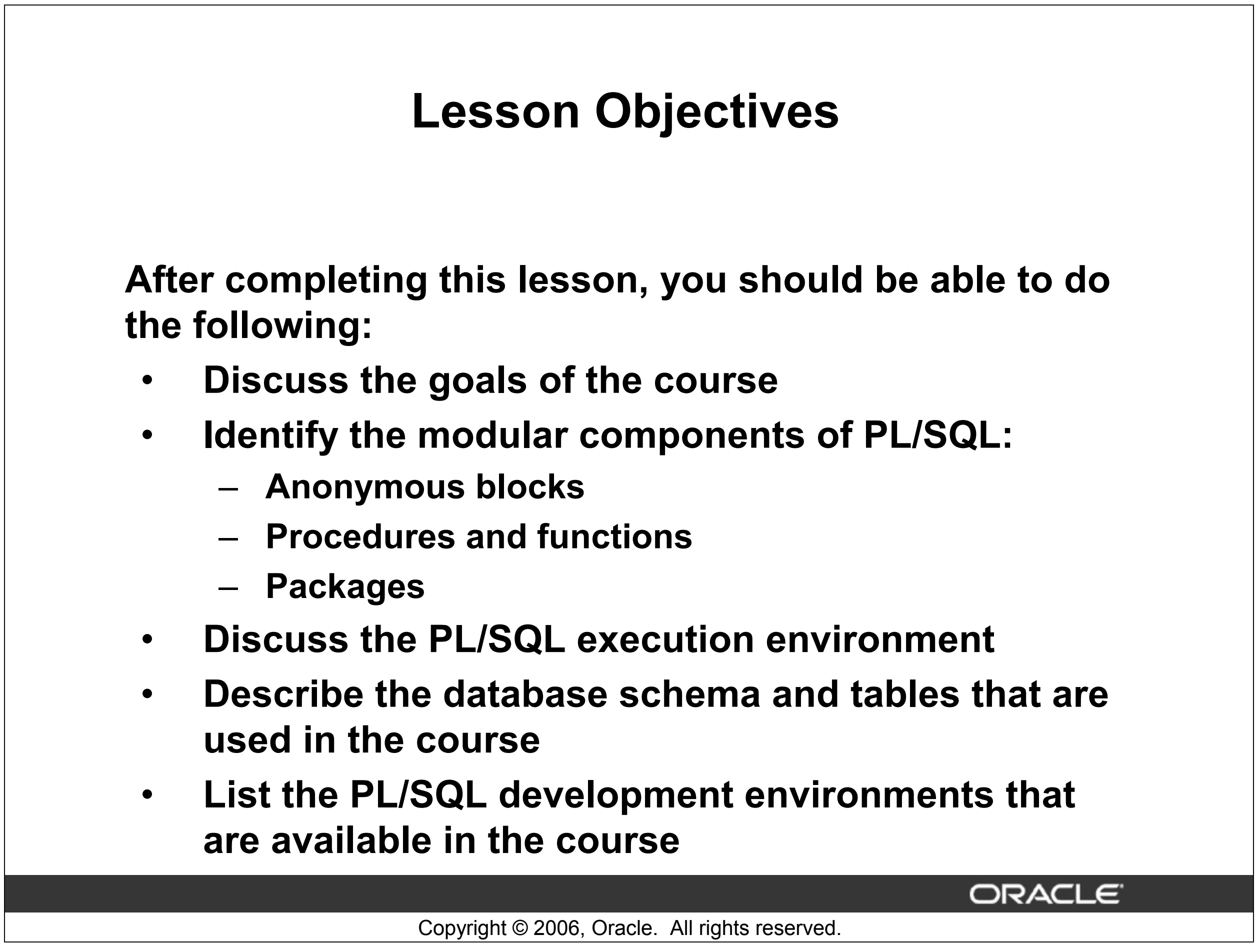
Introduction



Lesson Objectives

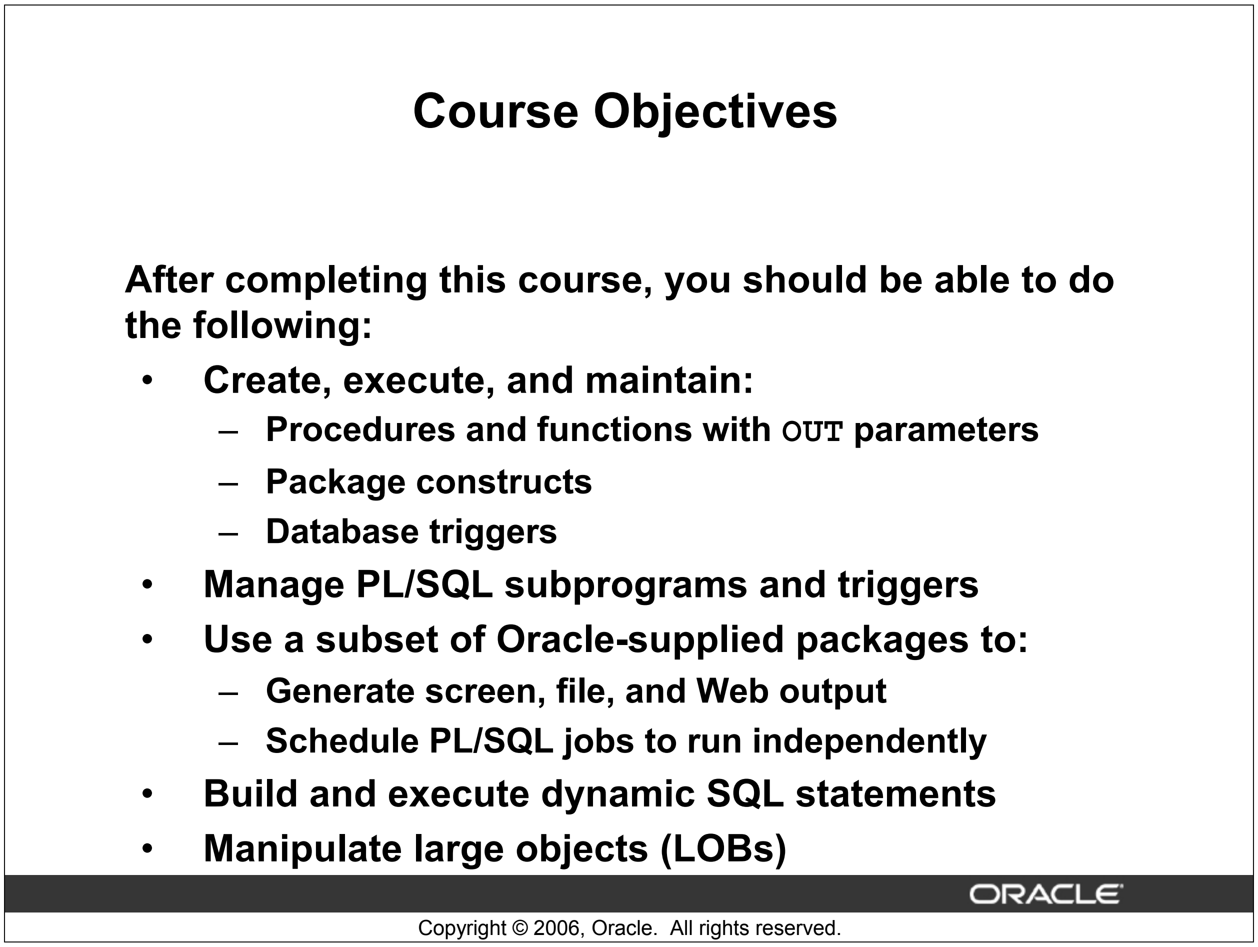
After completing this lesson, you should be able to do the following:

- **Discuss the goals of the course**
- **Identify the modular components of PL/SQL:**
  - **Anonymous blocks**
  - **Procedures and functions**
  - **Packages**
- **Discuss the PL/SQL execution environment**
- **Describe the database schema and tables that are used in the course**
- **List the PL/SQL development environments that are available in the course**

Lesson Aim

PL/SQL supports many program constructs. In this lesson, you review program units in the form of anonymous blocks, and you are introduced to named PL/SQL blocks. The named PL/SQL blocks are also referred to as subprograms. The named PL/SQL blocks include procedures and functions.

