Oracle 11g – PL SQL

Introduction to PL/SQL

Objectives

- After completing this lesson, you should be able to do the following:
  - Explain the Overview of PL/SQL
  - Explain the benefits of PL/SQL
  - Identify the different types of PL/SQL blocks
  - Use *i*SQL*Plus as a development environment for PL/SQL
  - Output messages of Anonynous Block PL/SQL

Overview of PL/SQL

- PL/SQL is the procedural extension to SQL with design features of programming languages.

- Query and Data manipulation of SQL are included within procedural units of code.

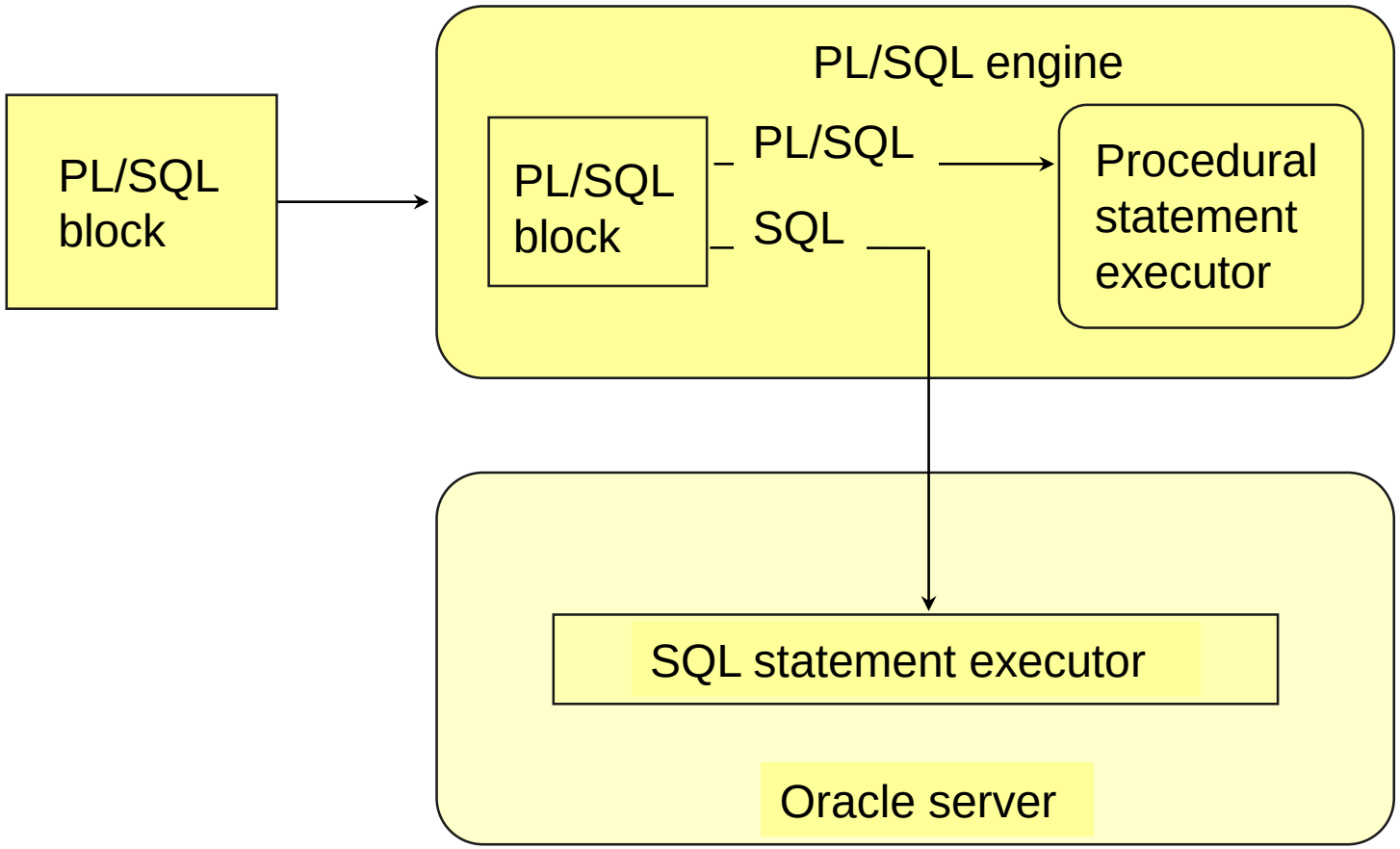
Overview of PL/SQL

- 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

Overview of PL/SQL

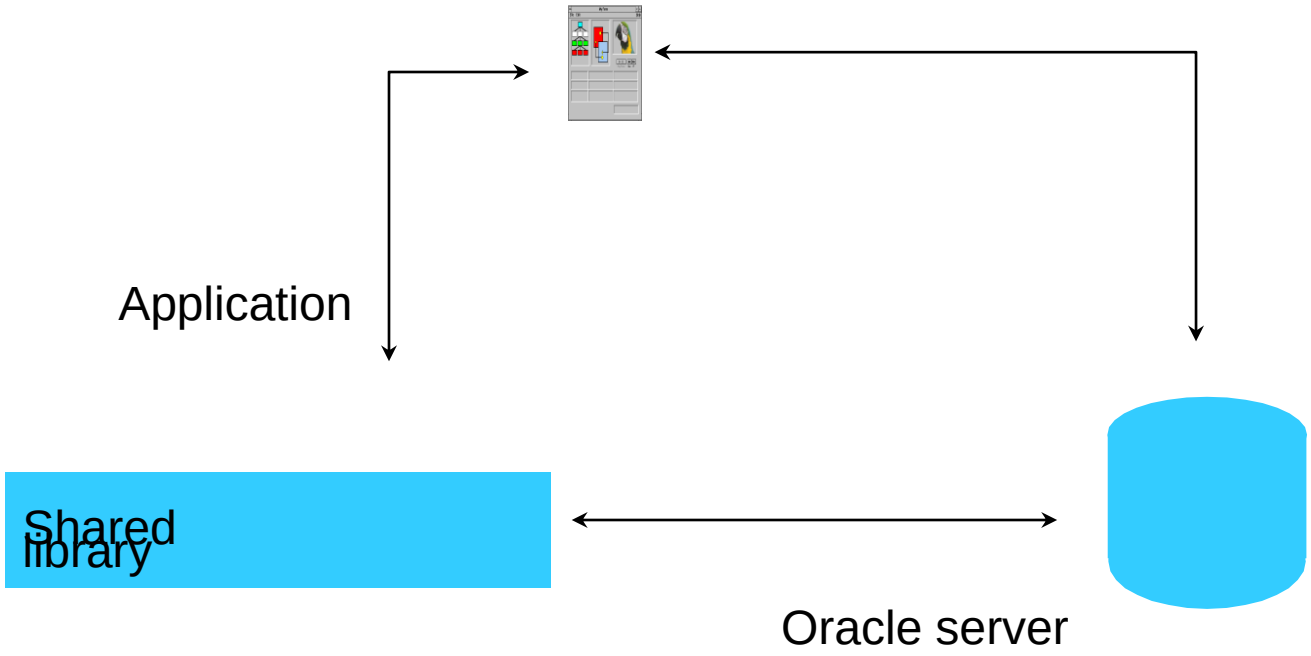
- PL/SQL:
  - Provides a block structure for executable units of code. Maintenance of code is made easier with such a well-defined structure.
  - Provides procedural constructs such as:
    - Variables, constants, and types
    - Control structures such as conditional statements and loops
    - Reusable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

