging and Tracing

21 Testing PL/SQL

22 Performance Basics

23 Security: Privileges, Definer and Invoker Rights, and SQL Injection

24 Error Logging and Auditing

PART VI — PROJECTS AND NEXT STEPS

25 Project: Inventory and Reorder Package

26 Project: Sales Reports with Ref Cursors and Scheduling

27 JSON and Modern PL/SQL Features

28 Next Steps

APPENDICES

A PL/SQL Cheat Sheet

B Built-in Functions and Packages Reference

C Checklists

D Glossary

PART I

Getting Started

Set up a free Oracle environment, learn just enough SQL, create the Bluewater schema, and write your first PL/SQL blocks.

1 · What Is PL/SQL, and Why Learn It?

2 · Setting Up: Oracle Database, SQL Developer, and Your First Connection

3 · Just Enough SQL: Tables, Queries, and the Bluewater Schema

4 · Your First PL/SQL Block

5 · Variables, Data Types, and %TYPE

CHAPTER 1 · PART I — GETTING STARTED

What Is PL/SQL, and Why Learn It?

Before we write a single line of code, we meet the language, the database it lives in, and the small outdoor-gear shop whose order system we will build together, one careful step at a time, across the rest of this book.

A Shop with a Growing Problem

Imagine a small outdoor-gear shop in a harbor town. It sells tents, backpacks, hiking boots, rain jackets, and camping stoves. It has a store on the main street and a modest online shop. For years the owner kept track of stock in a spreadsheet and wrote orders on paper. Then the online shop took off, and the spreadsheet started to crack: two people sold the last pair of trail runners at the same time, a discount was applied twice, and nobody could say quickly which products needed reordering.

The shop moved its data into a database. That solved part of the problem, because a database keeps one shared, consistent copy of the facts. But the *rules* of the business — how much discount a loyal customer gets, when shipping is free, that you cannot sell stock you do not have — still lived in people's heads and in scattered pieces of application code. Every new screen or report had to repeat those rules, and sooner or later someone repeated them slightly wrong.

This is exactly the kind of problem PL/SQL was designed to solve. PL/SQL lets you put the rules *inside* the database, right next to the data they protect, so that every program that touches the data goes through the same well-tested logic. In this book you will learn to write that logic yourself, starting from zero. The shop is called **Bluewater Outfitters**, and we will meet its people properly later in this chapter.

First, though, we need a shared vocabulary. If you have never worked with a database, the next two sections give you the plain-words version. If you already know what a table and a query are, feel free to skim them — but do read the section on what PL/SQL adds, because it frames everything that follows.

Databases in Plain Words

A **database** is an organized, permanent store of information that many people and programs can use at once. The kind of database we use in this book is a **relational database**, and the specific product is **Oracle Database**. A relational database stores information in **tables**. A table looks a lot like a spreadsheet: it has named **columns** across the top and **rows** of data underneath. Each row describes one thing — one customer, one product, one order.

Here is a small piece of the Bluewater product table, written out the way a database tool might show it:

Query Result

```
PRODUCT_ID SKU    NAME                   CATEGORY UNIT_PRICE  STOCK_QTY
---------- ------ ---------------------- -------- ---------- ----------
       101 TNT-2P Trailhead 2P Tent      TENTS           249         12
       106 FTW-TR Swift Trail Runner     FOOTWEAR        119          4
       109 CKG-ST Pocket Stove           COOKING          49         30
```

Unlike a spreadsheet, a database table is strict. Every column has a **data type**: `unit_price` holds numbers, `name` holds text, and an order date holds a date. You cannot accidentally type "twelve" into a number column. Tables can also carry **constraints** — rules the database enforces on every change. For example, Bluewater's product table will refuse a negative stock quantity and will refuse a category that is not one of the five the shop uses. Constraints are the database's first line of defense, and we will create them in Chapter 3.