The tables from the Human Resources (HR) schema (which is used for the practices in this course) are briefly discussed. The development tools for writing, testing, and debugging PL/SQL are listed.



Course Objectives

After completing this course, you should be able to do the following:

- **Create, execute, and maintain:**
  - **Procedures and functions with `OUT` parameters**
  - **Package constructs**
  - **Database triggers**
- **Manage PL/SQL subprograms and triggers**
- **Use a subset of Oracle-supplied packages to:**
  - **Generate screen, file, and Web output**
  - **Schedule PL/SQL jobs to run independently**
- **Build and execute dynamic SQL statements**
- **Manipulate large objects (LOBs)**

Course Objectives

You can develop modularized applications with database procedures by using database objects such as the following:

- Procedures and functions
- Packages
- Database triggers

Modular applications improve:

- Functionality
- Security
- Overall performance

Course Agenda

Lessons for day 1:

I. Introduction
1. Creating Stored Procedures
2. Creating Stored Functions
3. Creating Packages
4. Using More Package Concepts

Course Agenda

Lessons for day 2:

5. Using Oracle-Supplied Packages in Application Development
6. Dynamic SQL and Metadata
7. Design Considerations for PL/SQL Code
8. Managing Dependencies

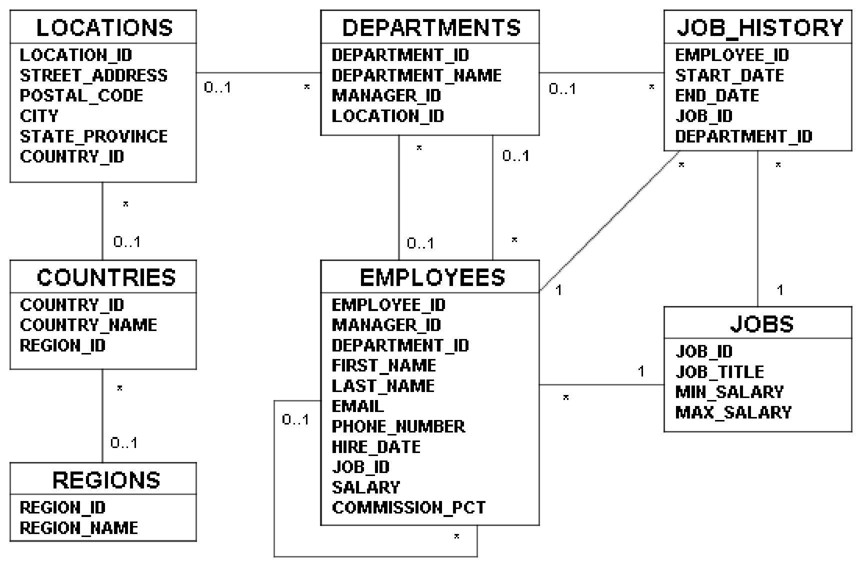
Course Agenda

Lessons for day 3:

9. Manipulating Large Objects
10. Creating Triggers
11. Applications for Triggers
12. Understanding and Influencing the PL/SQL Compiler

ORACLE

Human Resources (HR) Schema



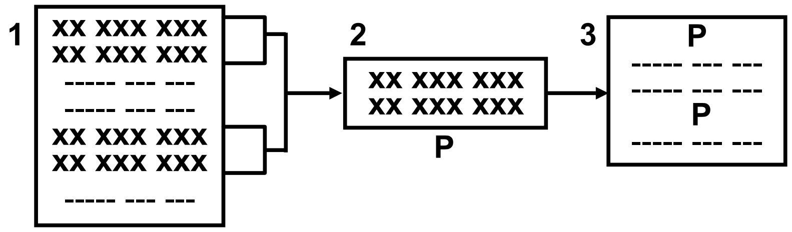
Human Resources (HR) Schema

The Human Resources (HR) schema is part of the Oracle Sample Schemas that can be installed in an Oracle database. The practice sessions in this course use data from the HR schema.

Table Descriptions

- `REGIONS` contains rows that represent a region such as Americas, Asia, and so on.
- `COUNTRIES` contains rows for countries, each of which is associated with a region.
- `LOCATIONS` contains the specific address of a specific office, warehouse, or production site of a company in a particular country.
- `DEPARTMENTS` shows details about the departments in which employees work. Each department may have a relationship representing the department manager in the `EMPLOYEES` table.
- `EMPLOYEES` contains details about each employee working for a department. Some employees may not be assigned to any department.
- `JOBS` contains the job types that can be held by each employee.
- `JOB_HISTORY` contains the job history of the employees. If an employee changes departments within a job or changes jobs within a department, then a new row is inserted into this table with the old job information of the employee.

Creating a Modularized and Layered Subprogram Design



- **Modularize code into subprograms.**
  1. **Locate code sequences repeated more than once.**
  2. **Create subprogram P containing the repeated code.**
  3. **Modify original code to invoke the new subprogram.**
- **Create subprogram layers for your application.**
  - **Data access subprogram layer with SQL logic**
  - **Business logic subprogram layer, which may or may not use data access layer**

Creating a Modularized and Layered Subprogram Design

The diagram illustrates the principle of modularization with subprograms: the creation of smaller manageable pieces of flexible and reusable code. Flexibility is achieved by using subprograms with parameters, which in turn makes the same code reusable for different input values. To modularize existing code, perform the following steps:

1. Locate and identify repetitive sequences of code.
2. Move the repetitive code into a PL/SQL subprogram.
3. Replace the original repetitive code with calls to the new PL/SQL subprogram.

Subprogram Layers

Because PL/SQL allows SQL statements to be seamlessly embedded into the logic, it is too easy to have SQL statement spread all over the code. However, it is recommended that you keep the SQL logic separate from the business logic—that is, create a layered application design with a minimum of two layers:

- **Data access layer:** For subroutines to access the data by using SQL statements
- **Business logic layer:** For subprograms to implement the business processing rules, which may or may not call on the data access layer routines

Following this modular and layered approach can help you create code that is easier to maintain, particularly when the business rules change. In addition, keeping the SQL logic simple and free of complex business logic can benefit from the work of Oracle Database Optimizer, which can reuse parsed SQL statements for better use of server-side resources.

