PL/SQL

Executable statements

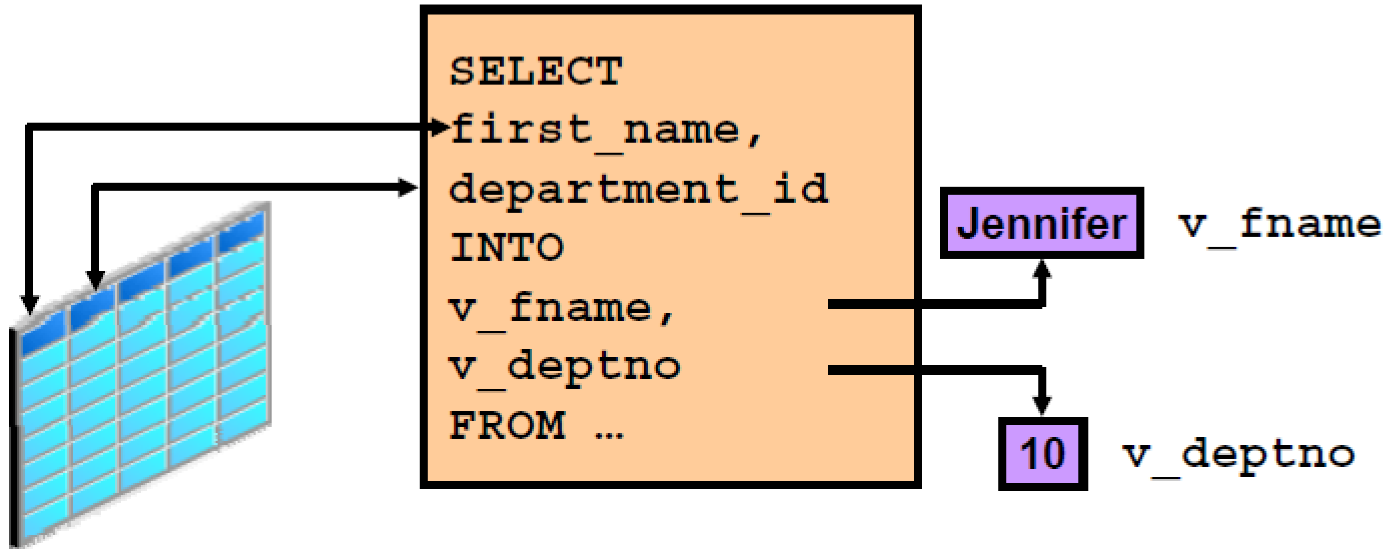
DE HOGESCHOOL MET HET NETWERK

Elfde-Liniestraat 24, 3500 Hasselt, www.pxl.be

Use of Variables

Variables can be used for:

- Temporary storage of data
- Manipulation of stored values
- Reusability



```
SELECT
first_name,
department_id
INTO
v_fname,
v_deptno
FROM …
```

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

Base Scalar Data Types

- CHAR [(maximum_length)]
- VARCHAR2 (maximum_length)
- NUMBER [(precision, scale)]
- BINARY_INTEGER
- PLS_INTEGER
- BOOLEAN
- BINARY_FLOAT
- BINARY_DOUBLE
- DATE
- TIMESTAMP
- TIMESTAMP WITH TIME ZONE
- TIMESTAMP WITH LOCAL TIME ZONE
- INTERVAL YEAR TO MONTH
- INTERVAL DAY TO SECOND

Declaring and Initializing PL/SQL Variables

Syntax:

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

Examples:

```
DECLARE
  v_hiredate      DATE;
  v_deptno        NUMBER(2) NOT NULL := 10;
  v_location      VARCHAR2(13) := 'Atlanta';
  c_comm          CONSTANT NUMBER := 1400;
```

Guidelines for Declaring and Initializing PL/SQL Variables

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

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

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

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

Declaring Scalar Variables

Examples:

```
DECLARE
  v_emp_job          VARCHAR2(9);
  v_count_loop       BINARY_INTEGER := 0;
  v_dept_total_sal   NUMBER(9,2) := 0;
  v_orderdate        DATE := SYSDATE + 7;
  c_tax_rate         CONSTANT NUMBER(3,2) := 8.25;
  v_valid            BOOLEAN NOT NULL := TRUE;
  ...
```

