# Oracle 11g – PL SQL

## Introduction to PL/SQL

# Objectives

- After completing this lesson, you should be able to do the following:
  - Explain the need for PL/SQL
  - Explain the benefits of PL/SQL
  - Identify the different types of PL/SQL blocks
  - Output messages in PL/SQL

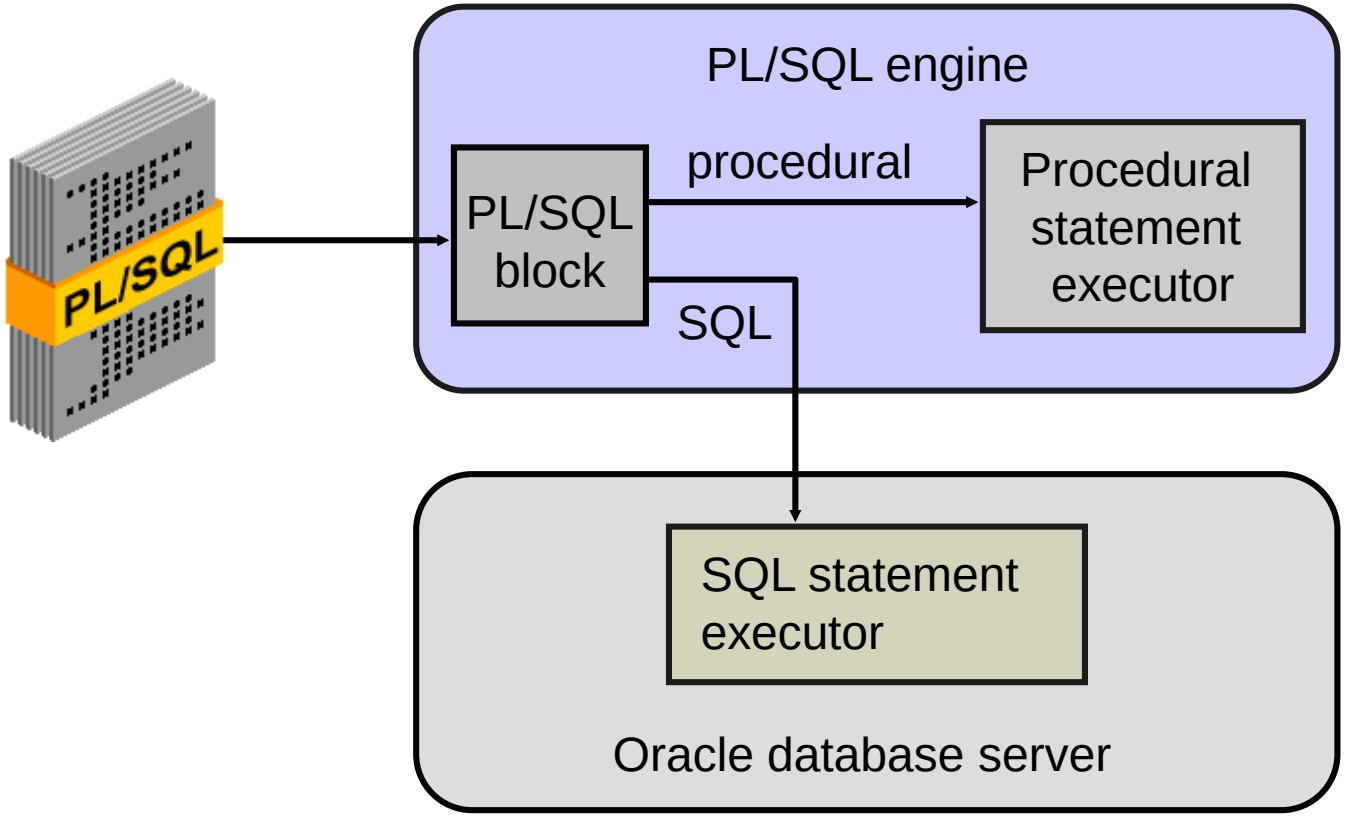
# About PL/SQL

- PL/SQL:
  - Stands for "Procedural Language extension to SQL"
  - Is Oracle Corporation's standard data access language for relational databases
  - Seamlessly integrates procedural constructs with SQL