Modularizing Development with PL/SQL Blocks

- **PL/SQL is a block-structured language. The PL/SQL code block helps modularize code by using:**
  - **Anonymous blocks**
  - **Procedures and functions**
  - **Packages**
  - **Database triggers**
- **The benefits of using modular program constructs are:**
  - **Easy maintenance**
  - **Improved data security and integrity**
  - **Improved performance**
  - **Improved code clarity**

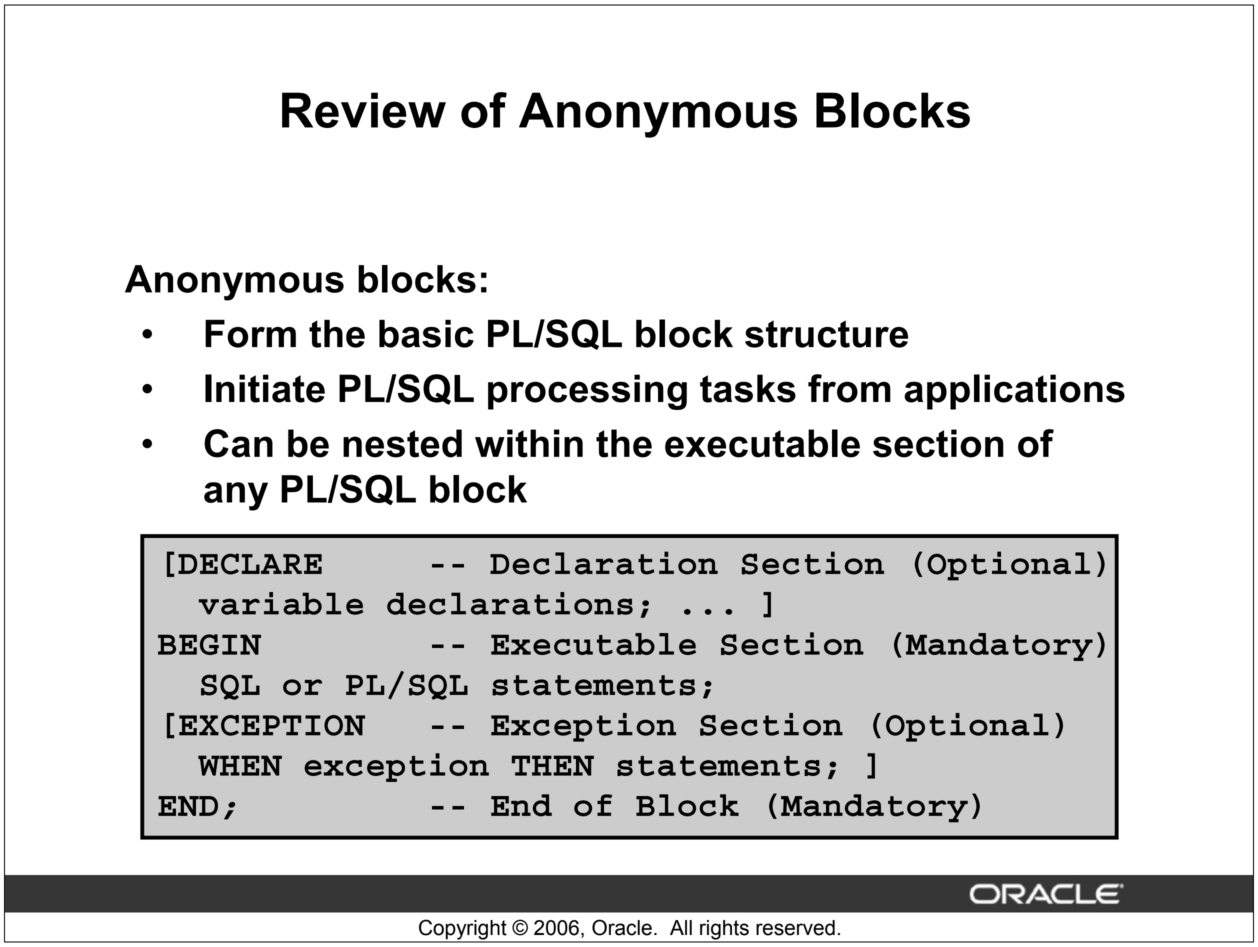
Modularizing Development with PL/SQL Blocks

A subprogram is based on standard PL/SQL structures. It contains a declarative section, an executable section, and an optional exception-handling section (for example, anonymous blocks, procedures, functions, packages, and triggers). Subprograms can be compiled and stored in the database, providing modularity, extensibility, reusability, and maintainability.

Modularization converts large blocks of code into smaller groups of code called modules. After modularization, the modules can be reused by the same program or shared with other programs. It is easier to maintain and debug code that comprises smaller modules than it is to maintain code in a single large program. Modules can be easily extended for customization by incorporating more functionality, if required, without affecting the remaining modules of the program.

Subprograms provide easy maintenance because the code is located in one place and any modifications required to the subprogram can therefore be performed in this single location. Subprograms provide improved data integrity and security. The data objects are accessed through the subprogram, and a user can invoke the subprogram only if the appropriate access privilege is granted to the user.

Note: Knowing how to develop anonymous blocks is a prerequisite for this course. For detailed information about anonymous blocks, see the course titled *Oracle 10g: PL/SQL Fundamentals*.



Review of Anonymous Blocks

Anonymous blocks:

- **Form the basic PL/SQL block structure**
- **Initiate PL/SQL processing tasks from applications**
- **Can be nested within the executable section of any PL/SQL block**

```
[DECLARE     -- Declaration Section (Optional)
  variable declarations; ... ]
BEGIN        -- Executable Section (Mandatory)
  SQL or PL/SQL statements;
[EXCEPTION   -- Exception Section (Optional)
  WHEN exception THEN statements; ]
END;         -- End of Block (Mandatory)
```

Review of Anonymous Blocks

Anonymous blocks are typically used for:

- Writing trigger code for Oracle Forms components
- Initiating calls to procedures, functions, and package constructs
- Isolating exception handling within a block of code
- Nesting inside other PL/SQL blocks for managing code flow control

The `DECLARE` keyword is optional, but it is required if you declare variables, constants, and exceptions to be used within the PL/SQL block.

`BEGIN` and `END` are mandatory and require at least one statement between them, either SQL, PL/SQL, or both.

The exception section is optional and is used to handle errors that occur within the scope of the PL/SQL block. Exceptions can be propagated to the caller of the anonymous block by excluding an exception handler for the specific exception, thus creating what is known as an *unhandled* exception.

Introduction to PL/SQL Procedures

Procedures are named PL/SQL blocks that perform a sequence of actions.

```
CREATE PROCEDURE getemp IS  -- header
  emp_id  employees.employee_id%type;
  lname   employees.last_name%type;
BEGIN
  emp_id := 100;
  SELECT last_name INTO lname
  FROM EMPLOYEES
  WHERE employee_id = emp_id;
  DBMS_OUTPUT.PUT_LINE('Last name: '||lname);
END;
/
```