`%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 names
  - 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;
...
```

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.

Naming Conventions of PL/SQL

PL/SQL Structure	Convention	Example
Variable	`v_variable_name`	`v_rate`
Constant	`c_constant_name`	`c_rate`
Subprogram parameter	`p_parameter_name`	`p_id`
Bind (host) variable	`b_bind_name`	`b_salary`
Cursor	`cur_cursor_name`	`cur_emp`
Record	`rec_record_name`	`rec_emp`
Type	`type_name_type`	`ename_table_type`
Exception	`e_exception_name`	`e_products_invalid`
File handle	`f_file_handle_name`	f_file

Declaring and Initializing PL/SQL Variables

1

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

2

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

1. My name is:
   My name is : John
2. My name is : Steven

Commenting Code

- Prefix single-line comments with two hyphens (--).
- Place multiple-line comments between the symbols /* and */.

Example:

```
DECLARE
...
v_annual_sal NUMBER (9,2);
BEGIN
/* Compute the annual salary based on the
   monthly salary input from the user */
v_annual_sal := monthly_sal * 12;
--The following line displays the annual salary
DBMS_OUTPUT.PUT_LINE(v_annual_sal);
END;
/
```

Operators in PL/SQL

- Logical
- Arithmetic
- Concatenation
- Parentheses to control order of operations

Same as in SQL

- Exponential operator (**)

Operators in PL/SQL: Examples

- Increment the counter for a loop.

```
loop_count := loop_count + 1;
```

- Set the value of a Boolean flag.

```
good_sal := sal BETWEEN 50000 AND 150000;
```

- Validate whether an employee number contains a value.

```
valid  := (empno IS NOT NULL);
```

Programming Guidelines

Make code maintenance easier by:

- Documenting code with comments
- Developing a case convention for the code
- Developing naming conventions for identifiers and other objects
- Enhancing readability by indenting

Indenting Code

For clarity, indent each level of code.

```
BEGIN
  IF x=0 THEN
     y:=1;
  END IF;
END;
/
```

```
DECLARE
  deptno       NUMBER(4);
  location_id  NUMBER(4);
BEGIN
  SELECT  department_id,
          location_id
  INTO    deptno,
          location_id
  FROM    departments
  WHERE   department_name
          = 'Sales';
...
END;
/
```

SELECT Statements in PL/SQL

Retrieve data from the database with a SELECT statement.

Syntax:

```
SELECT  select_list
INTO    {variable_name[, variable_name]...
        | record_name}
FROM    table
[WHERE  condition];
```

SELECT Statements in PL/SQL

- The INTO clause is required.
- Queries must return only one row.

```
DECLARE
 v_fname VARCHAR2(25);
BEGIN
 SELECT first_name INTO v_fname
 FROM employees WHERE employee_id=200;
 DBMS_OUTPUT.PUT_LINE(' First Name is : '||v_fname);
END;
/
```

```
anonymous block completed
 First Name is : Jennifer
```

Retrieving Data in PL/SQL: Example

Retrieve `hire_date` and `salary` for the specified employee.

```
DECLARE
 v_emp_hiredate   employees.hire_date%TYPE;
 v_emp_salary     employees.salary%TYPE;
BEGIN
  SELECT   hire_date, salary
  INTO     v_emp_hiredate, v_emp_salary
  FROM     employees
  WHERE    employee_id = 100;
END;
/
```

Retrieving Data in PL/SQL

Return the sum of the salaries for all the employees in the specified department.

Example:

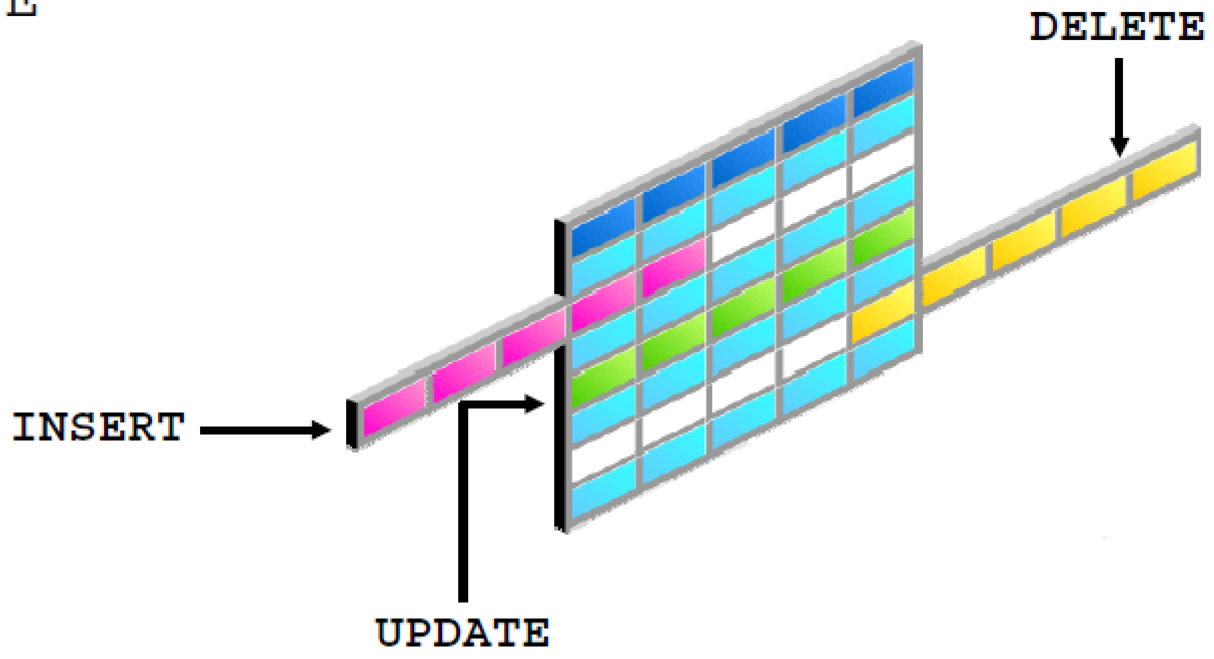
```
DECLARE
   v_sum_sal   NUMBER(10,2);
   v_deptno    NUMBER NOT NULL := 60;
BEGIN
  SELECT  SUM(salary)  -- group function
  INTO v_sum_sal  FROM employees
  WHERE     department_id = v_deptno;
  DBMS_OUTPUT.PUT_LINE ('The sum of salary is ' || v_sum_sal);
END;
```