# About PL/SQL

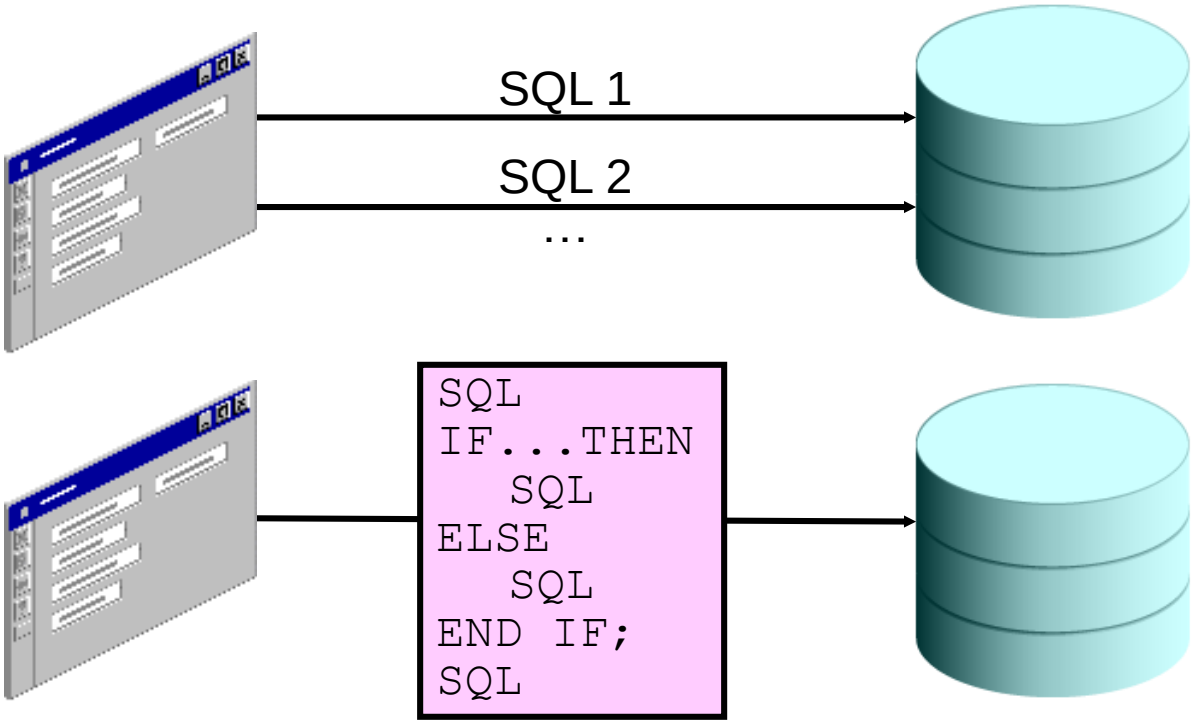
- PL/SQL:
  - Provides a block structure for executable units of code. Maintenance of code is made easier with such a well-defined structure.
  - Provides procedural constructs such as:
    - Variables, constants, and data types
    - Control structures such as conditional statements and loops
    - Reusable program units that are written once and executed many times

# PL/SQL Environment



# Benefits of PL/SQL

- Integration of procedural constructs with SQL
- Improved performance



# Benefits of PL/SQL

- Modularized program development
- Integration with Oracle tools
- Portability
- Exception handling

# PL/SQL Block Structure

- `DECLARE` (optional)
  - Variables, cursors, user-defined exceptions
- `BEGIN` (mandatory)
  - SQL statements
  - PL/SQL statements
- `EXCEPTION` (optional)
  - Actions to perform when errors occur
- `END;` (mandatory)

