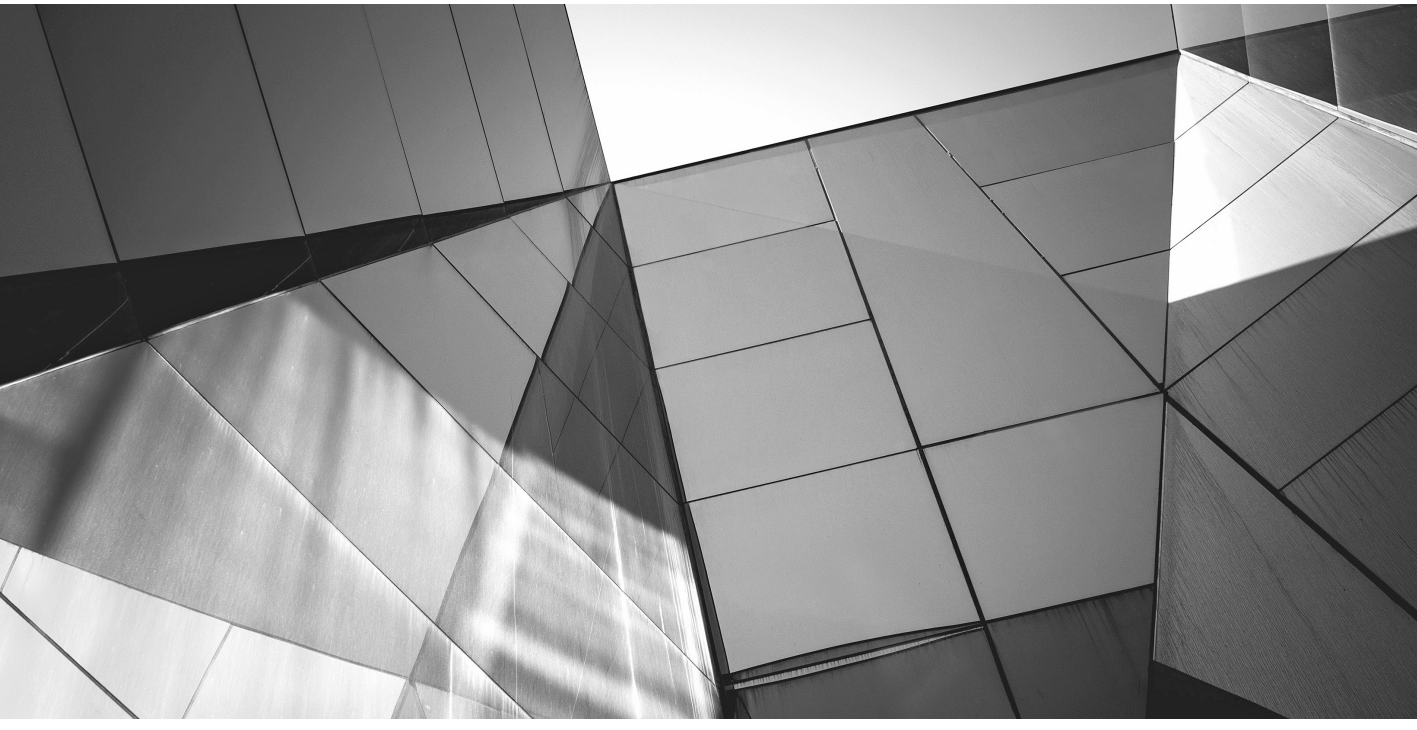


CHAPTER 1

Oracle PL/SQL Development Overview

This chapter introduces you to Oracle PL/SQL development. Understanding the how, what, and why of a programming language provides a strong foundation for learning how to use the programming language effectively to solve problems.

This chapter covers the following:

- PL/SQL's history and background
- Oracle development architecture

The development examples in this book are presented using the SQL*Plus tool because it's the lowest common denominator when it comes to Oracle development. Although development tools such as Dell's Toad and Oracle SQL Developer are great in many ways, they also have a few weaknesses. Their greatness lies in simplifying tasks and disclosing metadata that otherwise might be hidden for months or years. Their weaknesses are more subtle. Tools provide opportunities to solve problems without requiring that you understand either the problem or the solution. Occasionally, this may lead you to choose a suggested solution that is suboptimal or incorrect. Relying on tools also stymies the learning process for new developers. While SQL*Plus is also a tool, it's the foundational tool upon which all other integrated development environments (IDEs) are based. A solid understanding of Oracle basics and SQL*Plus lets you use IDE tools more effectively.

PL/SQL's History and Background

This is the short version of how Oracle Corporation came to exist in its present form. In the 1970s, Larry Ellison recognized a business opportunity in the idea of relational database management systems (RDBMSs). Along with a few friends, Ellison formed the company Software Development Laboratories (SDL) in 1977. A few years later, the founders changed the company name to Relational Software, Inc. (RSI), and subsequently changed it first to Oracle Systems Corporation and finally to Oracle Corporation. Through a combination of its own internal development and the acquisition of multiple companies over the past three and a half decades, Oracle, as it is commonly called, has captured the majority of the RDBMS market.

