

This course covers PL/SQL, a procedural language extension to SQL. Through an innovative project-based approach, students learn procedural logic constructs such as variables, constants, conditional statements and iterative controls. Students have the opportunity to sit for the second of two exams required to earn the Oracle Certified Associate.

Section 1

This section provides an introduction to PL/SQL and explains the difference between SQL and the PL/SQL block structure. Students explore some of the limitations of SQL and learn why PL/SQL is needed. PL/SQL key words and definitions and appropriate usages are introduced. The students learn the characteristics of the PL/SQL programming language including how it compares to other programming languages like C and Java. Finally, students learn about the PL/SQL block structure, the basic unit in PL/SQL. They begin coding anonymous blocks in Oracle Application Express, a browser-based development environment.

Lesson 1: Introduction to PL/SQL

- Describe PL/SQL.
- Differentiate between SQL and PL/SQL.
- Explain the need for PL/SQL.

Lesson 2: Benefits of PL/SQL

- List and explain the benefits of PL/SQL.
- List differences between PL/SQL and other programming languages.
- Give examples of how PL/SQL can be used in other Oracle products.

Lesson 3: Creating PL/SQL Blocks

- Describe the structure of a PL/SQL block.
- Identify the different types of PL/SQL blocks.
- Identify PL/SQL programming environments.
- Create and execute an anonymous PL/SQL block.
- Output messages in PL/SQL.

Lesson 4: Review of SQL SELECT Statements

- Create a basic SQL statement including ORDER BY.
- Perform and display arithmetic calculations.
- Construct a query using a column alias.
- Apply the concatenation operator.
- Use literal values in a select statement.
- Use DISTINCT syntax in a query to eliminate duplicate rows.
- Use conditional syntax including BETWEEN, IN, and LIKE, in a query.

Lesson 5: Review of SQL Single-Row Functions

- Select and apply single-row functions to change the case of character data.
- Select and apply the CONCAT, SUBSTR and LENGTH functions to manipulate character data.
- Select and apply single-row functions to round or truncate numerical data.
- Select and apply single-row functions to convert data stored as one data type to another data type.
- Select and apply single-row functions to perform month-level arithmetic.
- Select and apply single-row functions to enhance query results containing null values.

Section 2

Section two explains how PL/SQL extends SQL by adding program structures and subroutines. This section introduces students to the syntax, vocabulary, and parts of speech or lexical units of the PL/SQL programming language. Students learn to define variables for storing and manipulating data. They learn about lexical units such as reserved words, delimiters, and literals. The students also learn about data types supported by PL/SQL such as integer, floating point, character, Boolean, date, collection, and LOB. After a brief review of SQL functions, students learn how to incorporate SQL functions in PL/SQL statements and how it is occasionally necessary to explicitly convert data types. Section 2 also expands on simple block structure with an exploration of scope and nesting blocks. It also covers the use of variables and data types and explains the scope of nested blocks.

Lesson 1: Using Variables in PL/SQL

- List the uses of variables in PL/SQL.
- Identify the syntax for variables in PL/SQL.
- Declare and initialize variables in PL/SQL.
- Assign new values to variables in PL/SQL.

Lesson 2: Recognizing PL/SQL Lexical Units

- List and define the different types of lexical units available in PL/SQL.
- Describe identifiers and identify valid and invalid identifiers in PL/SQL.
- Describe and identify reserved words, delimiters, literals, and comments in PL/SQL.

Lesson 3: Recognizing Data Types

- Define data type and explain why it is needed.
- List and describe categories of data types.
- Give examples of scalar, composite, and large object (LOB) data types.

Lesson 4: Using Scalar Data Types

- Declare and use scalar data types in PL/SQL.
- Define guidelines for declaring and initializing PL/SQL variables.
- Identify the benefits of anchoring data types with the %TYPE attribute.

Lesson 5: Review of SQL Joins

- Construct and execute SELECT statements to access data from more than one table using an equijoin.
- Construct and execute SELECT statements to access data from more than one table using a nonequijoin.
- Construct and execute SELECT statements to access data from more than one table using an outer join.
- Construct and execute SELECT statements that result in a Cartesian product.