# Block Types

Anonymous

Procedure

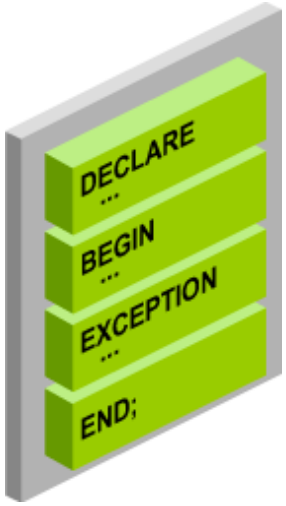
Function

```
[DECLARE]


BEGIN
  --statements

[EXCEPTION]

END;
```

```
PROCEDURE name
IS

BEGIN
  --statements

[EXCEPTION]

END;
```

```
FUNCTION name
RETURN datatype
IS
BEGIN
  --statements
  RETURN value;
[EXCEPTION]

END;
```

# Program Constructs




| Tools Constructs |
| --- |
| Anonymous blocks |
| Application procedures or functions |
| Application packages |
| Application triggers |
| Object types |

| Database Server Constructs |
| --- |
| Anonymous blocks |
| Stored procedures or functions |
| Stored packages |
| Database triggers |
| Object types |

# Create an Anonymous Block

- Enter the anonymous block in the SQL Developer workspace:

ora41

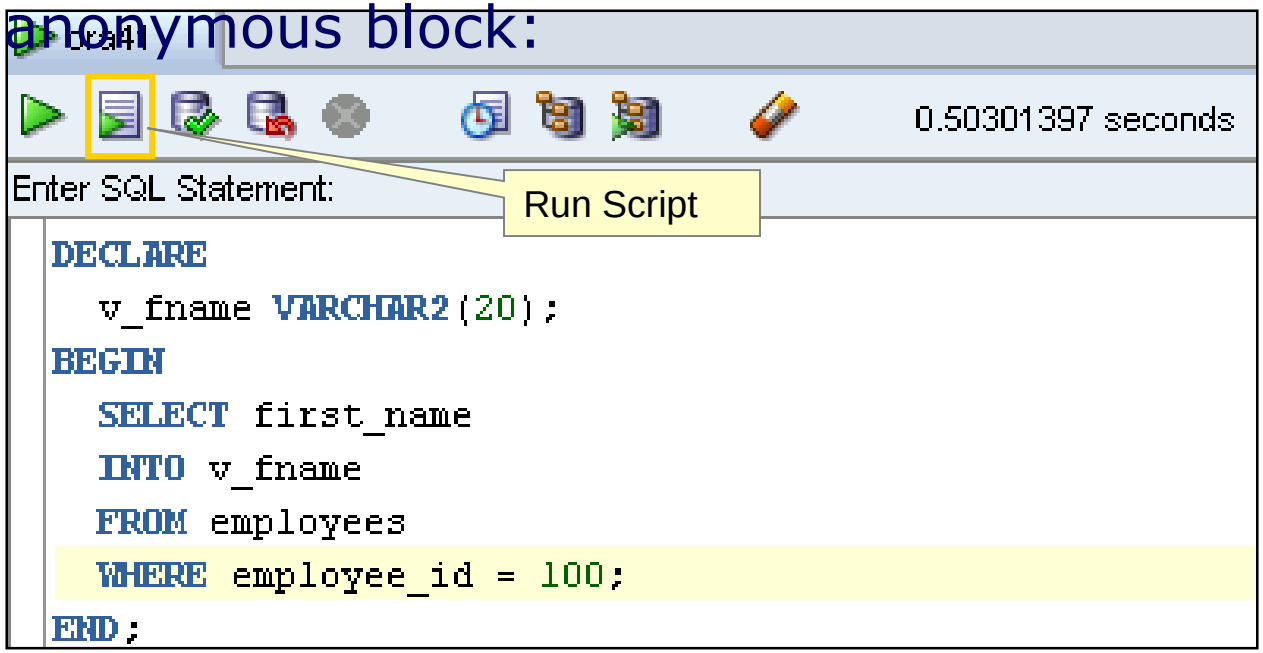
0.50301397 seconds

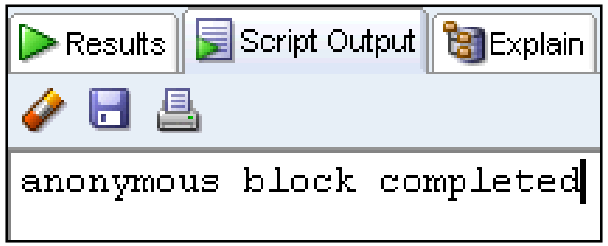
Enter SQL Statement:

```
DECLARE
   v_fname VARCHAR2(20);
BEGIN
   SELECT first_name
   INTO v_fname
   FROM employees
   WHERE employee_id = 100;
END;
```

# Execute an Anonymous Block

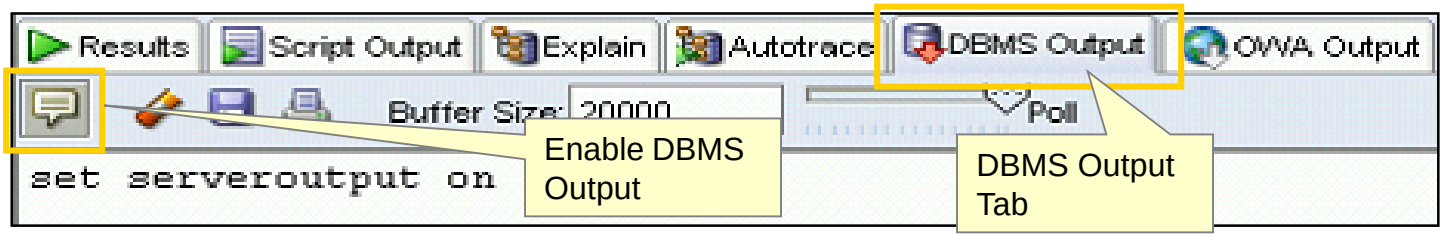
- Click the Run Script button to execute the anonymous block:



# Test the Output of a PL/SQL Block

- Enable output in SQL Developer by clicking the Enable DBMS Output button on the DBMS Output tab:




- Use a predefined Oracle package and its procedure:

```
SET SERVEROUTPUT ON ;
--Must Write this Line at the starting of PLSQL
Block for Print the Output.
```

```
DBMS_OUTPUT.PUT_LINE(' The First Name of the
Employee is ' || f_name);
…
```

# Test the Output of a PL/SQL Block

Connections

- Connections
  - ACDB_192.168.83.139_ORCL
  - BISAMPLE
  - HR_192.168.83.139_ORCL
  - SCOTT
  - SCOTT_192.168.83.139@ORCL
    - Tables (Filtered)
      - BONUS
      - DEMO
      - DEPT
      - EMP
        - EMPNO
        - ENAME
        - JOB
        - MGR
        - HIREDATE
        - SAL
        - COMM
        - DEPTNO
      - SALGRADE
    - Views
    - Editioning Views
    - Indexes
    - Packages
    - Procedures
    - Functions
    - Operators
    - Queues
    - Queues Tables
    - Triggers
    - Crossedition Triggers
    - Types
    - Sequences
    - Materialized Views
    - Materialized View Logs
    - Synonyms
    - Public Synonyms

Welcome Page | SCOTT_192.168.83.139@ORCL

0.123 seconds

Worksheet | Query Builder

```
1  SET SERVEROUTPUT ON;
2  DECLARE
3     v_name VARCHAR2(20);
4  BEGIN
5     SELECT ENAME
6     INTO v_name
7     FROM emp
8     WHERE empno=7839;
9     dbms_output.put_line('THE NAME OF THE EMPLOYEE IS ' || v_name);
10 END;
```