The concept of an RDBMS is complex. More or less, the idea is to (a) store information about how data is stored or structured, (b) store the data, and (c) access and manage both the structure and data through a common language. Structured Query Language (SQL) is that language (pronounced "sequel" in this book).

Oracle innovated beyond the original specification of SQL and created its own SQL dialect, Procedural Language/Structured Query Language (PL/SQL). While many of the new features of PL/SQL were adopted by the ANSI 92 SQL standard, some remain proprietary to Oracle Database. Those proprietary features give Oracle a competitive edge. Unlike some companies, Oracle isn't content to simply be the leader. It maintains its lead and competitive edge because it continues to innovate. Likewise, Oracle currently sets the industry standard for relational and object-relational databases.

Oracle created PL/SQL in the late 1980s, recognizing the need for a procedural extension to SQL. PL/SQL was and remains an innovative imperative language that supports both event-driven and object-oriented programming. Perhaps the most important aspect of PL/SQL is that you can call SQL statements from inside it, and call PL/SQL from SQL. People still shy away from PL/SQL because they want to remain *database agnostic*, which is a fancy way to say they want SQL solutions that are easily portable to other platforms. Although major competitors have added stored procedures to their competing database products, they've failed to deliver the same power and capability of PL/SQL. The single exception is IBM, which simply implemented PL/SQL very

similarly to how it works in Oracle Database. Unfortunately for IBM, the collections of Oracle SQL and PL/SQL built-ins and proprietary SQL extensions leave Oracle in the RDBMS technology lead.

In the late 1990s, Oracle saw the need for an object-relational extension to SQL. In response, it introduced object types in the Oracle 8 database and transformed the Oracle database server. It went from a *relational* database management system (RDBMS) to an *object-relational* database management system (ORDBMS). Oracle continued to improve how object types work in the Oracle 8*i*, 9*i*, 10*g*, 11*g*, and 12*c* releases. PL/SQL is a natural gateway to both creating and using these object types. Clearly, PL/SQL enabled the deployment and evolution of object-relational technologies in the Oracle database.

NOTE
The term object-relational model is interchangeable with the term extended-relational model, but Oracle prefers the former term over the latter.

Oracle also recognized, in 1998, the importance of implementing a Java Virtual Machine (JVM) inside the database. Oracle introduced a JVM in Oracle 9*i* Database. Oracle made improvements in the implementation of the JVM in Oracle Database 10*g*, 11*g*, and 12*c*. PL/SQL interfaces are used to access internal Java libraries that run in the JVM and to access external libraries in C-callable languages. The full functionality of an ORDBMS is possible because of the powerful combination of PL/SQL and an embedded JVM. In fact, PL/SQL has made possible the object-relational model we know as the Oracle database.

Figure 1-1 shows a timeline that covers the evolution of PL/SQL in the Oracle database. Interestingly, Oracle has provided 12 major feature upgrades during the 28-year history of the language. You'll note that Pascal is all but dead and gone, and Ada has had only four upgrades in the past 30+ years. The only language other than PL/SQL showing such feature investment is Java, which Oracle now owns.

From my years of experience with the product and other databases, I conclude that Oracle made the right call by adding PL/SQL to the Oracle database. PL/SQL is an extremely valuable and powerful tool for leveraging the database server. The ability to exploit the Oracle Database 12*c* server is critical to developing dynamic and effective database-centric applications.

Review Section

This section has presented the following details about the history and background of Oracle database:

- Oracle evolved from Relational Software, Inc. (RSI), which evolved from Software Development Laboratories (SDL).
- The SQL language is the interface to the Oracle Database 12*c* database engine, and Oracle extensions provide a competitive advantage.
- The PL/SQL language extends the behavior of SQL and has enabled the evolution of object-relational technologies.
- PL/SQL wraps access to embedded Java libraries.
- PL/SQL makes possible the implementation of an object-relational Oracle database.
- PL/SQL enables developers to exploit the Oracle Database 12*c* server.

