COMP322003 Introduction to Database (Fall 2018)

KNU CSE

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

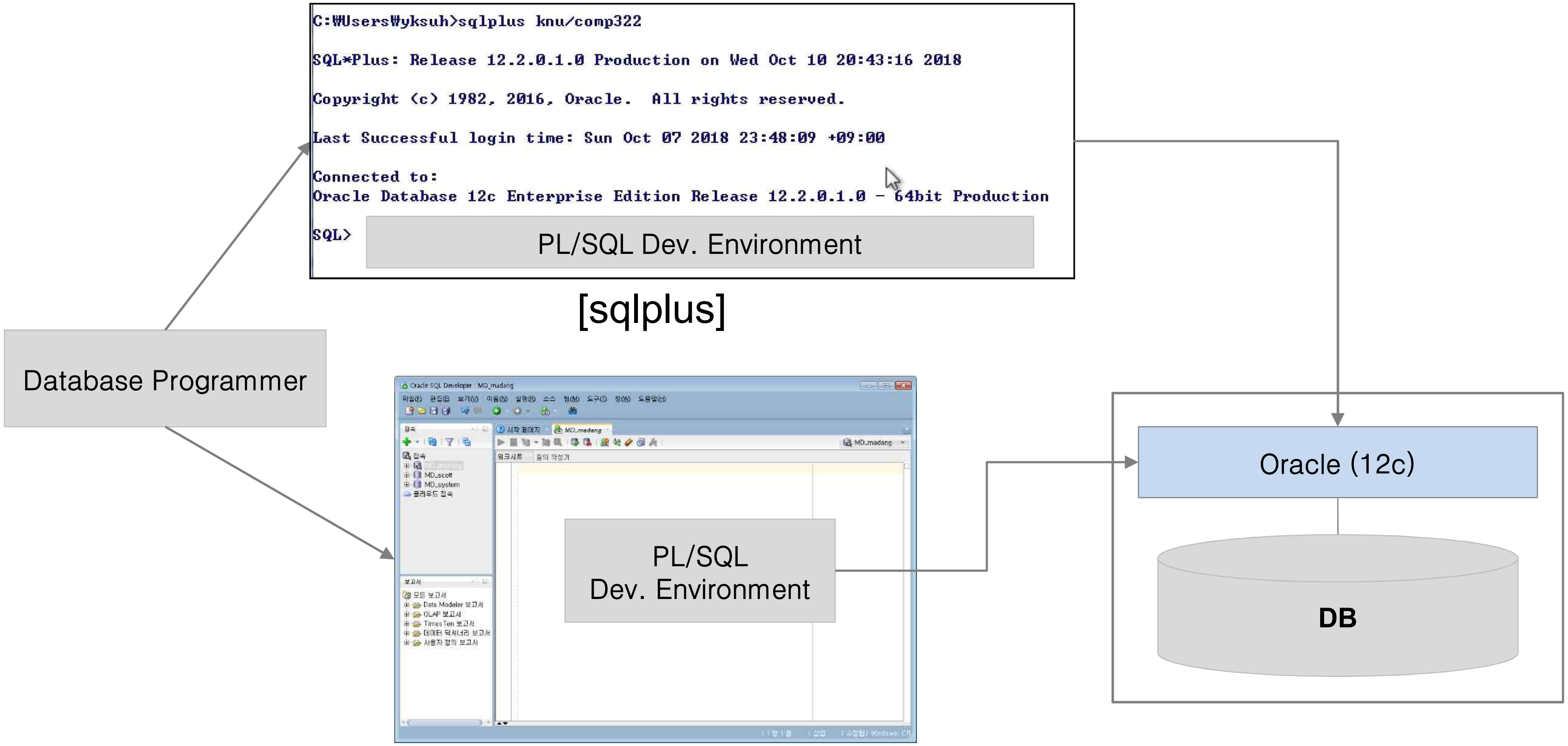
LAB 7: PL/SQL

- 실습교재: Chapter 5.2
- 실습 부교재: Chapter 12

Procedural Language/Structured Query Language (PL/SQL)