Overview Architecture of PL/SQL



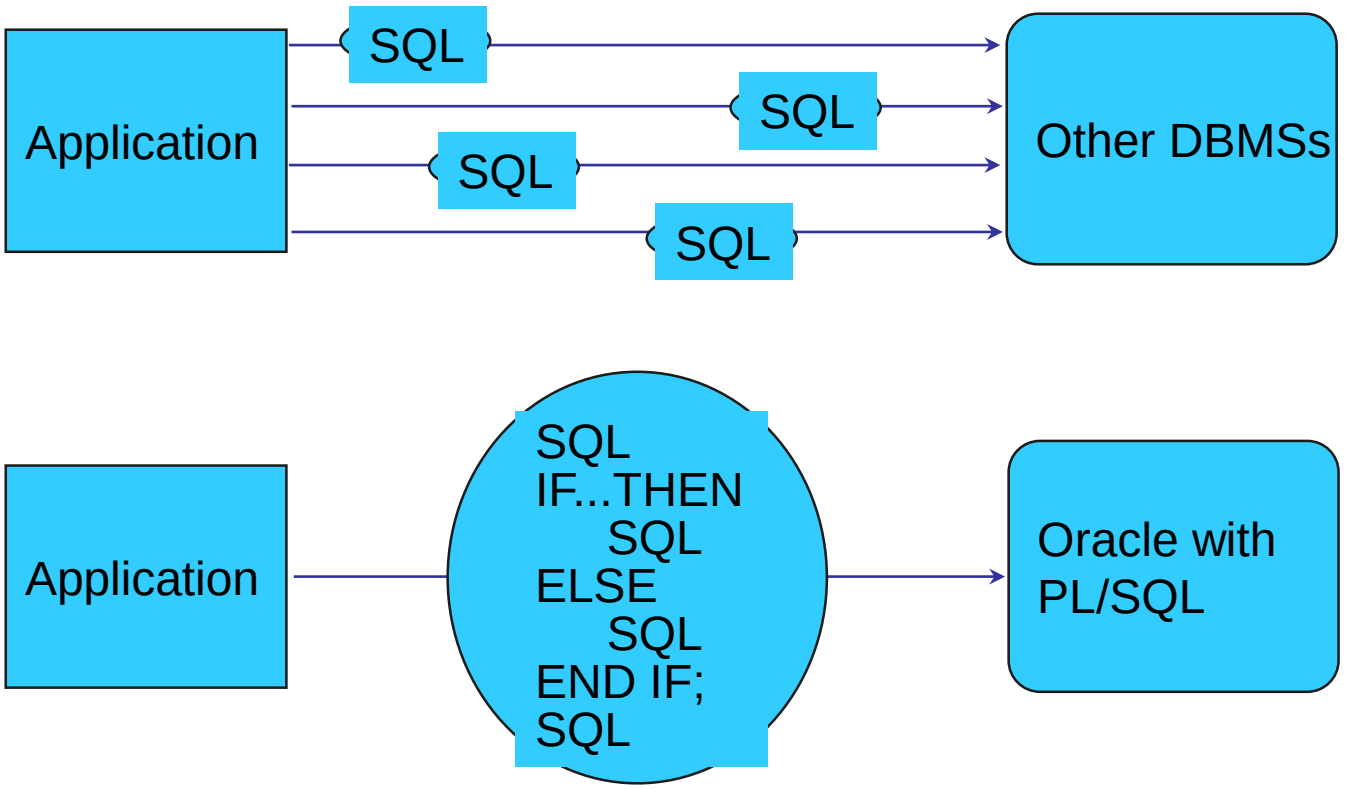
Benefits of PL/SQL

Integration



Benefits of PL/SQL

Improved performance



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)

Benefits of PL/SQL

1. PL/SQL is portable
2. You can declare variables
3. You can program with procedural language control structures.
4. PL/SQL can handle errors.