Lesson 6: Review of SQL Group Functions and Subqueries

- Construct and execute a SQL query that utilizes group functions to determine a sum total, an average amount, and a maximum value.
- Construct and execute a SQL Query that groups data based on specified criteria.
- Construct and execute a query that contains a WHERE clause using a single-row subquery.
- Construct and execute a query that contains a WHERE clause using a multiple-row subquery.

Lesson 7: Writing PL/SQL Executable Statements

- Construct accurate variable assignment statements in PL/SQL.
- Construct accurate statements using built-in SQL functions in PL/SQL.
- Differentiate between implicit and explicit conversions of data types.
- Describe when implicit conversions of data types take place.
- List drawbacks of implicit data type conversions.
- Construct accurate statements using functions to explicitly convert data types.
- Construct statements using operators in PL/SQL.

Lesson 8: Nested Blocks and Variable Scope

- Understand the scope and visibility of variables.
- Write nested blocks and qualify variables with labels.
- Understand the scope of an exception.
- Describe the effect of exception propagation in nested blocks.

Lesson 9: Good Programming Practices

- List examples of good programming practices.
- Accurately insert comments into PL/SQL code.
- Create PL/SQL code that follows formatting guidelines to produce readable code.

Section 3

Section 3 begins with a brief review of Data Manipulation Language (DML), that is, SQL commands that allow for the manipulation of data in the database. Students learn to embed standard DML statements such as SELECT, INSERT, DELETE, and UPDATE in PL/SQL blocks. They also learn about the implicit cursors that are created when executing SQL statements in PL/SQL as well as the cursor attributes that allow them to test the outcome of these statements. Finally, students learn about database transactions and how group together SQL statements into logical units using COMMIT, ROLLBACK, and SAVEPOINT to make changes to the database permanent or to discard them.

Lesson 1: Review of SQL DML

- Construct and execute a DML statement to insert data into a database table.
- Construct and execute a DML statement to update data in a database table.
- Construct and execute a DML statement to delete data from a database table.
- Construct and execute a DML statement to merge data into a database table.

Lesson 2: Retrieving Data in PL/SQL

- Recognize the SQL statements that can be directly included in a PL/SQL executable block.
- Construct and execute an INTO clause to hold the values returned by a single-row SQL SELECT statement.
- Construct statements to retrieve data that follow good practice guidelines.
- Construct statements that apply good practice guidelines for naming variables.

Lesson 3: Manipulating Data in PL/SQL

- Construct and execute PL/SQL statements that manipulate data with DML statements.
- Describe when to use implicit or explicit cursors in PL/SQL.
- Create PL/SQL code to use SQL implicit cursor attributes to evaluate cursor activity.

Lesson 4: Using Transaction Control Statements

- Define a transaction and give an example.
- Construct and execute a transaction control statement in PL/SQL.

Section 4

Conditional control allows control of the flow of program execution based on a condition. Program structures such as conditional control statements and iterative control statements control the flow of execution through a program. Conditional control is the ability to direct the flow of execution through a program based on a condition. Iterative control is the ability to execute a sequence of statements multiple times until a certain condition is met. In this section the students will explore conditional control structures and gain an understanding on how these structures can be nested one inside of another. This section also covers the different kinds of conditional and iterative control statements available in PL/SQL. In terms of conditional control, students learn the syntax and usage of IF-THEN-ELSE and CASE statements. In terms of iterative control statements, students learn the purpose of and usage guidelines for simple loops, WHILE loops, and FOR loops. Students will also gain an understanding on how these structures can be nested one inside of another.

Lesson 1: Conditional Control: IF Statements

- List the types of conditional control structures.
- Construct and use an IF statement.
- Construct and use an IF-THEN-ELSIF-ELSE statement.
- Create PL/SQL to handle the null condition in an IF statement.

Lesson 2: Conditional Control: Case Statements

- Construct and use CASE statements in PL/SQL.
- Construct and use CASE expressions in PL/SQL.
- Include the correct syntax to handle null conditions in PL/SQL CASE statements.
- Include the correct syntax to handle Boolean conditions in PL/SQL IF and CASE statements.

Lesson 3: Iterative Control: Basic Loops

- Describe the need for LOOP statements in PL/SQL.
- Recognize different types of LOOP Statements.
- Create PL/SQL containing a basic loop and an EXIT statement.
- Create PL/SQL containing a basic loop and an EXIT statement with conditional termination.