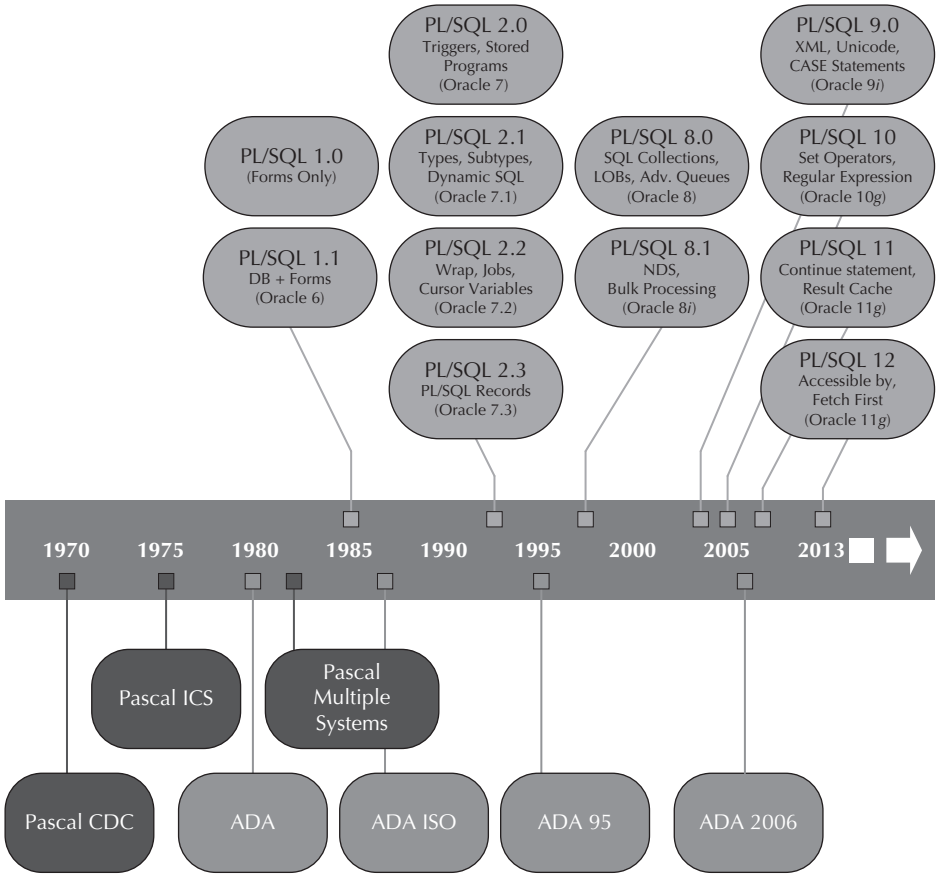


FIGURE 1-1. *PL/SQL language timeline*

Oracle Development Architecture

The architecture of a database has many levels. At the core of Oracle Database 12*c* is SQL. SQL is the interface to the Oracle Database 12*c* database engine, analogous to the steering wheel, brakes, and dashboard of a car. Analogous to the car engine is the server software, which includes an "engine" that stores and processes data, a "transmission" that governs transactions, and an enhanced "odometer" that logs what the system does to files. The server software also includes support programs that manage the system's content integrity, which are analogous to tires, body components, seat cushions, and bumpers. You can find much more about the Oracle Database 12*c* architecture in Appendix A.

Good mechanics must be aware of all the components that make up the vehicle. Likewise, database administrators (DBAs) must be aware of the many components related to an Oracle database system. The DBA is the primary mechanic who works with the engine, with the database

developer getting involved from time to time. The rest of the time, the database developer drives the car, which means they focus mostly on the data and working with SQL.

Just as a mechanic maintains and tunes a car's engine to optimize its performance and longevity, a DBA works with the numerous details of the database engine to get the most value from an Oracle database. Many of the details related to RDBMS management don't involve developers. That's because developers focus on interacting with the data, much like how a racecar driver who reports performance problems as they arise. While developers do worry about performance, they often defer resolution of performance problems to DBAs. Those developers who don't take the work to the mechanic (DBA) all the time often cross the line between the DBA and developer roles. By crossing that line, developers often learn new diagnostic skills. Developers who frequently cross that line between DBA and developer roles often become known as *application DBAs*.

Driving a car requires skill handling the steering wheel, accelerator, and brakes, and driving the "Oracle car" requires skill with SQL statements. While some SQL statements let you build database instances (cars), like a factory, others let you maintain, repair, and tune the database. Still other SQL statements let you interact with data, allowing you to insert, update, delete, and query database data. SQL statements that let you interact with data are sometimes called *CRUD* functions, representing create, read, update, and delete (check Appendix B for more details).

Developers who drive the Oracle car often work on small to medium-sized projects, and they're only exposed to the necessary tables, views, and stored programs that support a business application. Application developers like this only work with a small subset of the SQL interface, similar to how drivers of real cars focus on the steering wheel, accelerator, brakes, and fuel gauge.

The following section explains how DBAs can use PL/SQL to maintain and tune the engine and how developers can use PL/SQL to optimize performance. While the details of how you maintain and tune the engine are interesting on their own, this book is targeted at showing you how to use SQL and PL/SQL to solve database-centric application programming problems.