Introduction to PL/SQL Procedures

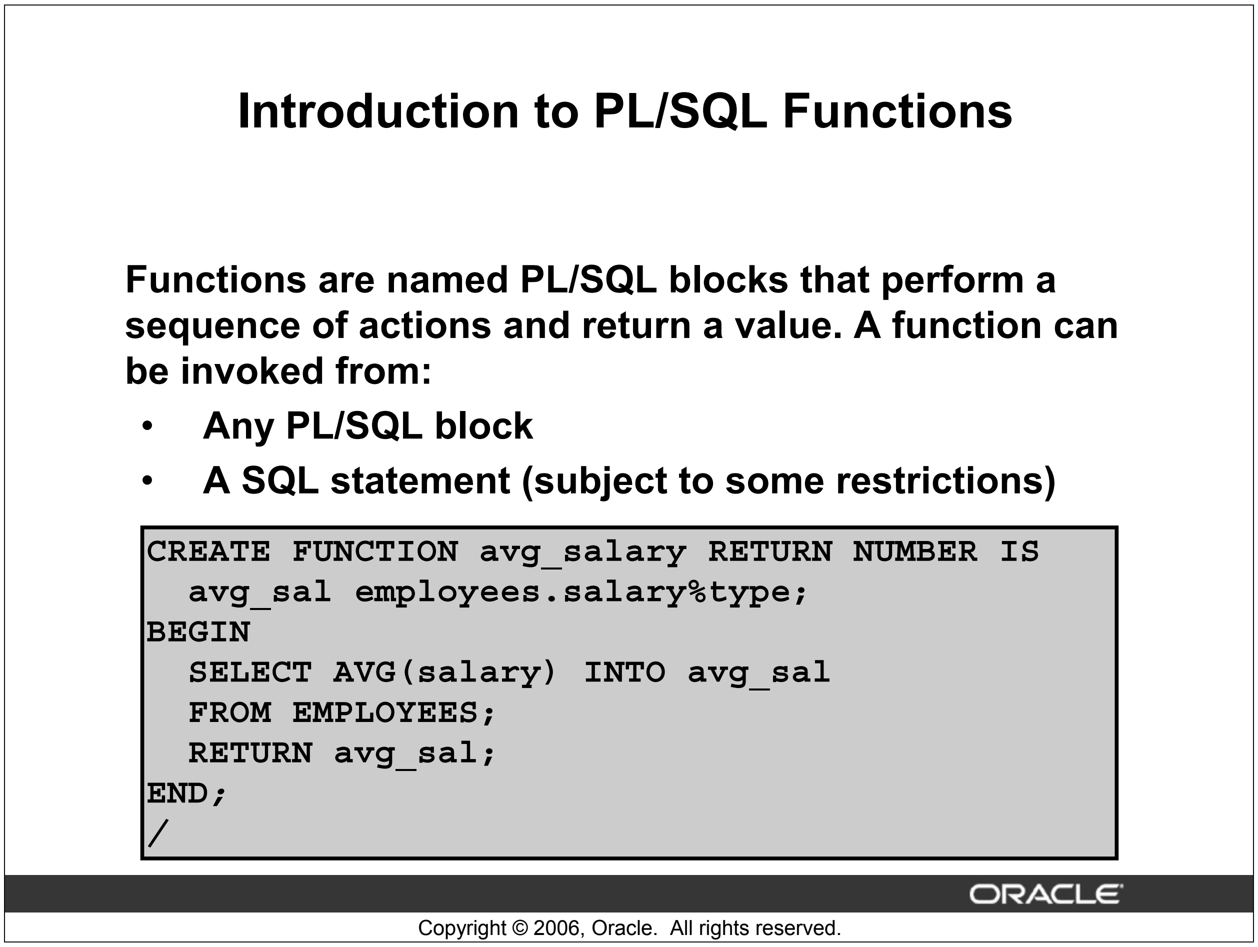
A procedure is a named PL/SQL block that is created by using a `CREATE PROCEDURE` statement. When called or invoked, a procedure performs a sequence of actions. The anatomy of the procedure includes:

- **The header:** "`PROCEDURE getemp IS`"
- **The declaration section:** With variable declarations `emp_id` and `lname`
- **The executable section:** Contains the PL/SQL and SQL statements used to obtain the last name of employee 100. The executable section makes use of the `DBMS_OUTPUT` package to print the last name of the employee.
- **The exception section:** Optional and not used in the example

Note: Hard-coding the value of `100` for `emp_id` is inflexible. The procedure would be more reusable if used as a parameter to obtain the employee ID value. Using parameters is covered in the lesson titled "Creating Stored Procedures."

To call a procedure by using an anonymous block, use the following:

```
BEGIN
  getemp;
END;
```



Introduction to PL/SQL Functions

Functions are named PL/SQL blocks that perform a sequence of actions and return a value. A function can be invoked from:

- **Any PL/SQL block**
- **A SQL statement (subject to some restrictions)**

```
CREATE FUNCTION avg_salary RETURN NUMBER IS
  avg_sal employees.salary%type;
BEGIN
  SELECT AVG(salary) INTO avg_sal
  FROM EMPLOYEES;
  RETURN avg_sal;
END;
/
```

Introduction to PL/SQL Functions

A function is a named PL/SQL block that is created with a `CREATE FUNCTION` statement. Functions are used to compute a value that must be returned to the caller.

PL/SQL functions follow the same block structure as procedures. However, the header starts with the keyword `FUNCTION` followed by the function name. The header includes the return data type after the function name (using the keyword `RETURN` followed by the data type). The example declares a variable called `avg_sal` in the declaration section, and uses the `RETURN` statement to return the value retrieved from the `SELECT` statement. The value returned represents the average salary for all employees.

A function can be called from:

- Another PL/SQL block where its return value can be stored in a variable or supplied as a parameter to a procedure
- A SQL statement, subject to restrictions (This topic is covered in the lesson titled "Creating Stored Functions.")

To call a function from an anonymous block, use the following:

```
BEGIN
  dbms_output.put_line('Average Salary: ' ||
     avg_salary);
END;
```

Introduction to PL/SQL Packages

PL/SQL packages have a specification and an optional body. Packages group related subprograms together.

```
CREATE PACKAGE emp_pkg IS
  PROCEDURE getemp;
  FUNCTION avg_salary RETURN NUMBER;
END emp_pkg;
/
CREATE PACKAGE BODY emp_pkg IS
  PROCEDURE getemp IS ...
  BEGIN ... END;

  FUNCTION avg_salary RETURN NUMBER IS ...
  BEGIN ... RETURN avg_sal; END;
END emp_pkg;
/
```

Introduction to PL/SQL Packages

A PL/SQL package is typically made up of two parts:

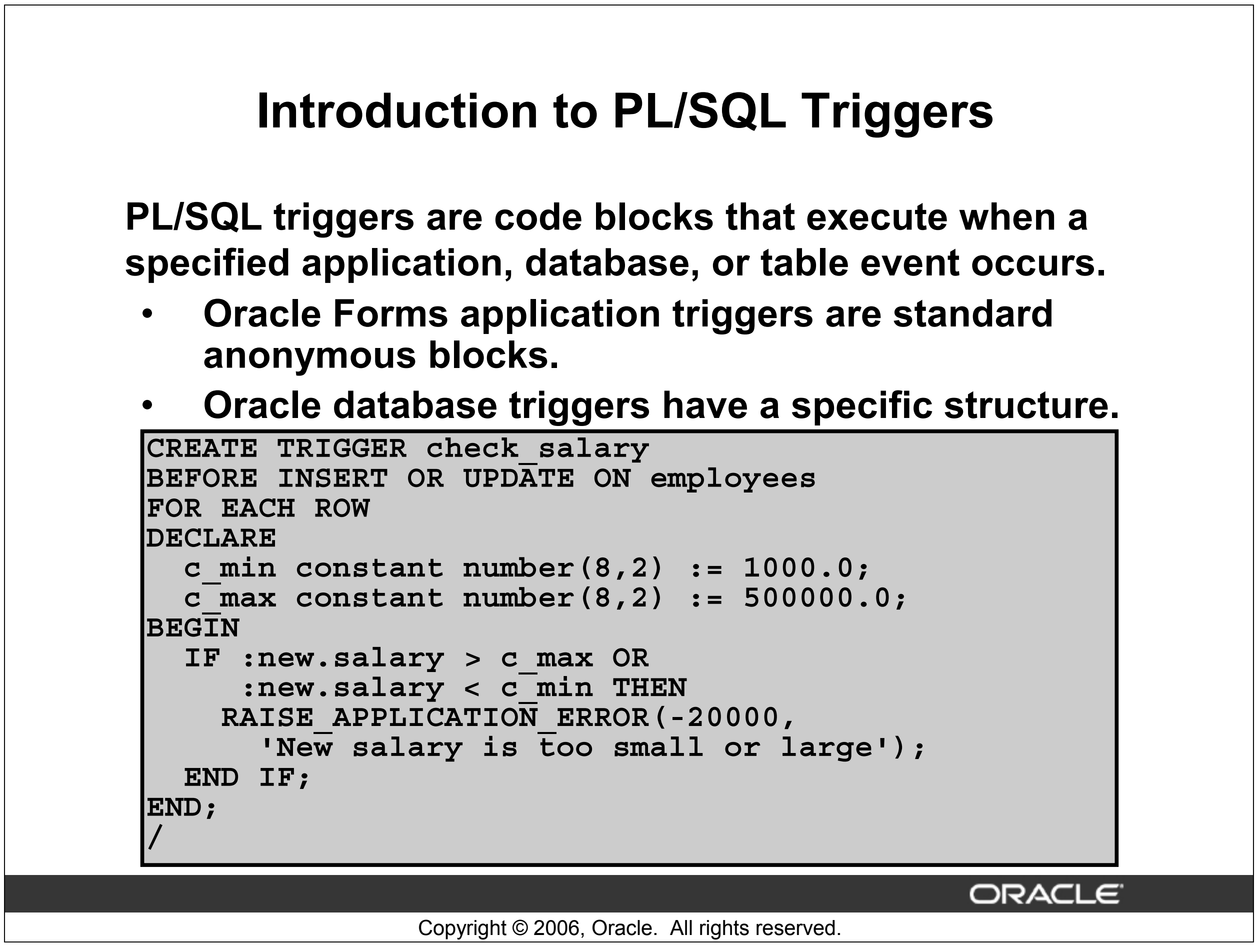
- A package specification declaring the public (accessible) components of the package
- A package body that implements public and private components of the package

A package is used to group related PL/SQL code and constructs together. This helps developers manage and maintain related code in one place. Packages are powerful constructs and provide a way to logically modularize code into application-specific or functional groups.

The example declares a package called `emp_package` that comprises a procedure `getemp` and a function `avg_salary`. The specification defines the procedure and function heading. The package body provides the full implementation of the procedure and function declared in the package specification. The example is incomplete but provides an introduction to the concept of a PL/SQL package. The details about PL/SQL packages are covered in the lesson titled “Creating Packages.”

To call the package procedure from an anonymous block, use the following:

```
BEGIN
 emp_pkg.getemp;
END;
```



Introduction to PL/SQL Triggers

PL/SQL triggers are code blocks that execute when a specified application, database, or table event occurs.

- **Oracle Forms application triggers are standard anonymous blocks.**
- **Oracle database triggers have a specific structure.**

```
CREATE TRIGGER check_salary
BEFORE INSERT OR UPDATE ON employees
FOR EACH ROW
DECLARE
  c_min constant number(8,2) := 1000.0;
  c_max constant number(8,2) := 500000.0;
BEGIN
  IF :new.salary > c_max OR
     :new.salary < c_min THEN
    RAISE_APPLICATION_ERROR(-20000,
      'New salary is too small or large');
  END IF;
END;
/
```

Introduction to PL/SQL Triggers

A trigger is a PL/SQL block that executes when a particular event occurs.

A PL/SQL trigger has two forms:

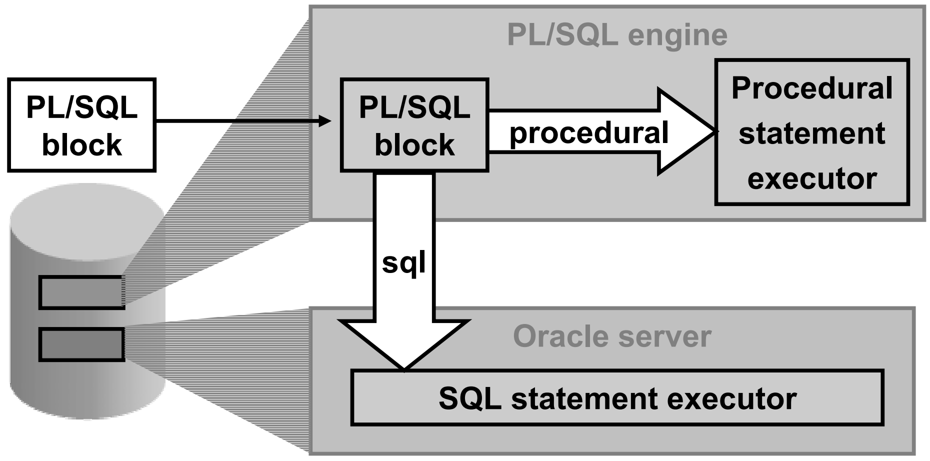
- **Application trigger:** Block of code that executes when a nominated application event occurs in an Oracle Forms execution environment
- **Database trigger:** Named block of code that is associated and executes when a nominated database or table event occurs

The `check_salary` trigger example shows a database trigger that executes before either an `INSERT` or an `UPDATE` operation occurs on the `EMPLOYEES` table. The trigger code checks whether the `salary` column value is within an acceptable range. If the value is outside the specified range, the code uses the `RAISE_APPLICATION_ERROR` built-in procedure to fail the operation. The `:new` syntax, which is used in the example, is a special bind/host variable that can be used in row-level triggers.

Triggers are covered in detail in the lesson titled "Creating Triggers."

PL/SQL Execution Environment

The PL/SQL run-time architecture:



PL/SQL Execution Environment

The diagram shows a PL/SQL block being executed by the PL/SQL engine. The PL/SQL engine resides in:

- The Oracle database for executing stored subprograms
- The Oracle Forms client when running client/server applications, or in the Oracle Application Server when using Oracle Forms Services to run Forms on the Web

Irrespective of the PL/SQL run-time environment, the basic architecture remains the same. Therefore, all PL/SQL statements are processed in the Procedural Statement Executor, and all SQL statements must be sent to the SQL Statement Executor for processing by the Oracle server processes.

The PL/SQL engine is a virtual machine that resides in memory and processes the PL/SQL m-code instructions. When the PL/SQL engine encounters a SQL statement, a context switch is made to pass the SQL statement to the Oracle server processes. The PL/SQL engine waits for the SQL statement to complete and for the results to be returned before it continues to process subsequent statements in the PL/SQL block.

The Oracle Forms PL/SQL engine runs in the client for the client/server implementation, and in the application server for the Forms Services implementation. In either case, SQL statements are typically sent over a network to an Oracle server for processing.