Lesson 4: Iterative Control: While and For Loops

- Construct and use the WHILE looping construct in PL/SQL.
- Construct and use the FOR looping construct in PL/SQL.
- Describe when a WHILE loop is used in PL/SQL.
- Describe when a FOR loop is used in PL/SQL.

Lesson 5: Iterative Control: Nested Loops

- Construct and execute PL/SQL using nested loops.
- Evaluate a nested loop construct and identify the exit point.

Section 5

In Section 5 students are introduced to explicit cursors, that is, cursors that are declared by the programmer. Students learn that cursors can be used to select multiple rows from a database table allowing the programming code to process rows one at a time. In addition to learning about the different syntactical rules for creating cursors, students also learn what a parameter is and how to pass parameters into a cursor.

Lesson 1: Introduction to Explicit Cursors

- Distinguish between an implicit and an explicit cursor.
- Describe why and when to use an explicit cursor in PL/SQL code.
- List two or more guidelines for declaring and controlling explicit cursors.
- Create PL/SQL code that successfully opens a cursor and fetches a piece of data into a variable.
- Use a simple loop to fetch multiple rows from a cursor.
- Create PL/SQL code that successfully closes a cursor after fetching data into a variable.

Lesson 2: Using Explicit Cursor Attributes

- Define a record structure using the %ROWTYPE attribute.
- Create PL/SQL code to process the row of an active set using record types in cursors.
- Retrieve information about the state of an explicit cursor using cursor attributes.

Lesson 3: Cursor FOR Loops

- List and explain the benefits of using cursor FOR loops.
- Create PL/SQL code to declare a cursor and manipulate it in a FOR loop.
- Create PL/SQL code containing a cursor FOR loop using a subquery.

Lesson 4: Cursors with Parameters

- List the benefits of using parameters with cursors.
- Create PL/SQL code to declare and manipulate a cursor with a parameter.

Lesson 5: Using Cursors For Update

- Create PL/SQL code to lock rows before an update using the appropriate clause.
- Explain the effect of using NOWAIT in an update cursor declaration.
- Create PL/SQL code to use the current row of the cursor in an UPDATE or DELETE statement.

Lesson 6: Using Multiple Cursors

- Explain the need for using multiple cursors to produce multilevel reports.
- Create PL/SQL code to declare and manipulate multiple cursors within nested loops.
- Create PL/SQL code to declare and manipulate multiple cursors using parameters.

Section 6

Up to this point in the course, it is assumed that the code written by students works fine as long as it compiles correctly. However, when these compiled programs are executed, the code may cause some unanticipated errors. These run-time errors are also known as exceptions. In this section, students learn how to specify an exception handler so that the program does not terminate when PL/SQL raises an exception. Exception handlers make the program more robust by specifying the final actions to perform before the block ends. Students learn how to handle Oracle Server (built in) and user-defined exceptions. Students also revisit the scope of variables and use this knowledge to recognize the effects of exceptions, which are propagated within nested blocks of code.

Lesson 1: Handling Exceptions

- Describe several advantages of including exception handling code in PL/SQL.
- Describe the purpose of an EXCEPTION section in a PL/SQL block.
- Create PL/SQL code to include an EXCEPTION section.
- List several guidelines for exception handling.

Lesson 2: Trapping Oracle Server Exceptions

- Describe and provide an example of an error defined by the Oracle server.
- Describe and provide an example of an error defined by the PL/SQL programmer.
- Differentiate between errors that are handled implicitly and explicitly by the Oracle Server.
- Write PL/SQL code to trap a predefined Oracle Server error.
- Write PL/SQL code to trap a non-predefined Oracle Server error.
- Write PL/SQL code to identify an exception by error code and by error message.

Lesson 3: Trapping User-Defined Exceptions

- Write PL/SQL code to name a user-defined exception.
- Write PL/SQL code to raise an exception.
- Write PL/SQL code to handle a raised exception.
- Write PL/SQL code to use RAISE_APPLICATION_ERROR.

Lesson 4: Recognizing the Scope of Variables

- Describe the rules for variable scope when a variable is nested in a block.
- Recognize a variable scope issue when a variable is used in nested blocks.
- Qualify a variable nested in a block with a label.
- Describe the scope of an exception.
- Recognize an exception scope issue when an exception is within nested blocks.
- Describe the effect of exception propagation in nested blocks.