Script Output

Task completed in 0.123 seconds

```
THE NAME OF THE EMPLOYEE IS KING


PL/SQL procedure successfully completed.
```

# Summary

- In this lesson, you should have learned how to:
  - Integrate SQL statements with PL/SQL program constructs
  - Describe the benefits of PL/SQL
  - Differentiate between PL/SQL block types
  - Output messages in PL/SQL

# Practice 1: Overview

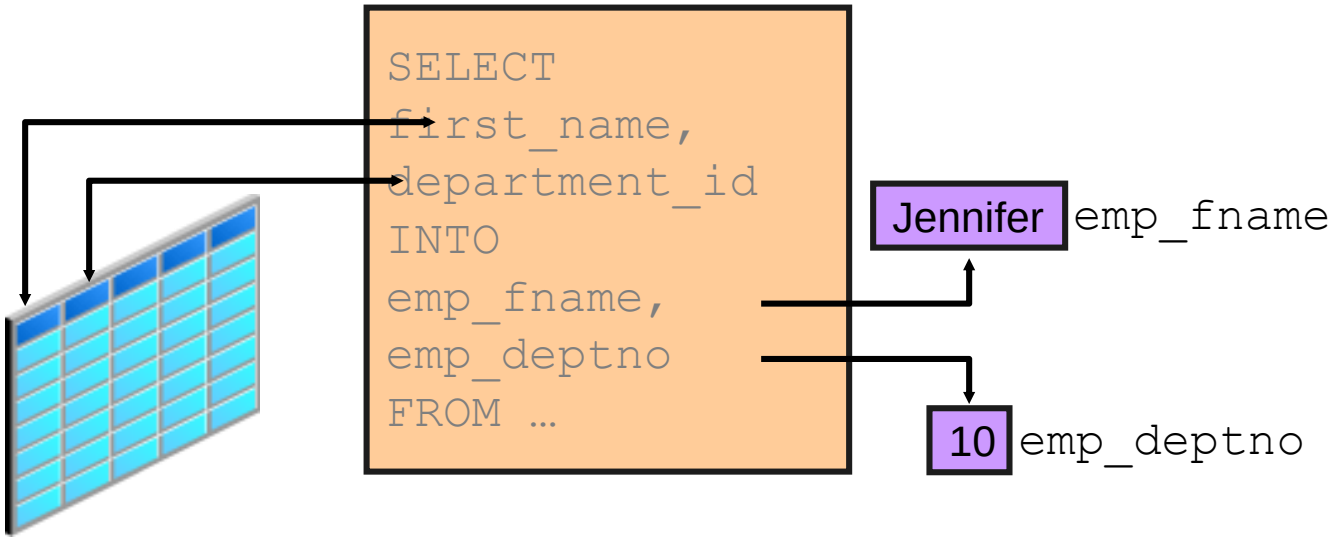
- This practice covers the following topics:
  - Identifying the PL/SQL blocks that execute successfully
  - Creating and executing a simple PL/SQL block

# Types of Variables

- PL/SQL variables:
  - Scalar
  - Composite
  - Reference
  - Large object (LOB)
- Non-PL/SQL variables: Bind variables

# Use of Variables

- Variables can be used for:
  - Temporary storage of data
  - Manipulation of stored values
  - Reusability



# Requirements for Variable Names

- A variable name:
  - Must start with a letter
  - Can include letters or numbers
  - Can include special characters (such as `$`, `_`, and `#` )
  - Must contain no more than 30 characters
  - Must not include reserved words

# Handling Variables in PL/SQL

- Variables are:
  - Declared and initialized in the declarative section
  - Used and assigned new values in the executable section
  - Passed as parameters to PL/SQL subprograms
  - Used to hold the output of a PL/SQL subprogram

# Declaring and Initializing PL/SQL Variables

Syntax

```
identifier [CONSTANT] datatype [NOT NULL]
     [:= | DEFAULT expr];
```