```
anonymous block completed
The sum of salary is 28800
```

Using PL/SQL to Manipulate Data

Make changes to database tables by using DML commands:

- INSERT
- UPDATE
- DELETE



Inserting Data: Example

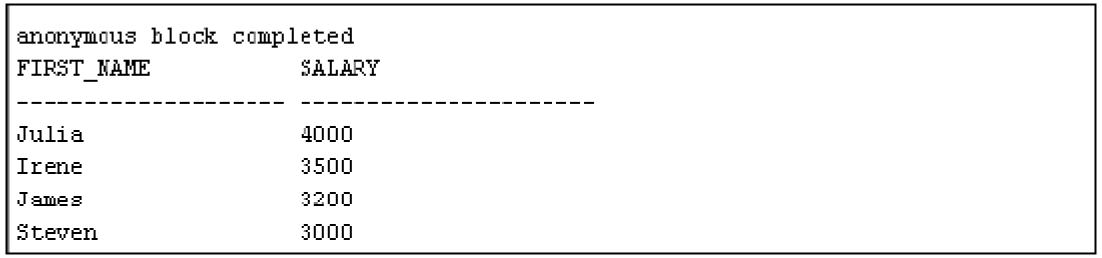
Add new employee information to the EMPLOYEES table.

```
BEGIN
 INSERT INTO employees
  (employee_id, first_name, last_name, email,
   hire_date, job_id, salary)
   VALUES(employees_seq.NEXTVAL, 'Ruth', 'Cores',
   'RCORES',CURRENT_DATE, 'AD_ASST', 4000);
END;
/
```

Updating Data: Example

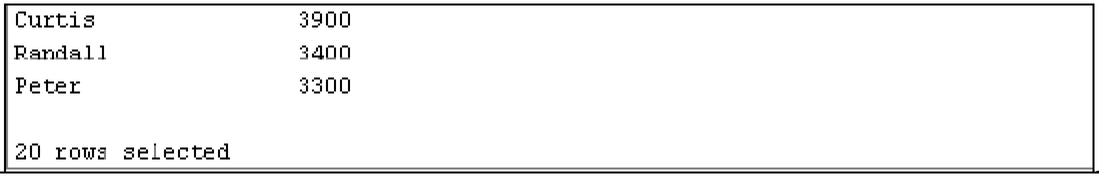
Increase the salary of all employees who are stock clerks.

```
DECLARE
  sal_increase   employees.salary%TYPE := 800;
BEGIN
  UPDATE     employees
  SET        salary = salary + sal_increase
  WHERE      job_id = 'ST_CLERK';
END;
/
```



```
anonymous block completed
FIRST_NAME           SALARY
-------------------- ----------------------
Julia                4000
Irene                3500
James                3200
Steven               3000
```

. . .



```
Curtis               3900
Randall              3400
Peter                3300

20 rows selected
```

Deleting Data: Example

Delete rows that belong to department 10 from the `employees` table.

```
DECLARE
  deptno   employees.department_id%TYPE := 10;
BEGIN
  DELETE FROM   employees
  WHERE  department_id = deptno;
END;
/
```