Section 7

Programs focused on completing specific tasks and functions can be modularized into procedures, PL/SQL blocks that are generally used to perform an action. Procedures, which are always named, can be stored in the database and called when needed for repeated execution. In this section, students create, invoke, and correct errors in procedures. Students learn to use simple parameters to pass information into a procedure. They then learn how PL/SQL procedures support different parameter modes that allow parameters to not only pass information into a procedure, but also to return information back to the calling PL/SQL block. Finally, students learn how to delete procedures and view them in the data dictionary.

Lesson 1: Creating Procedures

- Differentiate between anonymous blocks and subprograms.
- Identify benefits of subprograms.
- Define a stored procedure.
- Create a procedure.
- Describe how a stored procedure is invoked.
- List the development steps for creating a procedure.

Lesson 2: Using Parameters in Procedures

- Describe how parameters contribute to a procedure.
- Define a parameter.
- Create a procedure using a parameter.
- Invoke a procedure that has parameters.
- Differentiate between formal and actual parameters.

Lesson 3: Passing Parameters

- List the types of parameter modes.
- Create a procedure that passes parameters.
- Identify three methods for passing parameters.
- Describe the DEFAULT option for parameters.

Section 8

A function is a named PL/SQL block that can accept parameters, be invoked, and return a value. Functions are very much like procedures. Section 8 introduces user-defined functions, that is, functions that are created by a user and saved in the database as schema objects. Students learn how to create and invoke functions. They learn how to use functions in different parts of a SQL statement such as the SELECT list and the WHERE and HAVING clauses. Additionally, students learn how to delete functions and view them in the data dictionary. Finally, students learn the differences between definer's and invoker's rights.

Lesson 1: Creating Functions

- Define a stored function.
- Create a PL/SQL block containing a function.
- List ways in which a function can be invoked.
- Create a PL/SQL block that invokes a function that has parameters.
- List the development steps for creating a function.
- Describe the differences between procedures and functions.

Lesson 2: Using Functions in SQL Statements

- List the advantages of user-defined functions in SQL statements.
- List where user-defined functions can be called from within an SQL statement.
- Describe the restrictions on calling functions from SQL statements.

Lesson 3: Review of the Data Dictionary

- Describe the purposes of the Data Dictionary.
- Differentiate between the three types of Data Dictionary view.
- Write SQL SELECT statements to retrieve information from the Data Dictionary.
- Explain the use of DICTIONARY as a Data Dictionary search engine.

Lesson 4: Managing Procedures and Functions

- Describe how exceptions are propagated.
- Remove a function and a procedure.
- Use data dictionary views to identify and manage stored programs.

Lesson 5: Review of Object Privileges

- List and explain several object privileges.
- Explain the function of the EXECUTE object privilege.
- Write SQL statements to grant and revoke object privileges.

Lesson 6: Using Invoker's Rights

- Contrast invoker's rights with definer's rights.
- Create a procedure that uses invoker's rights.

Section 9

Section 9 introduces the PL/SQL package. Packages allow related PL/SQL types, variables, data structures, exceptions, and subprograms to be bundled into one container. Students are introduced to the different parts of a PL/SQL package—the specification and body—as well as the syntax for creating each. Students learn about the visibility of objects in packages and how to invoke subprograms in packages. After a brief discussion about managing packages, students are exposed to the more advanced features of PL/SQL, including overloading, and forward referencing. To extend the functionality of the database, Oracle includes many packages with the Oracle Server. Students gain experience using three of the more commonly used Oracle server-supplied packages. Finally, students learn to construct and execute SQL statements dynamically (at run time) using the Native Dynamic SQL statements in PL/SQL.

Lesson 1: Creating Packages

- Describe the reasons for using a package.
- Describe the two components of a package: specification and body.
- Create packages containing related variables, cursors, constants, exceptions, procedures, and functions.
- Create a PL/SQL block that invokes a package construct.

Lesson 2: Managing Package Concepts

- Explain the difference between public and private package constructs.
- Designate a package construct as either public or private.
- Specify the appropriate syntax to drop packages.
- Identify views in the data dictionary that manage packages.
- Identify guidelines for using packages.