Examples

```
DECLARE
  emp_hiredate    DATE;
  emp_deptno      NUMBER(2) NOT NULL := 10;
  location        VARCHAR2(13) := 'Atlanta';
  c_comm          CONSTANT NUMBER := 1400;
```

# Declaring and Initializing PL/SQL Variables

1

```
SET SERVEROUTPUT ON
DECLARE
  Myname VARCHAR2(20);
BEGIN
  DBMS_OUTPUT.PUT_LINE('My name is: '||Myname);
  Myname := 'John';
  DBMS_OUTPUT.PUT_LINE('My name is: '||Myname);
END;
/
```

2

```
SET SERVEROUTPUT ON
DECLARE
  Myname VARCHAR2(20):= 'John';
BEGIN
  Myname := 'Steven';
  DBMS_OUTPUT.PUT_LINE('My name is: '||Myname);
END;
/
```

# Delimiters in String Literals

```
SET SERVEROUTPUT ON
DECLARE
   event VARCHAR2(15);
BEGIN
  event := q'!Father's day!';
  DBMS_OUTPUT.PUT_LINE('3rd Sunday in June is :
   '||event);
  event := q'[Mother's day]';
  DBMS_OUTPUT.PUT_LINE('2nd Sunday in May is :
   '||event);
END;
/
```

```
3rd Sunday in June is : Father's day
2nd Sunday in May is : Mother's day
PL/SQL procedure successfully completed.
```

# Guidelines for Declaring and Initializing PL/SQL Variables

- Follow naming conventions.
- Use meaningful names for variables.
- Initialize variables designated as `NOT NULL` and `CONSTANT`.
- Initialize variables with the assignment operator (`:=`) or the `DEFAULT` keyword:

```
Myname VARCHAR2(20):='John';
```

Declare one identifier per line for better readability and code maintenance.

```
Myname VARCHAR2(20) DEFAULT 'John';
```

# Guidelines for Declaring PL/SQL Variables

- Avoid using column names as identifiers.

```
DECLARE
  employee_id  NUMBER(6);
BEGIN
  SELECT    employee_id
  INTO      employee_id
  FROM      employees
  WHERE     last_name = 'Kochhar';
END;
/
```

- Use the `NOT NULL` constraint when the variable must hold a value.

# Identify Scalar Data Types

- Hold a single value
- Have no internal components

# Base Scalar Data Types

- CHAR [(maximum_length)]
- VARCHAR2 (maximum_length)
- LONG
- LONG RAW
- NUMBER [(precision, scale)]
- BINARY_INTEGER
- PLS_INTEGER
- BOOLEAN
- BINARY_FLOAT
- BINARY_DOUBLE

# Base Scalar Data Types

- DATE
- TIMESTAMP
- TIMESTAMP WITH TIME ZONE
- TIMESTAMP WITH LOCAL TIME ZONE
- INTERVAL YEAR TO MONTH
- INTERVAL DAY TO SECOND

# Declaring Scalar Variables

- Examples

```
DECLARE
  emp_job          VARCHAR2(9);
  count_loop       BINARY_INTEGER := 0;
  dept_total_sal   NUMBER(9,2) := 0;
  orderdate        DATE := SYSDATE + 7;
  c_tax_rate       CONSTANT NUMBER(3,2) := 8.25;
  valid            BOOLEAN NOT NULL := TRUE;
  ...
```

# %TYPE Attribute

- The %TYPE attribute
  - Is used to declare a variable according to:
    - A database column definition
    - Another declared variable
  - Is prefixed with:
    - The database table and column
    - The name of the declared variable

# Declaring Anchored Datatype with the %TYPE Attribute

## Syntax

```
identifier     table.column_name%TYPE;
```

- Examples

```
SET SERVEROUTPUT ON;
DECLARE
v_fname students.first_name%TYPE;
BEGIN
SELECT first_name INTO v_fname FROM students WHERE
stu_id =1;
DBMS_OUTPUT.PUT_LINE (v_fname);
END;
```

# Declaring Boolean Variables

- Only the values `TRUE`, `FALSE`, and `NULL` can be assigned to a Boolean variable.
- Conditional expressions use the logical operators `AND` and `OR` and the unary operator `NOT` to check the variable values.
- The variables always yield `TRUE`, `FALSE`, or `NULL`.
- Arithmetic, character, and date expressions can be used to return a Boolean value.