NOTE
*Appendix A describes the database environment, the database components, and the primary interface points—the SQL*Plus command-line interface (CLI) and the Oracle SQL Developer graphical user interface (GUI). Appendix B describes Oracle's implementation of SQL, which is the most complete in the industry.*

Before I explain how to drive the "Oracle car," I need to give you a quick tour of the engine that runs the car. First, you need to understand some terminology if you're new to the Oracle database. Second, the same SQL that manufactures the database lets you "drive" the database. Likewise, SQL actually runs beneath the wizards that Oracle provides.

The Database

An Oracle database is composed of a series of files, a set of processes, and a single database catalog. You create a database by using a tool, such as the Oracle Database Configuration Assistant (whose executable name is `dbca` in all operating systems). The Database Configuration Assistant is one of the programs that you install on the server tier when you install the Oracle product. Collectively, these programs are called a relational database management system (RDBMS). The Database Configuration Assistant is a wizard that simplifies how you create an Oracle database.

When you use a wizard to create a database, it creates the necessary files, processes, and database catalog. The database catalog is a set of tables that knows everything about the structures and algorithms of the database. You probably have heard the database catalog or dictionary called *metadata*, or data about data. Metadata is nothing more than a bunch of tables that define what you can store, manipulate, and access in a database. An Oracle database is also known as a *database instance*. More or less, the RDBMS creates databases like a factory creates cars. Oracle Database 12*c* can create more than one database instance on any server, provided the server has enough memory and disk space. With Oracle Database 12*c*'s *multitenant architecture*, you also have the ability to create *container databases (CDBs)* and *pluggable databases (PDBs)*.

The easiest analogy for an RDBMS would be a word-processing program, such as Microsoft Word, Corel WordPerfect, or Apple Pages. After installing one of these programs on your computer, it becomes a factory that lets you produce documents. Another name that might fit these programs is document management system (DMS), but they're not quite that capable. In short, they provide a user interface (UI) that lets you create and edit documents. This UI is like the steering wheel, accelerator, brakes, and dashboard that enable you to drive a car, but the dashboard probably promotes the UI to a graphical user interface (GUI).

Oracle also provides you with a UI, known as SQL*Plus. Oracle actually originally called its SQL*Plus command-line interface the *Advanced Friendly Interface (AFI)*, as still evidenced by the default temporary file buffer, `afiedt.buf`. As an experienced user, I can testify that it isn't that advanced by today's standards, nor is it that friendly. At least that's true until you try the CLIs of MySQL and SQL Server. After using either, you'd probably conclude, as I have, that SQL*Plus is both advanced and friendly by comparison.

The CLI is the basic UI, but most users adopt GUI tools, such as Dell's Toad (expensive) or Oracle SQL Developer (free). (Appendix A provides guidance on installing, configuring, and working with SQL*Plus and SQL Developer.) Neither the SQL*Plus CLI nor the SQL Developer GUI is difficult to use once you understand the basics of how connections work (also covered in

Multitenant Architecture

Oracle Database 12*c* introduces the multitenant architecture, which is like an apartment complex for Oracle Database instances. While apartment complexes can be located in a single building or in multiple buildings, they generally have one location that manages the complex. Very large apartment complexes may have a centralized management office and local management offices in each of the buildings.

Oracle's multitentant architecture isn't too different from a large, multiple-building apartment complex with a centralized management office. The container database (CDB) is the centralized management office, and pluggable databases (PDBs) are the apartment buildings with local management offices.

Like an apartment complex's centralized management office, the CDB holds the master `sys` and `system` schemas. Individual PDBs hold an `ADMIN` user that enjoys `sysdba` privileges for the PDB, like the `sys` schema does in the CDB. PDBs also hold a `system` schema that works discretely with an individual PDB. The local PDB `ADMIN` user's `sys` and `system` schemas are like the local building manager in a very large apartment complex. Appendix A describes how you configure a PDB.

Appendix A). You need to understand how to use at least one of these tools to operate the Oracle database more effectively.

The command line is an essential tool when you write production code. Production code must be rerunnable, which means you can run the command when it has already been run before. To make production code rerunnable, you package together a set of related SQL and/or PL/SQL commands that you previously typed into a console interactively, and then put them into a file. The file, also known as a *script file*, could, for instance, drop a table before trying to re-create it. Dropping the table before re-creating it differently avoids an `ORA-00955` error, which tells you that you're trying to reuse a name already stored in the data catalog.

You run the script file from the command line, or from another script that calls scripts, which is why I'll show you how to use the command line in the "Two-Tier Model" section later in the chapter.

The PL/SQL Language

The PL/SQL language is a robust tool with many options. PL/SQL lets you write code once and deploy it in the database nearest the data. PL/SQL can simplify application development, optimize execution, and improve resource utilization in the database. PL/SQL isn't a replacement for SQL, which is a set-based declarative language that lets you interact with data and the database. As mentioned, PL/SQL is a powerful imperative language with both event-driven and object-oriented features.

Is PL/SQL Programming a Black Art?