- Oracle-exclusive language used for DB application programs
- Groups several SQL statements into a *single* block and sends to the database server the entire block via a single invocation and thus improves the performance
- Variables, loops, control structures are supported as in a general-purpose language (GPL).
  - Thus, a problem that can’t be processed by SQL alone may be solved.
- Exception handling during execution is supported.
- Provides a functionality of building procedure or function to perform specialized functions
- A PL/SQL program is a lot faster than that of GPL, as every code is created and processed within the DBMS internal. (c.f. JDBC)

PL/SQL Development Environment



```
C:\Users\yksuh>sqlplus knu/comp322

SQL*Plus: Release 12.2.0.1.0 Production on Wed Oct 10 20:43:16 2018

Copyright (c) 1982, 2016, Oracle.  All rights reserved.

Last Successful login time: Sun Oct 07 2018 23:48:09 +09:00

Connected to:
Oracle Database 12c Enterprise Edition Release 12.2.0.1.0 - 64bit Production

SQL>
```

[sqlplus]

[SQL Developer]

- PL/SQL can be written and executed directly on sqlplus or SQL developer.

Major Constructs of PL/SQL:

https://docs.oracle.com/database/121/LNPLS/toc.htm

Construct	Command

Data Definition Language (from SQL)	CREATE TABLE, **CREATE PROCEDURE, CREATE FUNCTION,**
	CREATE TRIGGER, ALTER, DROP

Data Manipulation Language (from SQL)	SELECT, INSERT, DELETE, UPDATE
Data Types (from SQL)	NUMBER(n), VARCHAR2(n), DATE
Variables	**DECLARE** for declaring a variable := for assignment
Operator	Arithmetic operator: +, -, *, / Comparison operator: =, <, >, >=, <=, <> String (concatenation) operator: || Logical operator: NOT, AND, OR
Language Element (Comment) (partially from SQL)	- -, /* */
Built-in Function (partially from SQL)	Numeric: ABS, CEIL, FLOOR, POWER Aggregate: AVG, COUNT, MAX, MIN, SUM Date: SYSDATE, NEXT_DAY, TO_CHAR String: CHR, LENGTH, LOWER, SUBSTR

Cursor	**DECLARE**
	  CURSOR *cursor_name* **IS**...
	OPEN *cursor_name*;
	LOOP
	  FETCH *cursor_name* **INTO** *var1*, *var2*, …;
	  EXIT WHEN …;
	END LOOP;
	CLOSE *cursor_name*;

Control of Flow	**BEGIN-END, IF-THEN-ELSE, FOR LOOP-END LOOP,** **WHILE LOOP-END LOOP, EXIT**
Data Control Language (from SQL)	GRANT, REVOKE

Block in PL/SQL

- A basic unit that can be logically divided in a program
  - Such a block is the minimum unit for processing in PL/SQL.