PL/SQL Development Environments

This course provides the following tools for developing PL/SQL code:

- **Oracle SQL*Plus (GUI or command-line versions)**
- **Oracle *i*SQL*Plus (used from a browser)**
- **Oracle JDeveloper IDE**

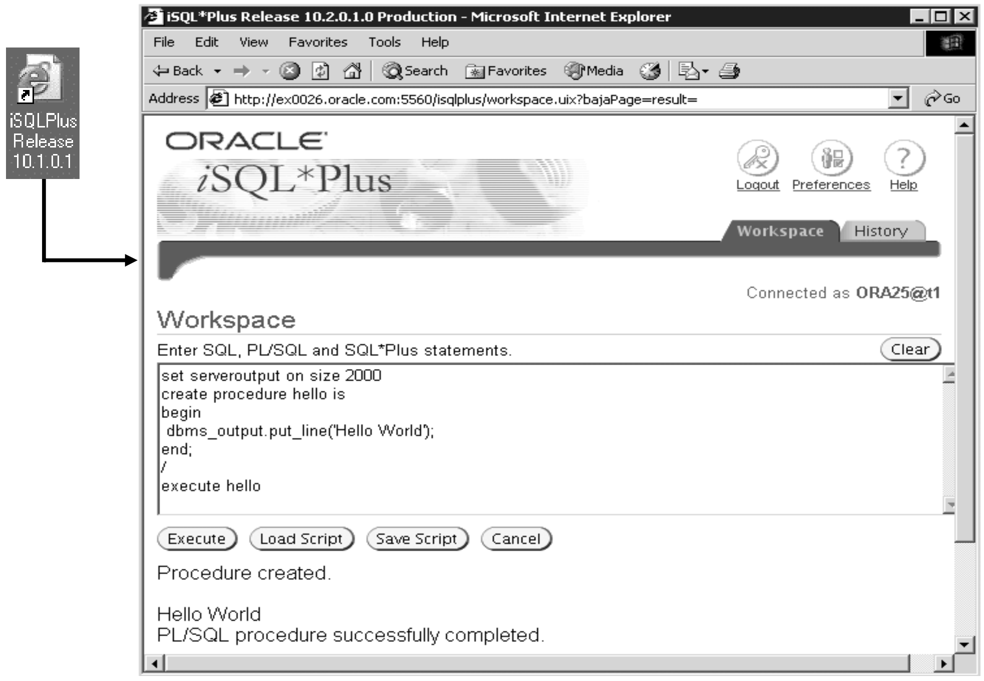
PL/SQL Development Environments

There are many tools that provide an environment for developing PL/SQL code. Oracle provides several tools that can be used to write PL/SQL code. Some of the development tools that are available for use in this course are:

- Oracle *i*SQL*Plus: A browser-based application
- Oracle SQL*Plus: A window or command-line application
- Oracle JDeveloper: A window-based integrated development environment (IDE)

Note: The code and screen examples presented in the course notes were generated from output in the *i*SQL*Plus environment.

Coding PL/SQL in *i*SQL*Plus


iSQLPlus Release 10.1.0.1
iSQL*Plus Release 10.2.0.1.0 Production - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://ex0026.oracle.com:5560/isqlplus/workspace.uix?bajaPage=result=
Go
ORACLE iSQL*Plus
Logout Preferences Help
Workspace History
Connected as ORA25@t1
Workspace
Enter SQL, PL/SQL and SQL*Plus statements.
Clear
Execute Load Script Save Script Cancel

```
set serveroutput on size 2000
create procedure hello is
begin
 dbms_output.put_line('Hello World');
end;
/
execute hello
```

```
Procedure created.

Hello World
PL/SQL procedure successfully completed.
```

Coding PL/SQL in *i*SQL*Plus

Oracle *i*SQL*Plus is a Web application that allows you to submit SQL statements and PL/SQL blocks for execution and receive the results in a standard Web browser.

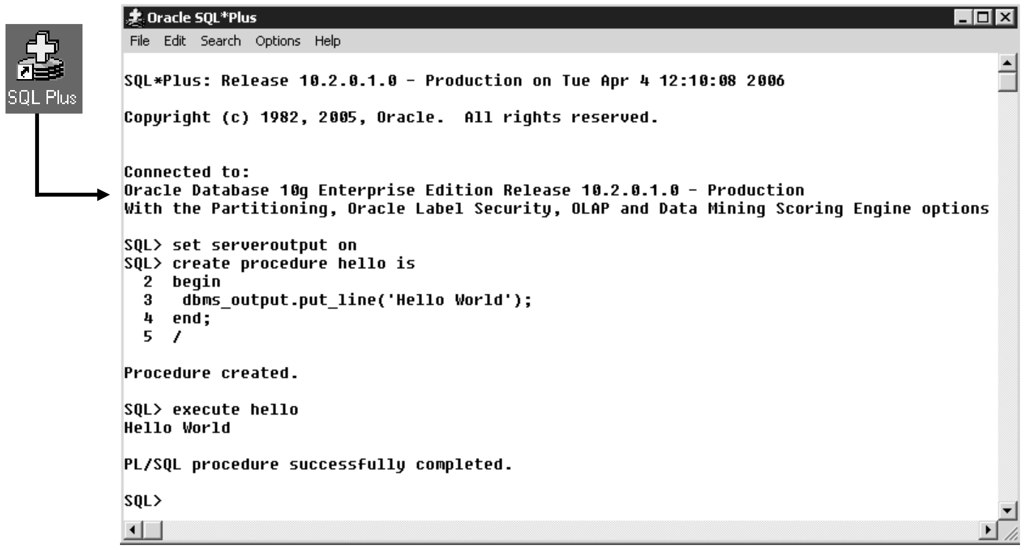
*i*SQL*Plus is:

- Shipped with the database
- Installed in the middle tier
- Accessed from a Web browser by using a URL format that is similar to the following example:
  `http://host:port/isqlplus`
  The `host` and `port` are for the Web server name and HTTP listener port.

When coding PL/SQL subprograms in the *i*SQL*Plus tool, consider the following:

- You create subprograms by using the `CREATE` SQL statement.
- You execute subprograms by using either an anonymous PL/SQL block or the `EXECUTE` command.
- If you use the `DBMS_OUTPUT` package procedures to print text to the screen, you must first execute the `SET SERVEROUTPUT ON` command in your session.

Coding PL/SQL in SQL*Plus



```
SQL*Plus: Release 10.2.0.1.0 - Production on Tue Apr 4 12:10:08 2006

Copyright (c) 1982, 2005, Oracle.  All rights reserved.


Connected to:
Oracle Database 10g Enterprise Edition Release 10.2.0.1.0 - Production
With the Partitioning, Oracle Label Security, OLAP and Data Mining Scoring Engine options

SQL> set serveroutput on
SQL> create procedure hello is
  2  begin
  3   dbms_output.put_line('Hello World');
  4  end;
  5  /

Procedure created.

SQL> execute hello
Hello World

PL/SQL procedure successfully completed.

SQL>
```