Early on, PL/SQL 1.0 was little more than a reporting tool. Now the `CASE` statement in SQL delivers most of that original functionality. In the mid-1990s, developers described PL/SQL 2.*x* programming as a "black art." This label was appropriate then: there was little written about the language, and the availability of code samples on the Web was limited because the Web didn't really exist as you know it today.

Today, there are still some who see PL/SQL as a black art. They also are passionate about writing database-neutral code in Java or other languages. This is *politically correct speak for avoiding PL/SQL solutions* notwithstanding their advantages. Why is Oracle PL/SQL still considered a black art to these people when there are so many PL/SQL books published today?

Perhaps the reason is the cursors, but the cursors exist in any program that connects through the Oracle Call Interface (OCI) or Java Database Connectivity (JDBC). If not cursors, perhaps it's the syntax, user-defined types, or nuances of functions and procedures. Are those really that much different from their equivalents in other programming languages? If you answer "no" to this question, you've been initiated into the world of PL/SQL. If you answer "yes" or think there's some other magic to the language, you haven't been initiated.

How do you become initiated? The cute answer is to read this book. The real answer is to disambiguate the Oracle jargon that shrouds the PL/SQL language. For example, a variable is always a variable of some type, and a function or procedure is always a subroutine that manages formal parameters by reference or by value and the subroutine may or may not return a result as a right operand. These types of simple rules hold true for every component in the PL/SQL language.

The language is a case-insensitive programming language, like SQL. This has led to numerous formatting best practice directions. Rather than repeat those arguments for one style or another, it seems best to recommend that you find a style consistent with your organization's standards and consistently apply it. *The PL/SQL code in this book uses all uppercase letters for command words and all lowercase letters for variables, column names, and stored program calls*.

PL/SQL was developed by modeling concepts of structured programming, static data typing, modularity, exception management, and parallel (concurrent) processing found in the Ada programming language. The Ada programming language, developed for the United States Department of Defense, was designed to support military real-time and safety-critical embedded systems, such as those in airplanes and missiles. The Ada programming language borrowed significant syntax from the Pascal programming language, including the assignment and comparison operators and the single-quote delimiters.

These choices also enabled the direct inclusion of SQL statements in PL/SQL code blocks. They were important because SQL adopted the same Pascal operators, string delimiters, and declarative scalar data types. Both Pascal and Ada have declarative scalar data types. Declarative data types do not change at runtime and are known as *strong* data types. Strong data types are critical to tightly integrating the Oracle SQL and PL/SQL languages. PL/SQL supports dynamic data types by mapping them at runtime against types defined in the Oracle Database 12c database catalog. Matching operators and string delimiters means simplified parsing because SQL statements are natively embedded in PL/SQL programming units.

NOTE
Primitives in the Java programming language describe scalar variables, which hold only one thing at a time.

The original PL/SQL development team made these choices carefully. The Oracle database has been rewarded over the years because of those choices. One choice that stands out as an awesome decision is letting you link PL/SQL variables to the database catalog or cursor. This is a form of runtime type inheritance, and is best implemented when you inherit from a cursor rather than from a table or column.

You use the `%TYPE` and `%ROWTYPE` pseudo types to inherit from the strongly typed variables defined in the database catalog. Oracle calls this type of inheritance *anchoring*, and you can read a complete treatment in the "Attribute and Table Anchoring" section of Chapter 3.

Anchoring PL/SQL variables to database catalog objects is an effective form of structural coupling. It can minimize the number of changes you need to make to your PL/SQL programs. At least, it limits how often you recode when a table's column changes size. However, structural coupling like this is expensive because it causes context switches inside the database server.

Oracle also made another strategic decision when it limited the number of SQL base types and allowed users to subtype base types in the database catalog, enabling them to create a multiple-hierarchy object tree. This type of object tree can continue to grow and mature over time. These types of changes increase the object-oriented features of the Oracle database.

The PL/SQL runtime engine exists as a resource inside the SQL*Plus environment. The SQL*Plus environment has both an interactive mode and a callable server mode. Every time you connect to the Oracle Database 12c database, the database creates a new session. Calls from the server's CLI or a remote client's CLI may open an interactive session, while calls from external programs open a server mode session. In either type of session, you can run SQL or PL/SQL statements from the SQL*Plus environment. PL/SQL program units can then run SQL statements

or external procedures, as shown in Figure 1-2. SQL statements may also call PL/SQL stored functions or procedures. SQL statements interact directly with the actual data.

Calls directly to PL/SQL can be made through the Oracle Call Interface (OCI) or Java Database Connectivity (JDBC). This lets you leverage PL/SQL directly in your database applications. This is important because it lets you manage transaction scope in your stored PL/SQL program units. This tremendously simplifies the myriad tasks often placed in the data abstraction layer of applications.