Syntax

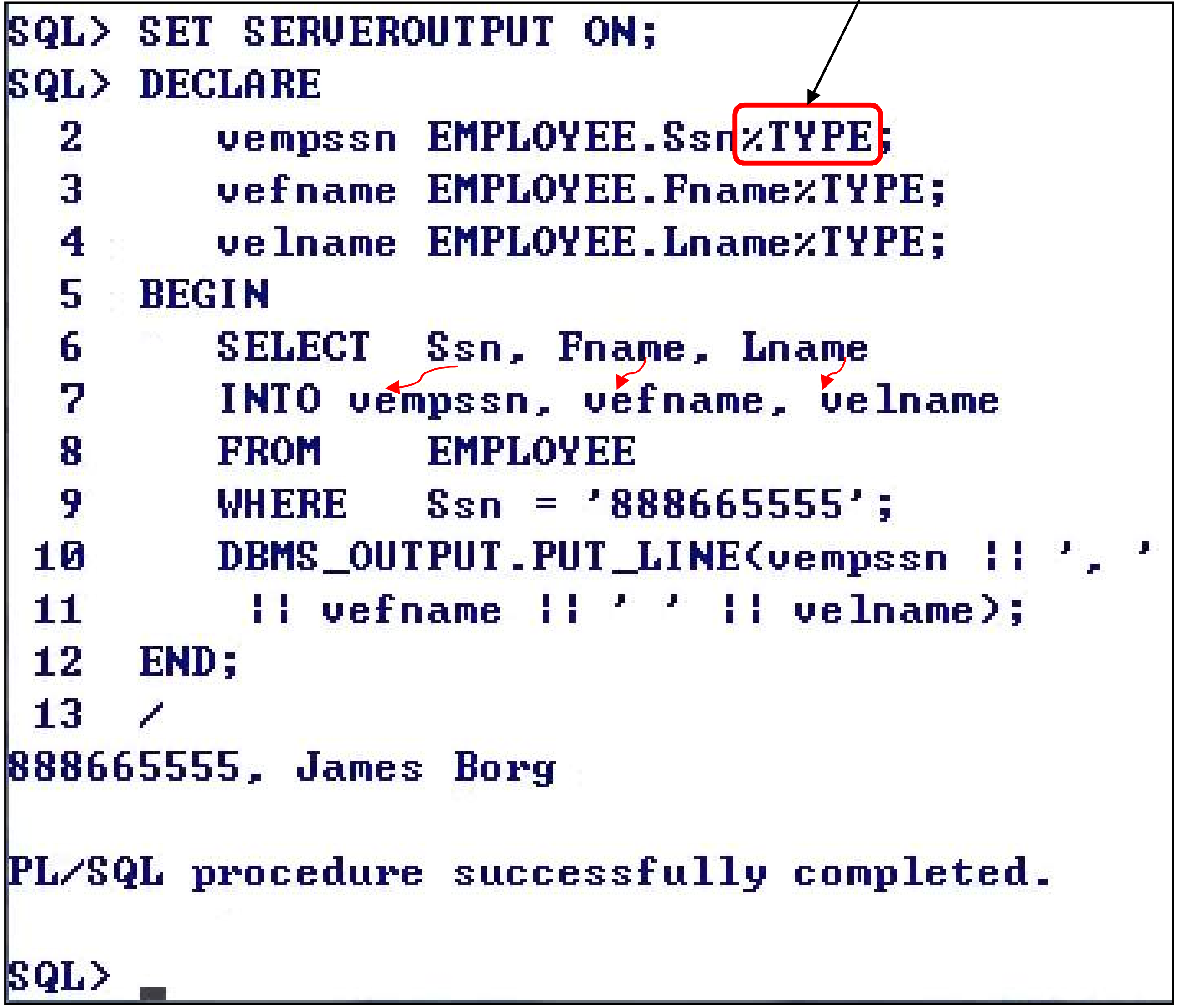
```
[DECLARE    -- Declare section
  (Scalar) Variables, Cursor, User-defined exception]

BEGIN       -- Executable section
  SQL statements, PL/SQL statement

[EXCEPTION  -- Exception section
  Task to be performed when an error occurs]

END;
```

Variables and Types in PL/SQL

- Same concept as that of GPL
- Should be declared in the declare section and used in the executable section

Syntax
DECLARE *Variable_name* [CONSTANT] *Data_type* [NOT NULL] [:= *Default_value*]