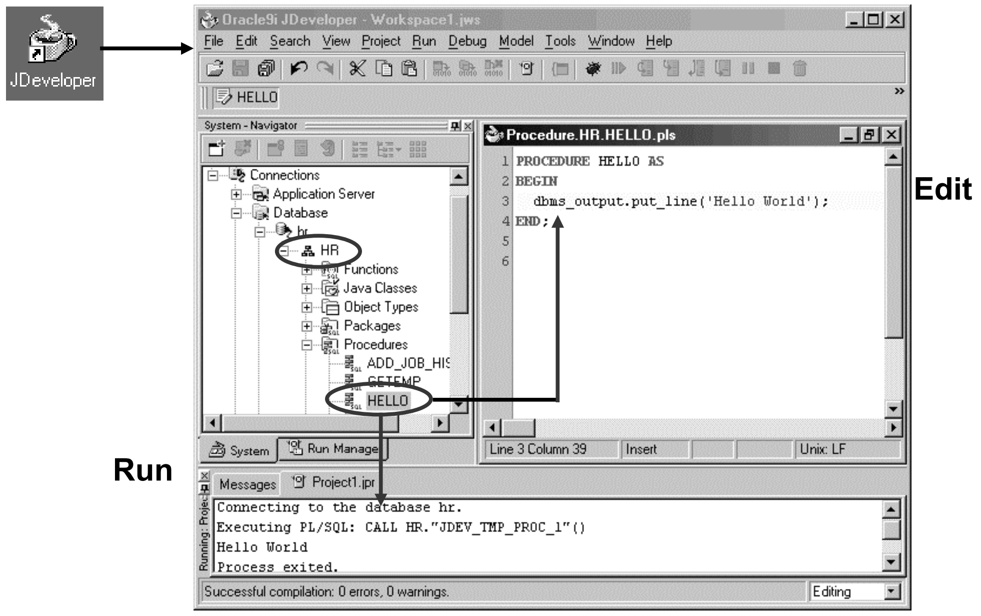
Coding PL/SQL in SQL*Plus

Oracle SQL*Plus is a graphical user interface (GUI) or command-line application that enables you to submit SQL statements and PL/SQL blocks for execution and receive the results in an application or command window.

SQL*Plus is:
- Shipped with the database
- Installed on a client and on the database server system
- Accessed from an icon or the command line

When coding PL/SQL subprograms using SQL*Plus, consider the following:
- You create subprograms by using the `CREATE SQL` statement.
- You execute subprograms by using either an anonymous PL/SQL block or the `EXECUTE` command.
- If you use the `DBMS_OUTPUT` package procedures to print text to the screen, you must first execute the `SET SERVEROUTPUT ON` command in your session.

Coding PL/SQL in Oracle JDeveloper



```
PROCEDURE HELLO AS
BEGIN
  dbms_output.put_line('Hello World');
END;
```

```
Connecting to the database hr.
Executing PL/SQL: CALL HR."JDEV_TMP_PROC_1"()
Hello World
Process exited.
```

Successful compilation: 0 errors, 0 warnings.

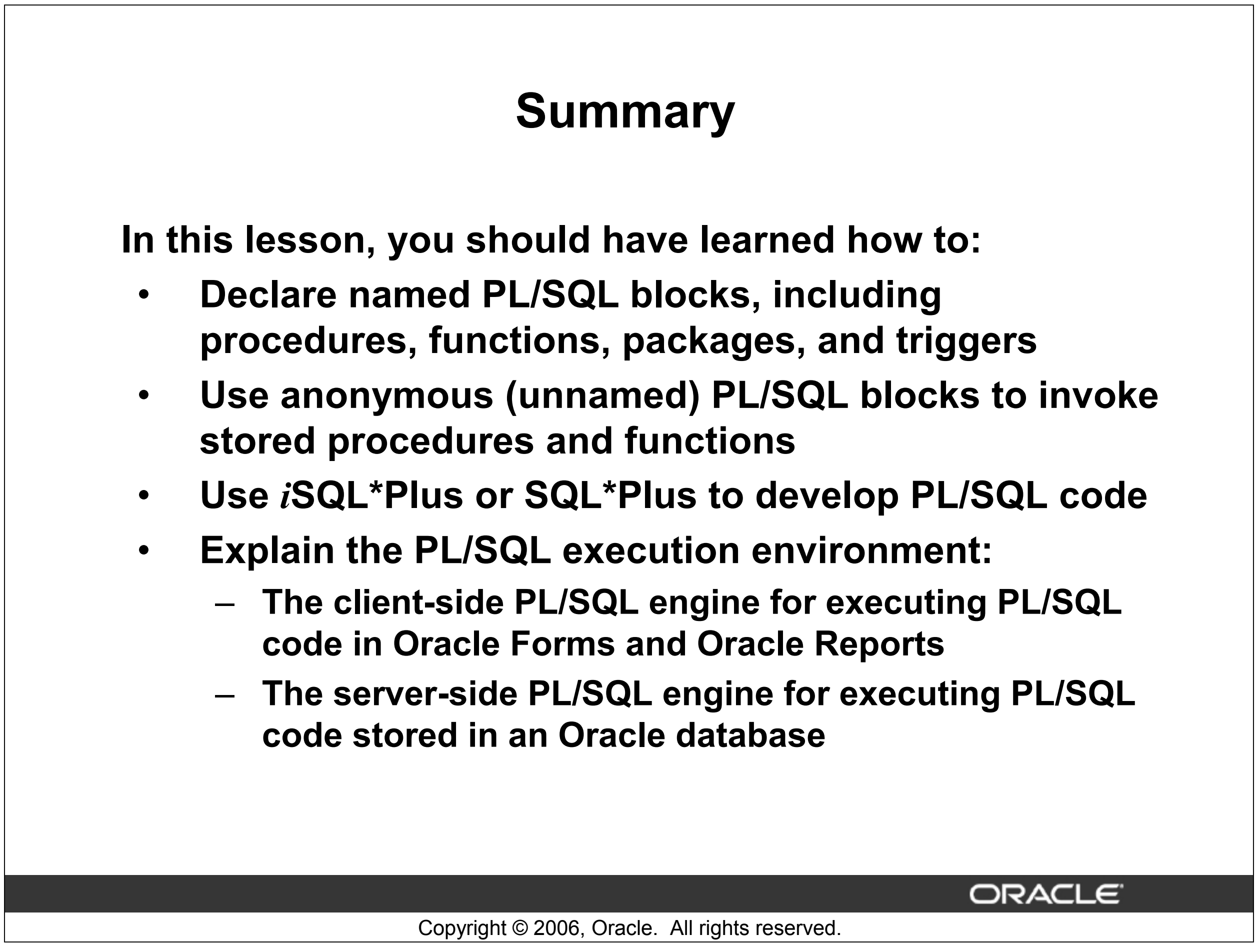
Coding PL/SQL in Oracle JDeveloper

Oracle JDeveloper allows developers to create, edit, test, and debug PL/SQL code by using a sophisticated GUI. Oracle JDeveloper is a part of Oracle Developer Suite and is also available as a separate product.

When coding PL/SQL in JDeveloper, consider the following:

- You first create a database connection to enable JDeveloper to access a database schema owner for the subprograms.
- You can then use the JDeveloper context menus on the Database connection to create a new subprogram construct using the built-in JDeveloper Code Editor. The JDeveloper Code Editor provides an excellent environment for PL/SQL development, with features such as the following:
  - Different colors for syntactical components of the PL/SQL language
  - Code insight to rapidly locate procedures and functions in supplied packages
- You invoke a subprogram by using a `Run` command on the context menu for the named subprogram. The output appears in the JDeveloper Log Message window, as shown in the lower portion of the screenshot.

Note: JDeveloper provides color-coding syntax in the JDeveloper Code Editor and is sensitive to PL/SQL language constructs and statements.