PL/SQL also supports building SQL statements at runtime. Runtime SQL statements are dynamic SQL. You can use two approaches for dynamic SQL: one is Native Dynamic SQL (NDS), and the other is the `DBMS_SQL` package. Chapter 13 demonstrates dynamic SQL and covers both NDS and the `DBMS_SQL` package.

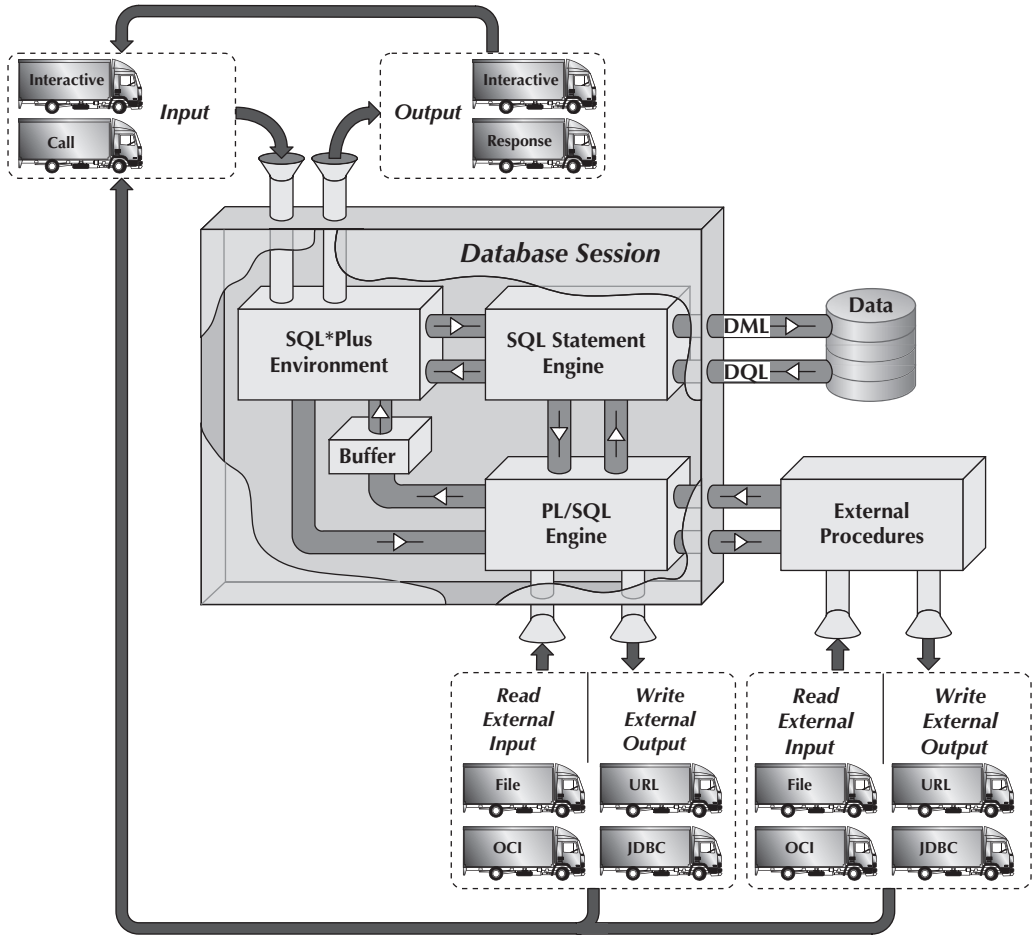


FIGURE 1-2. *Database processing architecture*

You now have a high-level view of the PL/SQL language. Chapter 3 provides an overview of PL/SQL block structures and programming basics.

The Oracle Processing Architecture

Figure 1-2 shows the Oracle processing architecture, or how you "operate the car." Notice that all input goes in through the SQL*Plus environment and all results or notifications return through the same environment. That means you're interfacing with the SQL*Plus CLI when you're working in the SQL Developer GUI, or through an external programming language such as PHP or Java. The only difference between external programming languages and PL/SQL is that you lose access to the interactive features of SQL*Plus when working through external calls in PHP or Java. You access the *call mode* of SQL*Plus when you call it through the Open Database Connectivity (ODBC) interface or JDBC interface.

As you can see in Figure 1-2, PL/SQL serves as the interface between the database and internally deployed Java libraries, file I/O (input/output) operations, and external procedures. SQL is the only point of access to the physical data, and as such it serves as an "automatic transmission" to the many processes that keep the Oracle Database 12c database running smoothly.