Each table usually has a **primary key**: a column (or set of columns) whose value is unique for every row and never empty. In the product table above, `product_id` is the primary key, so product 101 is always the Trailhead 2P Tent and there can never be two products numbered 101. Tables are connected by **foreign keys**: an order row stores the `customer_id` of the customer who placed it, and the database guarantees that this customer actually exists. That web of connected tables is what the word *relational* refers to.

The database software runs on a **server** — a computer, a virtual machine, or a container somewhere. Programs and people connect to it over a network, even when "the network" is just your own laptop talking to itself. The database handles many connections at once, keeps everyone's changes from trampling each other, and makes sure that a confirmed change survives a power cut. Those guarantees are a big part of why businesses trust databases with their most important data.

SQL: Asking for What You Want

You talk to a relational database with **SQL**, the Structured Query Language. SQL is the standard language for creating tables, adding and changing data, and asking questions about it. Nearly every relational database speaks SQL, with small dialect differences, so the time you invest in it pays off far beyond Oracle.

The most important thing to understand about SQL is that it is **declarative**. You describe *what* you want, not *how* to find it. Here is a question Eli, the store manager, asks every week: which products are running low?

low_stock.sql

```
SELECT product_id, name, stock_qty, reorder_level
FROM   products
WHERE  stock_qty <= reorder_level
ORDER  BY product_id;
```

Query Result

```
PRODUCT_ID NAME                 STOCK_QTY REORDER_LEVEL
---------- -------------------- ---------- -------------
       106 Swift Trail Runner            4             6
       110 Titanium Pot Set              3             5
```

Notice what the statement does *not* say. It does not tell the database to open a file, read the rows one by one, compare two numbers, and remember the matches. It simply states the condition, and the database works out the fastest way to answer. That is the power of SQL: one short statement can scan, filter, join, sort, and summarize thousands or millions of rows.

You do not need to understand every word of that query yet. Chapter 3 teaches "just enough SQL" to follow the rest of the book, using the Bluewater tables. For now, the key idea is that SQL works on *sets* of rows at a time and leaves the step-by-step mechanics to the database.

What PL/SQL Adds

SQL is wonderful at questions and at changes that can be described in a single statement. But real business processes are rarely one statement long. Consider what has to happen when Maya, who runs the online shop, takes a new order:

1. Check that the customer exists.
2. For each product on the order, check that there is enough stock.
3. If there is, record the order line and reduce the stock; if not, refuse that line with a clear message.
4. Look up the customer's loyalty level and work out the discount.
5. Decide whether shipping is free or costs a flat fee.
6. Save the final total — or, if anything went wrong, undo the whole thing so no half-finished order is left behind.

Those steps involve *decisions* ("if there is enough stock"), *repetition* ("for each product"), *remembered values* (the discount you worked out in step 4 is needed in step 6), and *error handling* ("if anything went wrong"). Plain SQL has no variables, no `IF` statements, and no loops. That is where **PL/SQL** comes in.

PL/SQL stands for **Procedural Language extensions to SQL**. It is Oracle's programming language for the database, and Oracle has shipped it since the early 1990s. "Procedural" means you describe a procedure — a sequence of steps — rather than only a result. PL/SQL wraps ordinary SQL statements inside a full programming language with:

- **Variables and constants** to hold values while your code runs (Chapter 5).
- **Conditions** with `IF` and `CASE` to make decisions (Chapter 6).
- **Loops** to repeat work (Chapter 7).
- **SQL inside the code**, so a query result can land straight in a variable (Chapter 8).

- **Exception handling** to catch and deal with errors in an organized way (Chapter 9).
- **Named program units** — procedures, functions, packages, and triggers — that are stored in the database and can be called by anyone with permission (Chapters 10, 16, and 17).