# Bind Variables

- Bind variables are:
  - Created in the environment
  - Also called *host* variables
  - Created with the `VARIABLE` keyword
  - Used in SQL statements and PL/SQL blocks
  - Accessed even after the PL/SQL block is executed
  - Referenced with a preceding colon

# Printing Bind Variables

- Example

```
VARIABLE emp_salary NUMBER
BEGIN
   SELECT salary  INTO :emp_salary
   FROM  employees WHERE employee_id = 178;
END;
/
PRINT emp_salary
SELECT first_name, last_name FROM employees
WHERE salary=:emp_salary;
```

# Printing Bind Variables

- Example

```
VARIABLE emp_salary NUMBER
SET AUTOPRINT ON
BEGIN
   SELECT salary  INTO :emp_salary
   FROM  employees WHERE employee_id = 178;
END;
/
```

# Substitution Variables

- Are used to get user input at run time
- Are referenced within a PL/SQL block with a preceding ampersand
- Are used to avoid hard-coding values that can be obtained at run time

```
VARIABLE emp_salary NUMBER
SET AUTOPRINT ON
DECLARE
  empno NUMBER(6):=&empno;
BEGIN
  SELECT salary  INTO :emp_salary
  FROM  employees WHERE employee_id = empno;
END;
/
```

# Substitution Variables

Input Required

Cancel Continue

Enter value for empno: 100

Cancel Continue

1

old 2: empno NUMBER(6):=&empno;
new 2: empno NUMBER(6):=100;
PL/SQL procedure successfully completed.

| EMP_SALARY |
| --- |
| 24000 |

2

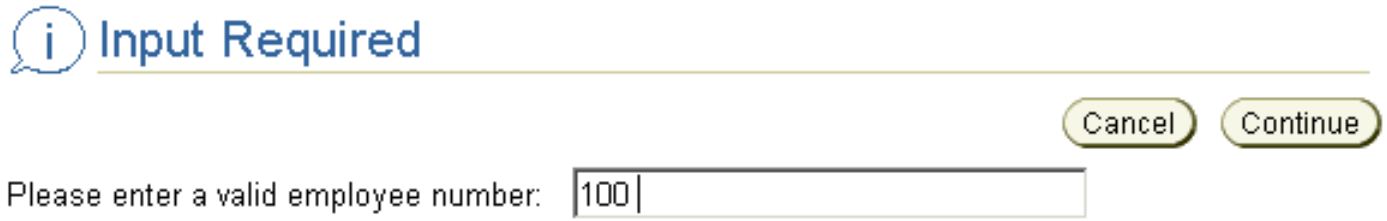
PL/SQL procedure successfully completed.

| EMP_SALARY |
| --- |
| 24000 |

3

# Prompt for Substitution Variables

```
SET VERIFY OFF
VARIABLE emp_salary NUMBER
ACCEPT empno PROMPT 'Please enter a valid employee
number: '
SET AUTOPRINT ON
DECLARE
  empno NUMBER(6):= &empno;
BEGIN
  SELECT salary  INTO :emp_salary  FROM  employees
  WHERE employee_id = empno;
END;
/
```



# Using DEFINE for a User Variable

- Example

```
SET VERIFY OFF
DEFINE lname= Urman
DECLARE
  fname VARCHAR2(25);
BEGIN
  SELECT first_name INTO fname FROM employees
  WHERE last_name='&lname';
END;
/
```