As covered in Appendix B, the SQL statement engine processes *all* types of SQL statements, which includes the following:

- **Data Definition Language (DDL) statements** `CREATE`, `ALTER`, `DROP`, `RENAME`, `TRUNCATE`, and `COMMENT`. They allow you to create, alter, drop, rename, truncate, and comment tables and other objects.
- **Data Manipulation Language (DML) statements** `SELECT`, `INSERT`, `UPDATE`, `DELETE`, and `MERGE`. They let you query, insert, change, and merge data in the database and remove data from the database.
- **Data Control Language (DCL) statements** `GRANT` and `REVOKE`. They let you grant and revoke privileges and groups of privileges (known as *roles*).
- **Transaction Control Language (TCL) statements** `COMMIT`, `ROLLBACK`, and `SAVEPOINT`. They let you control when to make data permanent or undo temporary changes. They enable you to control all-or-nothing behavior that's *ACID compliant* (check Appendix A for the details).

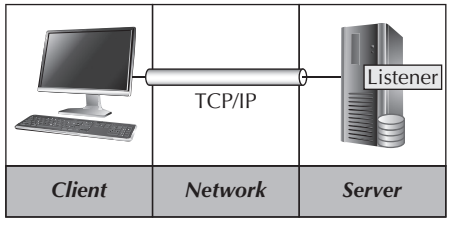
A SQL statement can call a named PL/SQL program unit, and a PL/SQL block can call a SQL statement. A named PL/SQL program unit is a function or procedure stored in the database catalog. A PL/SQL call to a SQL statement can only include SQL data types and named PL/SQL program units stored in the database catalog. That means it can't call a locally defined function inside a SQL statement. Procedures can't be called inside a SQL statement directly; they must be contained inside a stored function. The reason that you can't call a local function inside a SQL statement is that the SQL engine doesn't have access to a local function.

A complete book would be required to cover all the features in the Oracle SQL implementation, but Appendix B certainly exposes the majority of core features that any reader will use to develop applications or administer a database. SQL is like the automatic transmission to all the complex engine parts that run the Oracle database. Beyond an introduction to SQL, Appendix C covers SQL built-in functions and Appendix D covers PL/SQL built-in packages.

The next two sections discuss the connection mechanism for Oracle databases. The basics of the *two-tier* computing model are described first, followed by a discussion of the more complex *three-tier* model, which is really an *n*-tier model. Understanding these models is essential to understanding how you can use SQL or PL/SQL.

Two-Tier Model

All databases adopt a two-tier model: the engine and the interface. The server is the database engine and a database listener. A *listener* implements the object-oriented analysis and design *observer* pattern. The observer pattern is mainly used to implement distributed event-handling systems. Oracle's listener is a program that listens for incoming requests on an ephemeral (or short-lived) port, and forwards them to a SQL*Plus session. The client is the interface that lets you issue SQL commands and, in Oracle, lets you call PL/SQL blocks.



A typical installation of the Oracle database installs both the client and the server on the database server. That's because the mechanic (or DBA) who maintains the engine uses the same interface to manage many of the parts. Other server-side utilities let the DBA manage part replacement when the server is shut down. (Similar to how you'd replace parts in an engine, you'd shut off the engine before taking it apart to replace something.)

Our focus in this book is the interface to the running engine. We use the database server copy of the client software when we drive the database from the local server. Sometimes we want to drive the database remotely from a laptop. We have several ways to accomplish that process. One is to install a copy of the Oracle Client software on a remote machine. Another is to use a tool, such as SQL Developer, to connect across the network.

N-Tier Model

All databases support a three-tier model, because it's really just a middleware solution. As you can see in Figure 1-3, the middle tier of a three-tier model may have many moving parts, and they work like tiers. That's why the industry adopted the *n*-tier model over the original three-tier model. An *n*-tier model more aptly describes what's actually happening in web-based applications. The middleware

- Can have a multithreaded JServlet, Apache module, or general software appliance
- Can have a metric server layer to balance load across multiple devices
- Creates a pool of connections to the Oracle database and shares the connections with requests made by other clients

Typically in an *n*-tier model, the client-to-middleware communication doesn't enjoy a state-aware connection (see Figure 1-3). In fact, it's often stateless through the HTTP/HTTPS protocols.