Lesson 3: Advanced Package Concepts

- Write packages that use the overloading feature.
- Write packages that use forward declarations.
- Explain the purpose of a package initialization block.
- Identify restrictions on using packaged functions in SQL statements.

Lesson 4: Persistent State of Package Variables

- Identify persistent states of package variables.
- Control the persistent state of a package cursor.

Lesson 5: Using Oracle-Supplied Packages

- Describe two common uses for the DBMS_OUTPUT server-supplied package.
- Recognize the correct syntax to specify messages for the DBMS_OUTPUT package.
- Describe the purpose for the UTL_FILE server-supplied package.
- Recall the exceptions used in conjunction with the UTL_FILE server-supplied package.

Lesson 6: Dynamic SQL

- Recall the stages through which all SQL statements pass.
- Describe the reasons for using dynamic SQL to create a SQL statement.
- List four PL/SQL statements supporting Native Dynamic SQL.
- Describe the benefits of Execute Immediate over DBMS_SQL for Dynamic SQL.

Section 10

Triggers are another type of named PL/SQL block or procedure. Triggers are associated with a table, view, schema, or database. Triggers execute implicitly whenever a particular DML or DDL event takes place. Students learn about the different types of triggers, the events that cause the trigger to fire, and the steps required to create them. Finally, students learn how to manage triggers by viewing them in the data dictionary, altering their status, and deleting them.

Lesson 1: Introduction to Triggers

- Describe database triggers and their uses.
- Define a database trigger.
- Recognize the difference between a database trigger and an application trigger.
- List two or more guidelines for using triggers.
- Compare and contrast database triggers and stored procedures.

Lesson 2: Creating DML Triggers: Part I

- Create a DML trigger.
- List the DML trigger components.
- Create a statement level trigger.
- Describe the trigger firing sequence options.

Lesson 3: Creating DML Triggers: Part II

- Create a DML trigger that uses conditional predicates.
- Create a row level trigger.
- Create a row level trigger that uses OLD and NEW qualifiers.
- Create an INSTEAD OF trigger.

Lesson 4: Creating DDL and Database Event Triggers

- Describe events that cause DDL and database event triggers to fire.
- Create a trigger for a DDL statement.
- Create a trigger for a database event.
- Describe the functionality of the CALL statement.
- Describe the cause of a mutating table.

Lesson 5: Managing Triggers

- View trigger information in the Data Dictionary.
- Disable and enable a database trigger.
- Remove a trigger from the database.

Section 11

Section eleven expands upon the concept of composite data structures by describing the use of user-defined record types. These more complex data types include Large Objects (LOBs), BFILES, RECORDS, and TABLES. Students will learn how to use LOBs to store unstructured data such as graphic images, video clips, and sound wave forms in the database. They will also learn how to use BFILES to store large binary files outside the database, and how to use RECORDS, TABLES, and TABLES OF RECORDS to manipulate the individual components of composite data.

Lesson 1: Using Large Object Data Types

- Compare and contrast LONG and LOB (large object) data types.
- Describe LOB data types and how they are used.
- Differentiate between internal and external LOBs.
- Create and maintain LOB data types.
- Migrate data from LONG To LOB.

Lesson 2: Managing Bfiles

- Define BFILES and the BFILE column data type.
- Create directory objects and view them in the Data Dictionary.
- Manage and manipulate BFILES using BFILENAME and DBMS_LOB.

Lesson 3: User-Defined Records

- Create and manipulate user-defined PL/SQL records.

Lesson 4: Indexing Tables of Records

- Create an INDEX BY table.
- Create an INDEX BY table of records.
- Describe the difference between records, tables, and tables of records.

Section 12

In PL/SQL, some objects reference other objects as part of their definitions. If the definition of a referenced object is altered, the dependent object (or objects) may or may not continue to work properly. This is called a dependency. Section 12 teaches students about different kinds of dependencies, including how to identify them by running SQL scripts as well as how to fix them when necessary. Students will also learn how to write code so that dependency failures are unlikely.

Lesson 1: Understanding Dependencies

- Describe the implications of procedural dependencies.
- Contrast dependent objects and referenced objects.
- View dependency information in the data dictionary.
- Use the UTLDTREE script to create the objects required to display dependencies.
- Use the IDEPTREE and DEPTREE views to display dependencies.
- Describe when automatic recompilation occurs.
- List how to minimize dependency failures.