Types of PL/SQL Blocks

Anonymous

```
[DECLARE]

BEGIN
  --statements

[EXCEPTION]

END;
```

Procedure

```
PROCEDURE name
IS

BEGIN
  --statements

[EXCEPTION]

END;
```

Function

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

END;
```

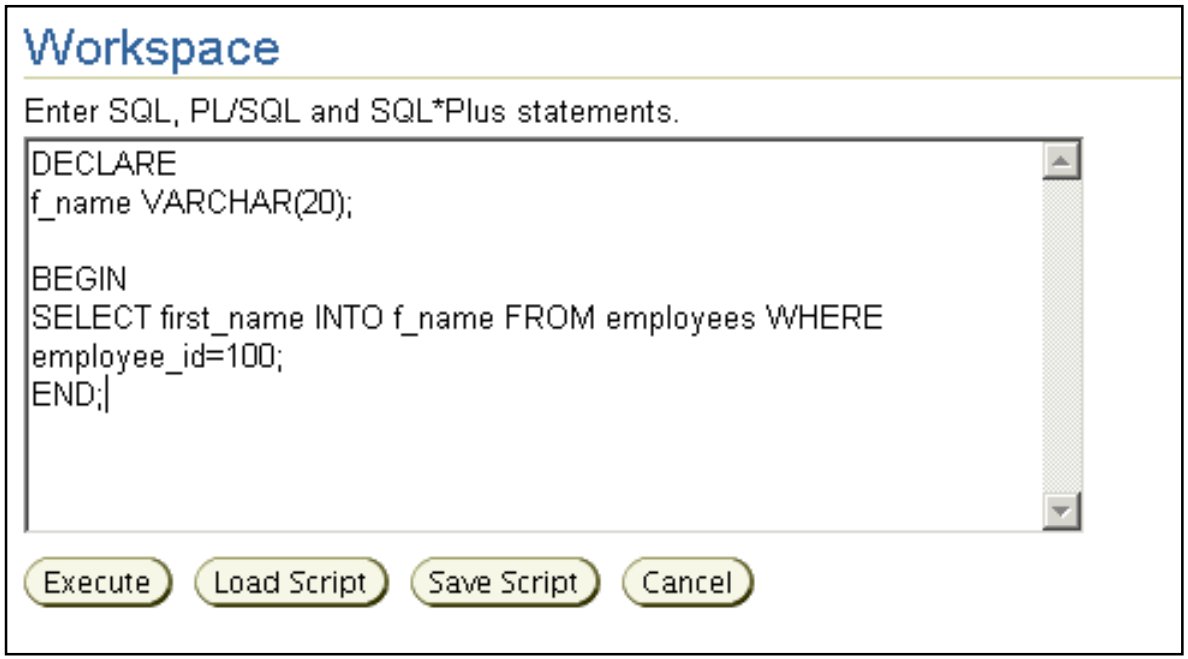
Block Constructs

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

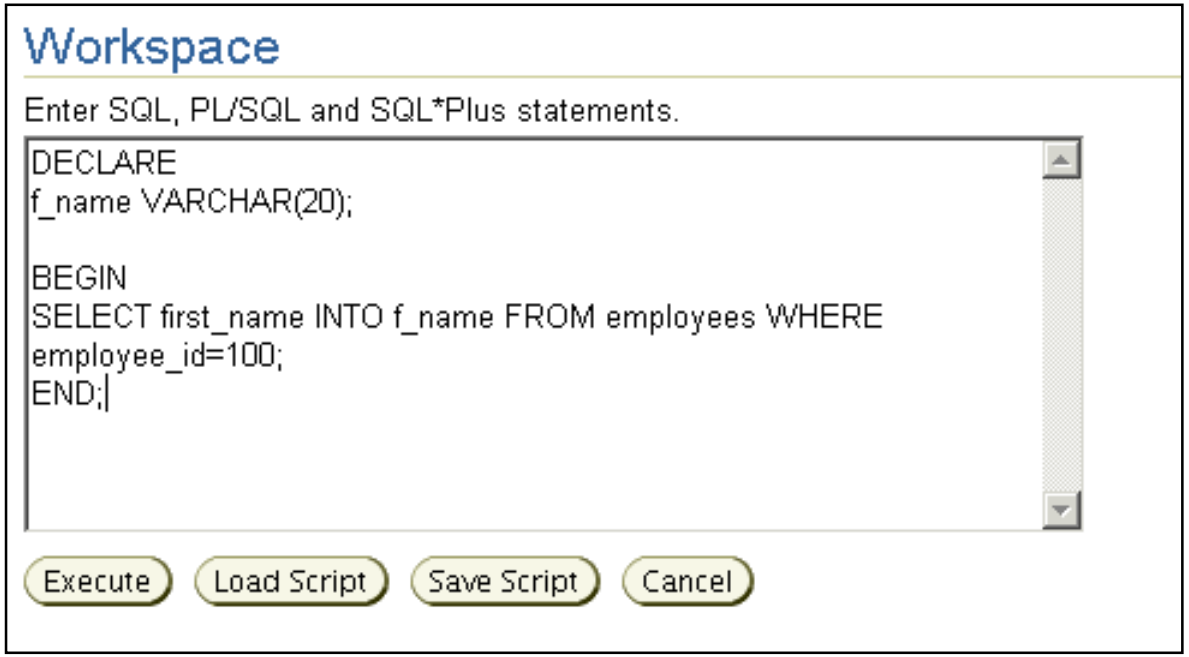
Create an Anonymous Block

- Type the anonymous block in the *i*SQL*Plus workspace:



Execute an Anonymous Block