Here is a tiny taste of what PL/SQL looks like. It works out the loyalty discount on one of Bluewater's real orders: Ava Patel, a GOLD customer, bought goods worth $347.00, and GOLD customers get 10% off. You do not need to run this yet — Chapter 4 explains every line — but read it and see how much you can guess.

first_taste.sql

```
SET SERVEROUTPUT ON

DECLARE
  l_subtotal      NUMBER := 347;
  l_discount_pct  NUMBER := 10;   -- GOLD customers get 10%
  l_discount      NUMBER;
BEGIN
  l_discount := l_subtotal * l_discount_pct / 100;
  DBMS_OUTPUT.PUT_LINE('Discount: ' || l_discount);
  DBMS_OUTPUT.PUT_LINE('To pay:   ' || (l_subtotal - l_discount));
END;
/
```

Output

```
Discount: 34.7
To pay:   312.3

PL/SQL procedure successfully completed.
```

Even without any training, you can probably see the shape: a `DECLARE` section that creates three named values, a `BEGIN` section that does some arithmetic and prints two lines, and an `END`. The two dashes start a comment, which the database ignores. The vertical bars `||` join pieces of text together. And the lonely slash on the last line tells the tool "the block is finished, run it now." That is a complete PL/SQL program, and by the end of Chapter 4 you will be writing ones like it from memory.

> **NOTE**
>
> Oracle prints the total as `312.3`, not `312.30`. When a number is turned into text without instructions, Oracle drops trailing zeros. That is fine for checking arithmetic, and in later chapters we use `TO_CHAR` with a format mask whenever we want money to look like money.

SQL and PL/SQL Work Together

A common beginner question is whether PL/SQL replaces SQL. It does not. PL/SQL *contains* SQL. The heavy lifting on data — finding rows, joining tables, adding up totals — is still done by SQL statements, and good PL/SQL code lets SQL do as much of that work as possible. PL/SQL provides the

glue: the decisions, the sequencing, the error handling, and the packaging into reusable, named units. A useful rule of thumb that we return to in Chapter 22 is: *if one SQL statement can do the job, use one SQL statement; reach for PL/SQL when you need logic around it*.

Where PL/SQL Runs

Most programming languages you may have heard of — Python, Java, JavaScript — usually run on a separate application server or on your own computer, and send SQL statements to the database over the network. PL/SQL is different: it runs **inside the database server itself**. Oracle Database contains two cooperating engines. The **SQL engine** runs SQL statements. The **PL/SQL engine** runs the procedural parts — variables, loops, `IF` statements — and hands each embedded SQL statement to the SQL engine. Because both engines live in the same server process, very close to the data, a PL/SQL program can run many SQL statements without a network round trip for each one.

Switching between the two engines is cheap but not free. Each switch is called a **context switch**, and in Chapter 14 we learn bulk techniques that reduce the number of switches when processing many rows.

PL/SQL code comes in two broad forms:

- **Anonymous blocks** are chunks of code that you type into a tool and run once, like the discount example above. They are not saved in the database. They are perfect for learning, testing, and one-off jobs. Chapter 4 is all about them.
- **Stored program units** — procedures, functions, packages, and triggers — are compiled and saved in the database under a name. Any program with permission can call them, from any language. A web shop written in Java, a report built in a spreadsheet tool, and a nightly job can all call the same stored procedure to place an order, and they all get exactly the same rules. A **trigger** is special: it runs automatically when data changes, without anyone calling it.

You write PL/SQL with a client tool on your own computer — in this book, **SQL Developer** or the command-line tool **SQLcl**, or the browser-based **Oracle Live SQL** — but the tool merely sends your code to the database. The code itself always executes on the server. Chapter 2 walks you through setting up those tools.

PL/SQL and Other Databases

Other database products have their own procedural languages. Microsoft SQL Server uses Transact-SQL, and PostgreSQL has PL/pgSQL, which was openly modeled on PL/SQL. The ideas you learn here — blocks, variables, cursors, exceptions, stored procedures, triggers — transfer well to those languages, even though the exact syntax differs. This book, however, is about Oracle PL/SQL only, and every listing is written for Oracle Database.