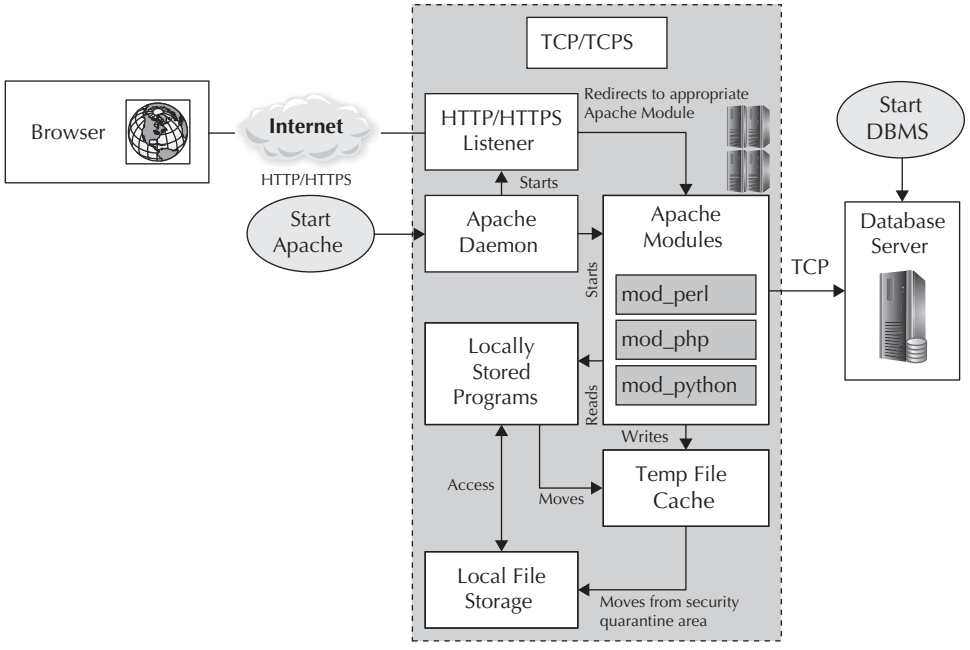


FIGURE 1-3. *N-tier computing model*

This shift in communication semantics means changes are automatic and permanent when they occur. If you submit a data change via an `INSERT`, `UPDATE`, or `DELETE` statement across HTTP/HTTPS and receive acknowledgement of success, that change is permanent. This is known as an *optimistic processing model*. It alone is a reason for stored procedures that manage transactions across multiple tables in any database.

The exception to an optimistic process occurs when the middleware maintains a lock on the data and manages your transaction scope for you. This type of implementation is done for you by default in Oracle Enterprise Manager (OEM) or Oracle Application Express (APEX). Describing the mechanics of how this works would require a chapter of its own. Suffice it to say, this is a possible architecture for your internally developed applications.

Review Section

This section has described the following points about Oracle database architecture:

- SQL is the interface that lets you manage, maintain, and use the Oracle Database 12c database engine.
- Oracle provides a SQL*Plus CLI and several GUIs that all interact with SQL and PL/SQL.

- The SQL language is the "automatic transmission" to the data and many processes that keep the Oracle Database 12c database running smoothly. SQL replaces imperative languages as the interface to relational data and RDBMS management.
- The two-tier model represents how SQL works with the data, with the SQL*Plus CLI or SQL GUI acting as the client and the database engine acting as the server.
- The *n*-tier model represents how web-based applications engage the data through a middle tier, which can be three or more tiers in depth.

Summary

This chapter has provided a tour of the Oracle development environment for client- and server-side PL/SQL development. In conjunction with Appendixes A and B, you should be positioned to understand, work with, and experiment with the examples in the subsequent chapters.

Mastery Check

The mastery check is a series of true-or-false and multiple-choice questions that let you confirm how well you understand the material in the chapter. You may check Appendix I for answers to these questions.

True or False:

1. ___Relational Software, Inc. became Oracle Corporation.
2. ___Relational databases store information about how data is stored.
3. ___Relational databases store data.
4. ___SQL is an imperative language that lets you work in the Oracle database.
5. ___The relational database model evolved from the object-relational database model.
6. ___PL/SQL is the procedural extension of SQL.
7. ___PL/SQL is an imperative language that is both event-driven and object-oriented.
8. ___The Oracle database relies on an external Java Virtual Machine to run stored Java libraries.
9. ___A two-tier model works between a browser and a database server.
10. ___A three-tier model is a specialized form of an *n*-tier model.

Multiple Choice:

11. Which of the following describes the roles of the Oracle listener? (Multiple answers possible)
    - **A.** Listen for incoming client requests
    - **B.** Send outgoing requests to client software
    - **C.** Forward requests to the PL/SQL engine
    - **D.** Forward requests to a SQL*Plus session
    - **E.** Forward requests to the SQL engine

12. Which of the following converts a relational model to an object-relational model? (Multiple answers possible)

- **A.** A data catalog
- **B.** A set of tables
- **C.** An object data type
- **D.** An imperative language that lets you build native object types
- **E.** A JVM inside the database

13. SQL*Plus provides which of the following? (Multiple answers possible)

- **A.** An interactive mode
- **B.** A call mode
- **C.** A server mode
- **D.** A client mode
- **E.** All of the above

14. Which of the following is a capability of PL/SQL?

- **A.** Call SQL
- **B.** Implement object types
- **C.** Wrap C-callable programs
- **D.** Wrap Java programs
- **E.** All of the above

15. Which of the following are types of SQL statements? (Multiple answers possible)

- **A.** Data Definition Language (DDL) statements
- **B.** Data Manipulation Language (DML) statements
- **C.** Data Control Language (DCL) statements
- **D.** Create, replace, update, and delete (CRUD) statements
- **E.** Transaction Control Language (TCL) statements