- Click the Execute button to execute the anonymous block:



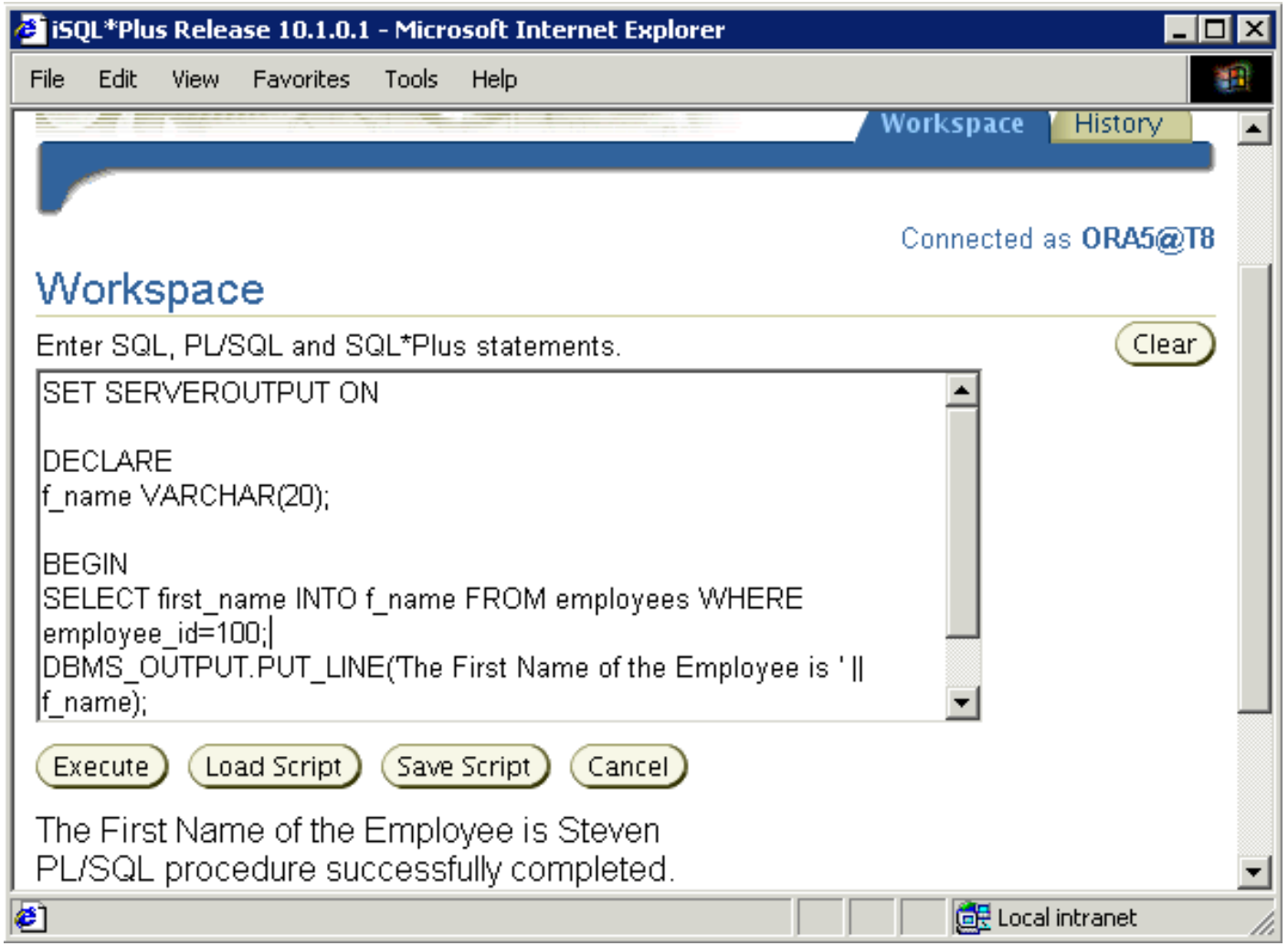
PL\SQL procedure successfully completed.

Output Anonymous PL/SQL Block

- Enable output in *i*SQL*Plus with the following command:
  `SET SERVEROUTPUT ON`
- Use a predefined Oracle package and its procedure:
  - `DBMS_OUTPUT.PUT_LINE`

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

Output Anonymous PL/SQL Block



Oracle 11g – PL SQL

Declaring PL/SQL Variables/Identifiers

Objectives

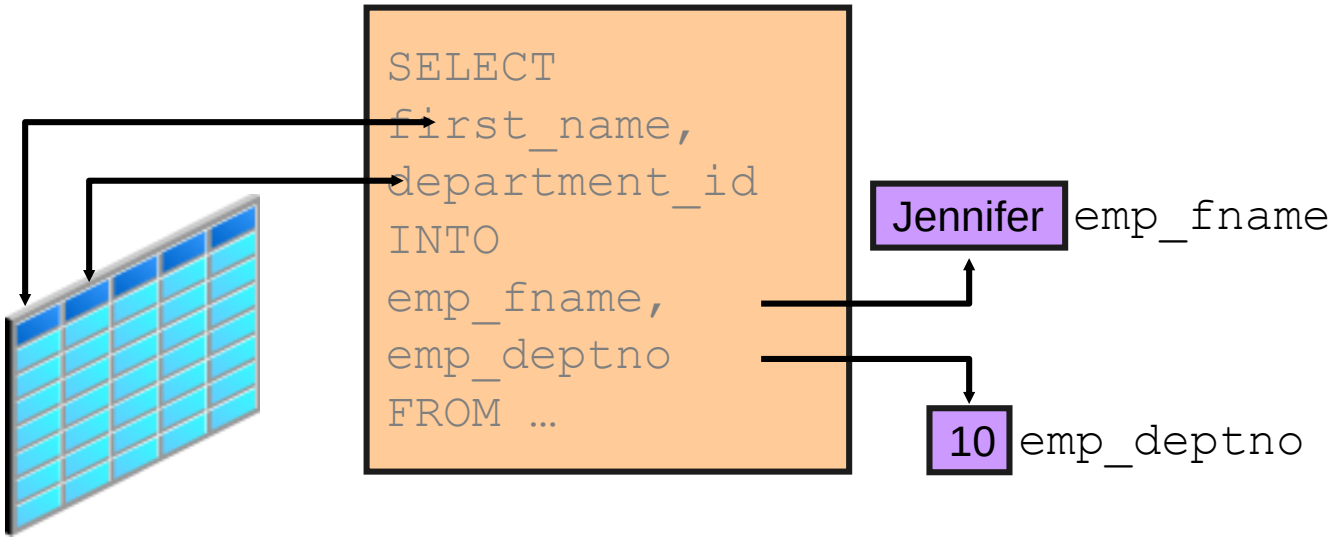
- After completing this lesson, you should be able to do the following:
  - Type of Variables/Identifier in a PLSQL Block
  - Use of Variables in PLSQL
  - Declare and initialize variables
  - Identify Scalar Data Types
  - List of various other data types
  - Identify the benefits of using the `%TYPE` attribute
  - Declare, use, and print bind variables

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



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 Variables with the %TYPE Attribute

Syntax

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

Examples