Summary

In this lesson, you should have learned how to:

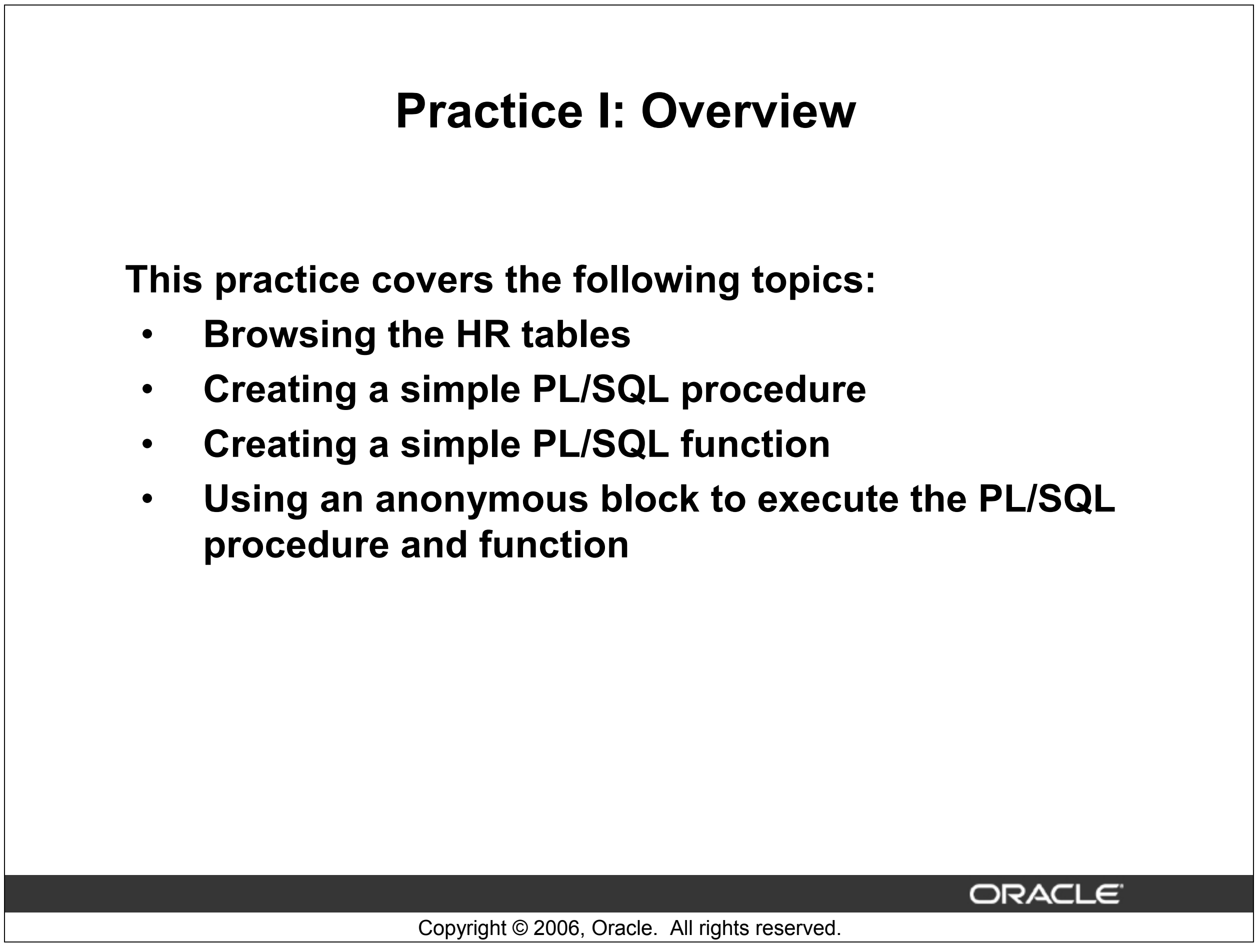
- **Declare named PL/SQL blocks, including procedures, functions, packages, and triggers**
- **Use anonymous (unnamed) PL/SQL blocks to invoke stored procedures and functions**
- **Use *i*SQL*Plus or SQL*Plus to develop PL/SQL code**
- **Explain the PL/SQL execution environment:**
  – **The client-side PL/SQL engine for executing PL/SQL code in Oracle Forms and Oracle Reports**
  – **The server-side PL/SQL engine for executing PL/SQL code stored in an Oracle database**

Summary

The PL/SQL language provides different program constructs for blocks of reusable code. Unnamed or anonymous PL/SQL blocks can be used to invoke SQL and PL/SQL actions, procedures, functions, and package components. Named PL/SQL blocks, otherwise known as subprograms, include:

- Procedures
- Functions
- Package procedures and functions
- Triggers

Oracle supplies several tools to develop your PL/SQL functionality. Oracle provides a client-side or middle-tier PL/SQL run-time environment for Oracle Forms and Oracle Reports, and provides a PL/SQL run-time engine inside the Oracle database. Procedures and functions inside the database can be invoked from any application code that can connect to an Oracle database and execute PL/SQL code.



Practice I: Overview

This practice covers the following topics:

- Browsing the HR tables
- Creating a simple PL/SQL procedure
- Creating a simple PL/SQL function
- Using an anonymous block to execute the PL/SQL procedure and function

Practice I: Overview

In this practice, you use *i*SQL*Plus to execute SQL statements to examine data in the HR schema. You also create a simple procedure and function that you invoke by using an anonymous block or the `EXECUTE` command in *i*SQL*Plus. Optionally, you can experiment by creating and executing the PL/SQL code in SQL*Plus.

Note: All written practices use *i*SQL*Plus as the development environment. However, you can use any of the tools that are provided in your course environment.

Practice I

1. Launch *i*SQL*Plus using the icon that is provided on your desktop.
   a. Log in to the database by using the username and database connect string details provided by your instructor (you may optionally write the information here for your records):
   Username: **`ora__`**
   Password: **`oracle`**
   Database Connect String/Tnsname: **`T__`**
   b. Execute basic `SELECT` statements to query the data in the `DEPARTMENTS`, `EMPLOYEES`, and `JOBS` tables. Take a few minutes to familiarize yourself with the data, or consult Appendix B, which provides a description and some data from each table in the Human Resources schema.
2. Create a procedure called `HELLO` to display the text `Hello World`.
   a. Create a procedure called `HELLO`.
   b. In the executable section, use the `DBMS_OUTPUT.PUT_LINE` procedure to print the text `Hello World`, and save the code in the database.
   Note: If you get compile-time errors, correct the PL/SQL code and replace the `CREATE` keyword with the text `CREATE OR REPLACE`.
   c. Execute the `SET SERVEROUTPUT ON` command to ensure that the output from the `DBMS_OUTPUT.PUT_LINE` procedure is displayed in *i*SQL*Plus.
   d. Create an anonymous block to invoke the stored procedure.
3. Create a function called `TOTAL_SALARY` to compute the sum of all employee salaries.
   a. Create a function called `TOTAL_SALARY` that returns a `NUMBER`.
   b. In the executable section, execute a query to store the total salary of all employees in a local variable that you declare in the declaration section. Return the value stored in the local variable. Save and compile the code.
   c. Use an anonymous block to invoke the function. To display the result computed by the function, use the `DBMS_OUTPUT.PUT_LINE` procedure.
   Hint: Either nest the function call inside the `DBMS_OUTPUT.PUT_LINE` parameter, or store the function result in a local variable of the anonymous block and use the local variable in the `DBMS_OUTPUT.PUT_LINE` procedure.

If you have time, complete the following exercise:

4. Launch SQL*Plus using the icon that is provided on your desktop.
   a. Invoke the procedure and function that you created in exercises 2 and 3.
   b. Create a new procedure called `HELLO_AGAIN` to print `Hello World again`.
   c. Invoke the `HELLO_AGAIN` procedure with an anonymous block.