Why Learn PL/SQL?

PL/SQL is not a fashionable new language, and that is part of its appeal: it is stable, well documented, and deeply woven into systems that businesses rely on every day. Here are the practical reasons it is worth your time.

- **Rules live next to the data.** When the discount rule is a stored function, every application that places an order uses the same rule. Change it once, and everything follows. Nobody can bypass the stock check by writing a clever script, because the check sits in the database.
- **Fewer round trips.** Placing an order may need a dozen SQL statements. If an application sends them one by one across a network, each one waits for a reply. A stored procedure runs them all inside the server and returns once.
- **Safety through transactions.** PL/SQL makes it natural to group several changes into one transaction that either succeeds completely or is undone completely. Half an order is never saved.
- **Security.** You can let an application call `place_order` without giving it the right to update the product table directly. Users get exactly the access they need and no more. Chapter 23 shows how.
- **It is everywhere Oracle is.** Many existing business systems built on Oracle Database contain large amounts of PL/SQL that someone needs to understand, maintain, and improve. Oracle's own low-code tool, APEX, uses PL/SQL for its server-side logic.
- **It is a gentle first language.** PL/SQL is readable, strongly typed, and forgiving of beginners in the sense that the compiler catches many mistakes before the code ever runs. Its structure — declare, begin, handle errors, end — teaches good habits.

We will not pretend PL/SQL is the right tool for everything. It is a database language: it is excellent at data-centered business logic and poor at, say, drawing user interfaces or crunching images. Chapter 17 discusses when *not* to use triggers, and Chapter 22 shows when plain SQL beats PL/SQL, because good judgment matters as much as syntax.

> **TIP**
>
> You will learn fastest by typing the examples yourself rather than copying and pasting them. Typing forces you to notice every semicolon and every keyword, and the small typos you make along the way teach you to read error messages — a skill that matters more than any single feature of the language.

Meet Bluewater Outfitters

Bluewater Outfitters is a fictional outdoor-gear shop in the fictional town of **Pine Harbor**. It has a small store and an online shop, and throughout this book you will build its database logic: placing orders, controlling stock, producing reorder reports, logging what happens, and auditing changes. Every example, exercise, and project uses the same tables and the same sample data, so by the end of the book you will know the shop as well as its staff do.

The People

- **Nora Quinn** is the owner. She cares about the rules being applied fairly and consistently, and about knowing how the business is doing. Her questions often turn into reports.
- **Eli Brooks** is the store manager and looks after stock. He wants to know what is running low before it runs out, and he wants every stock change to be traceable.
- **Maya Chen** runs online orders and reports. She needs order placement to be reliable, fast, and impossible to get wrong, and she wants clear error messages when something is refused.

When a chapter needs a reason to write some code, it usually comes from one of these three people, so every feature you learn solves a real problem at the shop.

The Business Rules

These rules are the heart of the project. We will turn each of them into PL/SQL during the book, and the same numbers appear in every chapter, so it is worth reading them carefully now.

Rule	Details
Loyalty discount	Applied to the order subtotal: BRONZE 0%, SILVER 5%, GOLD 10%.
Shipping	A flat $8.00, but free when the discounted subtotal is $150.00 or more. No tax in our examples.
Stock	Stock must never go negative. An order line for more than the stock on hand fails.
Reordering	A product needs reordering when its stock quantity is at or below its reorder level.
Order statuses	NEW, PAID, SHIPPED, or CANCELLED.
Changing orders	Only orders with status NEW can be changed or cancelled.