```
...
  emp_lname       employees.last_name%TYPE;
  balance         NUMBER(7,2);
  min_balance     balance%TYPE := 1000;
...
```

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

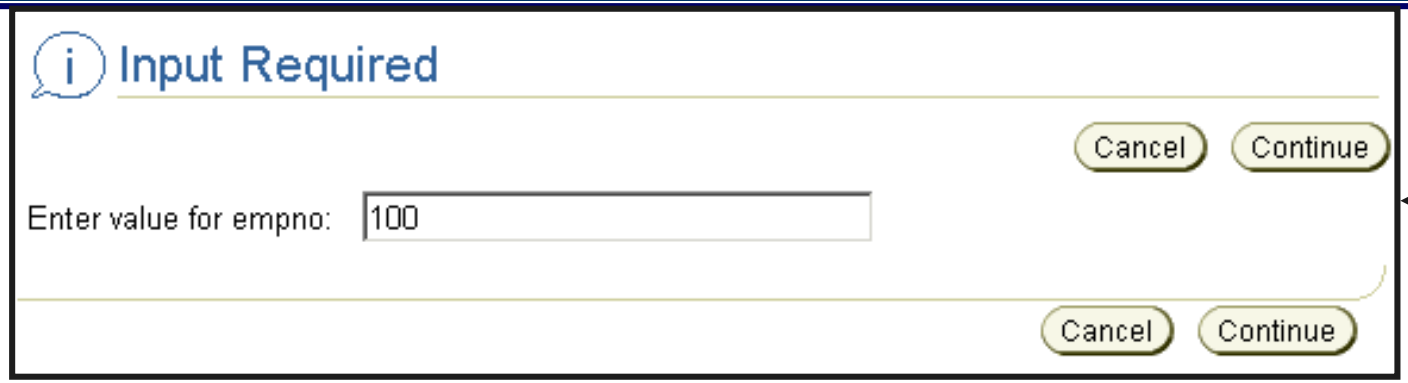
```
VARIABLE emp_salary NUMBER
SET AUTOPRINT ON
BEGIN
   SELECT salary  INTO :emp_salary
   FROM  employees WHERE employee_id = 178;
END;
/
```

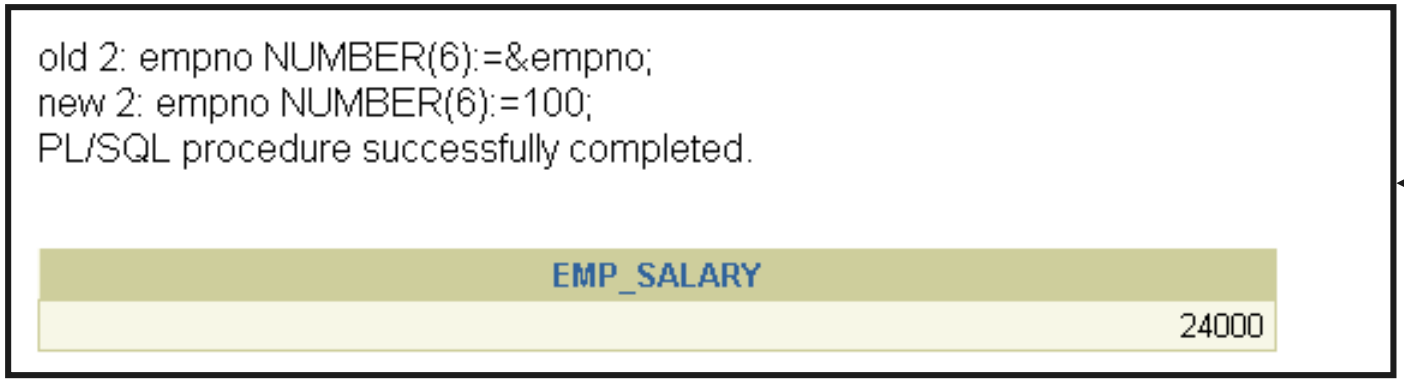
Substitution Variables

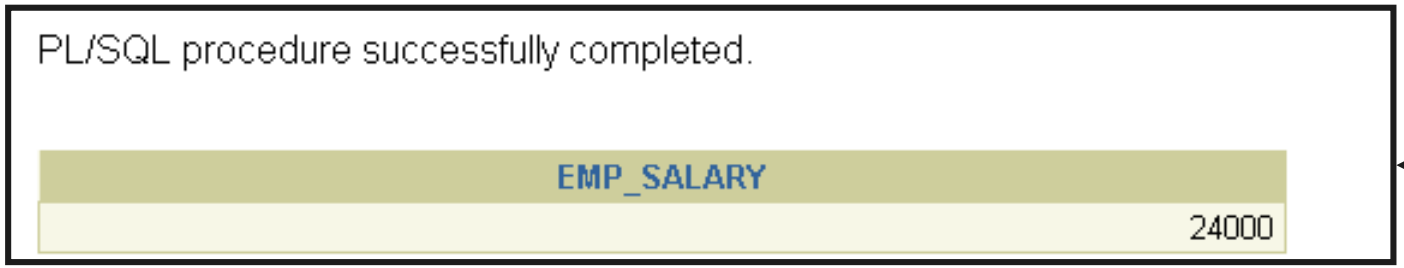
- Are used to get user input at run time
- Are referenced within a PL/SQL block with a preceding ampersand
- Are used to avoid hard-coding values that can be obtained at run time

```
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

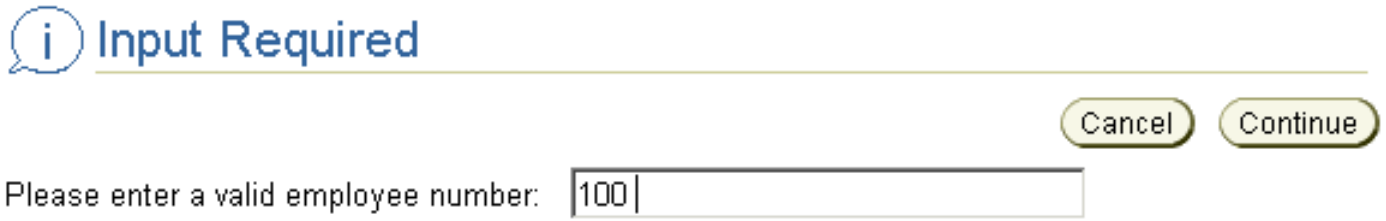






Prompt for Substitution Variables

```
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;
/
```