Let us check the rules against a real order from the sample data. Liam Walsh is a SILVER customer. His order contains one Granite Hiking Boot at $159.00 and one Stormshell Rain Jacket at $179.00, a subtotal of $338.00. SILVER means 5% off, which is $16.90, leaving $321.10. That is well over $150.00, so shipping is free, and the order total is $321.10. Sofia Rossi, a BRONZE customer, bought one Campfire Fleece at $79.00. She gets no discount, and $79.00 is below the free-shipping threshold, so she pays $8.00 shipping for a total of $87.00. By Chapter 10 you will have functions that do this arithmetic for you, and by Chapter 15 a procedure that builds whole orders.

The Tables

The Bluewater database has six tables. Chapter 3 creates all of them with one script, `bluewater_setup.sql`, and fills them with sample data. Here is the map:

Table	What it holds
`customers`	Six customers, with email, city, and loyalty level (BRONZE, SILVER, or GOLD).
`products`	Ten products in five categories: TENTS, PACKS, FOOTWEAR, CLOTHING, COOKING. Price, stock quantity, and reorder level for each.
`orders`	One row per order: customer, date, status, and total amount. Three sample orders, numbered 1001 to 1003.
`order_items`	The lines of each order: which product, how many, at what unit price.
`stock_movements`	A history of every stock change: goods in, goods out, and adjustments. Empty at the start.
`app_log`	Messages written by our programs, with a time and a level (INFO, WARN, ERROR). Empty at the start.

The sample data is deliberately small — six customers, ten products, three orders — so that you can check every result in your head or on paper. When a chapter prints a total or a count, you can verify it against the data yourself, which is a great habit for any programmer.

What We Will Build

The book is organized in six parts. Each part builds on the one before, and each adds a visible piece to the Bluewater system.

- **Part I — Getting Started (Chapters 1–5).** You set up a free Oracle environment, create the Bluewater tables, learn enough SQL to work with them, and write your first PL/SQL blocks with variables and data types.
- **Part II — Control Flow and Program Units (Chapters 6–10).** You add decisions, loops, SQL inside PL/SQL, transactions, and error handling, and you write your first stored functions: `get_discount_pct`, which returns the discount percentage for a loyalty level, and `calc_shipping`, which returns the shipping charge for a subtotal.
- **Part III — Working with Data (Chapters 11–15).** You process many rows with cursors, records, collections, and bulk operations, and then put it all together in the first big project: the order-processing procedures `place_order`, `add_item`, `recalc_total`, and `cancel_order`.
- **Part IV — Building Reusable Code (Chapters 16–19).** You gather the order logic into a package called `order_pkg`, add triggers that audit price changes, learn dynamic SQL, and become fluent with dates, strings, and numbers.
- **Part V — Professional PL/SQL (Chapters 20–24).** You learn to debug, test, tune, and secure your code, and you build `log_pkg`, a logging package that records what your programs do.
- **Part VI — Projects and Next Steps (Chapters 25–28).** You build `inventory_pkg` for stock control and reorder reports, and `report_pkg` for sales reports and a scheduled job, explore JSON and modern features, and plan where to go next.

By the end, Bluewater Outfitters will have a small but genuinely professional database back end: orders that cannot oversell stock, discounts and shipping applied consistently, a full movement history for every product, readable error messages, logs, audits, and reports. More importantly, you will understand every line of it, because you wrote it.

How to Use This Book

Each chapter introduces one idea at a time. A new idea first appears in a tiny block that you can run on its own; then we use it in the Bluewater project. Please run the examples as you go. Reading code teaches you some things, but running it — and breaking it, and fixing it — teaches you far more.

Code, Output, and File Names

Code listings appear in dark boxes. When a listing has a label above it, such as `first_taste.sql`, that is a suggested file name if you want to save it. Results appear in light boxes labeled **Output**, **Script Output**, or **Query Result**. Query results are shown as plain fixed-width text; your tool may display them in a grid instead, possibly with slightly different column widths, but the values will be the same.

In the text, SQL and PL/SQL keywords are written in UPPERCASE (`SELECT`, `BEGIN`, `END IF`) and names we choose are written in lowercase with underscores (`stock_qty`, `place_order`). Oracle does not care about case for keywords and unquoted names, but the convention makes code much easier to read.

Naming Conventions

We use short prefixes so that you can tell at a glance what kind of thing a name refers to. You will meet each of these as the book goes on:

Prefix	Meaning	Example
`p_`	Parameter of a procedure or function	`p_order_id`
`l_`	Local variable	`l_subtotal`
`g_`	Package-level (global) variable	`g_debug`
`c_`	Constant	`c_free_shipping_min`
`e_`	User-defined exception	`e_insufficient_stock`
`r_`	Record	`r_product`
`t_`	Type	`t_id_list`

We indent code by two spaces, end every named unit with `END unit_name;`, and put a slash on its own line after every PL/SQL block or stored unit in a script. These are habits shared by many Oracle teams, and they will make your code look familiar to other developers.

Boxes Along the Way

Three kinds of boxes appear in every chapter. A **Tip** offers a shortcut or a good habit. A **Note** adds background or explains a detail you might otherwise wonder about. A **Warning** points out a mistake that is easy to make and painful to discover later. Each chapter ends with a **Try It Yourself** section of small exercises and a **Key Takeaways** box that summarizes what you learned.

> **WARNING**
> Practice on a database that holds nothing important. Several exercises in this book deliberately delete rows, break code, and roll back changes. Chapter 2 shows how to set up a free personal database and a dedicated practice user, so you never have to experiment on real business data.

Versions

All main-text code in this book is written for **Oracle Database 19c and later**. A few newer features that need **Oracle Database 23ai and later** are clearly marked when they appear, so you will never be surprised by code that works on one version and not another. The free database options described in Chapter 2 are recent versions, so everything in the book runs on them.

Errors Are Normal

One last piece of advice before we start. You will see a lot of error messages while learning PL/SQL. Oracle errors have codes like `ORA-01403` or `PLS-00201`, followed by a short description. They can look intimidating, but they are simply the database telling you, quite precisely, what it did not understand or could not do. Every error message in this book is explained, and from Chapter 4 onward you will practice reading them. An error is not a failure; it is information.

Experienced developers see error messages all day long. The difference between a beginner and an expert is not that the expert makes fewer mistakes — it is that the expert reads the message calmly, finds the line it points to, and fixes the cause. You will build that habit from the very first block you write.

Try It Yourself

1. Using the business rules table, work out by hand the total for an order from Noah Kim (GOLD) containing one Summit 65L Pack at $219.00. Is shipping free? (Answer: 10% off gives $197.10, which is over $150.00, so shipping is free and the total is $197.10.)
2. Do the same for Grace Okoye (BRONZE) buying one Pocket Stove at $49.00 and one Titanium Pot Set at $69.00. What is the total, including shipping?
3. Look at the low-stock query result in this chapter. Which business rule does its `WHERE` line express? Write the rule in your own words.
4. Read the `first_taste.sql` block again and list every name that starts with `l_`. Using the naming conventions table, explain what kind of thing each one is.

5. Write down one task at a shop, school, or club you know that involves several steps, a decision, and a possible error. Which parts would be SQL, and which parts would need PL/SQL?

Key Takeaways

- A relational database stores data in tables of rows and columns, protected by data types, constraints, primary keys, and foreign keys.
- SQL is declarative: you describe the result you want and the database decides how to get it.
- PL/SQL is Oracle's procedural extension to SQL, adding variables, conditions, loops, error handling, and named program units.
- PL/SQL runs inside the database server, where a PL/SQL engine and a SQL engine cooperate close to the data.
- Anonymous blocks are run once from a tool; stored procedures, functions, packages, and triggers are saved in the database and shared.
- Bluewater Outfitters — Nora, Eli, and Maya, six tables, and a short list of business rules — is the running project for the whole book.
- The code targets Oracle Database 19c and later; newer 23ai features are always marked.
- Error messages are normal and informative; learning to read them calmly is a core skill.