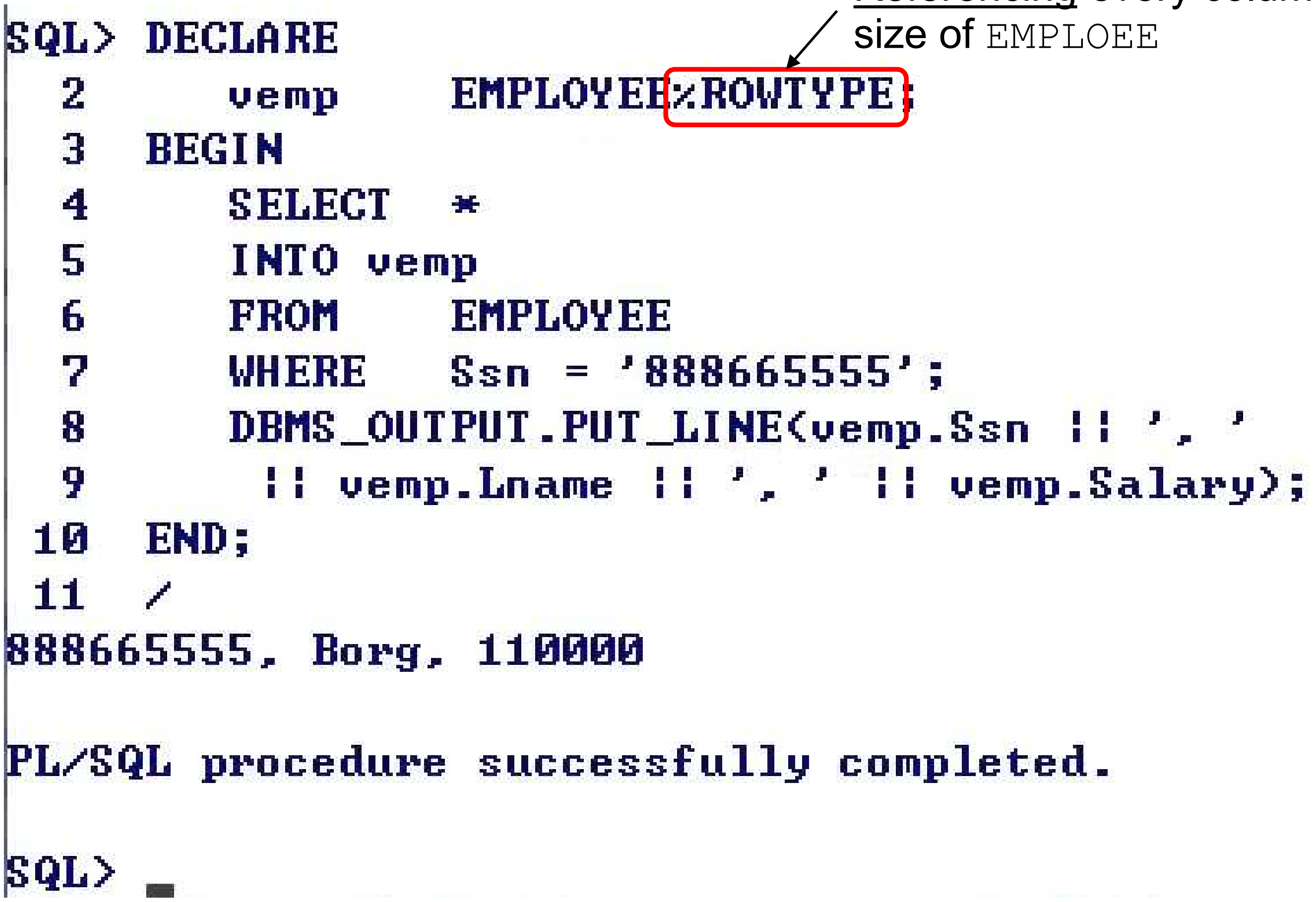
A Simple Example



```
SQL> SET SERVEROUTPUT ON;
SQL> DECLARE
  2       vempssn EMPLOYEE.Ssn%TYPE;
  3       vefname EMPLOYEE.Fname%TYPE;
  4       velname EMPLOYEE.Lname%TYPE;
  5  BEGIN
  6       SELECT  Ssn, Fname, Lname
  7       INTO vempssn, vefname, velname
  8       FROM    EMPLOYEE
  9       WHERE   Ssn = '888665555';
 10       DBMS_OUTPUT.PUT_LINE(vempssn || ', '
 11         || vefname || ' ' || velname);
 12  END;
 13  /
888665555, James Borg

PL/SQL procedure successfully completed.

SQL> _
```

<u>Referencing</u> the data type associated with `EMPLOYEE.Ssn`

Another Simple Example



Referencing every column type and size of EMPLOEE

```
SQL> DECLARE
  2      vemp    EMPLOYEE%ROWTYPE;
  3  BEGIN
  4      SELECT  *
  5      INTO vemp
  6      FROM    EMPLOYEE
  7      WHERE   Ssn = '888665555';
  8      DBMS_OUTPUT.PUT_LINE(vemp.Ssn || ', '
  9       || vemp.Lname || ', ' || vemp.Salary);
 10  END;
 11  /
888665555, Borg, 110000

PL/SQL procedure successfully completed.

SQL> _
```

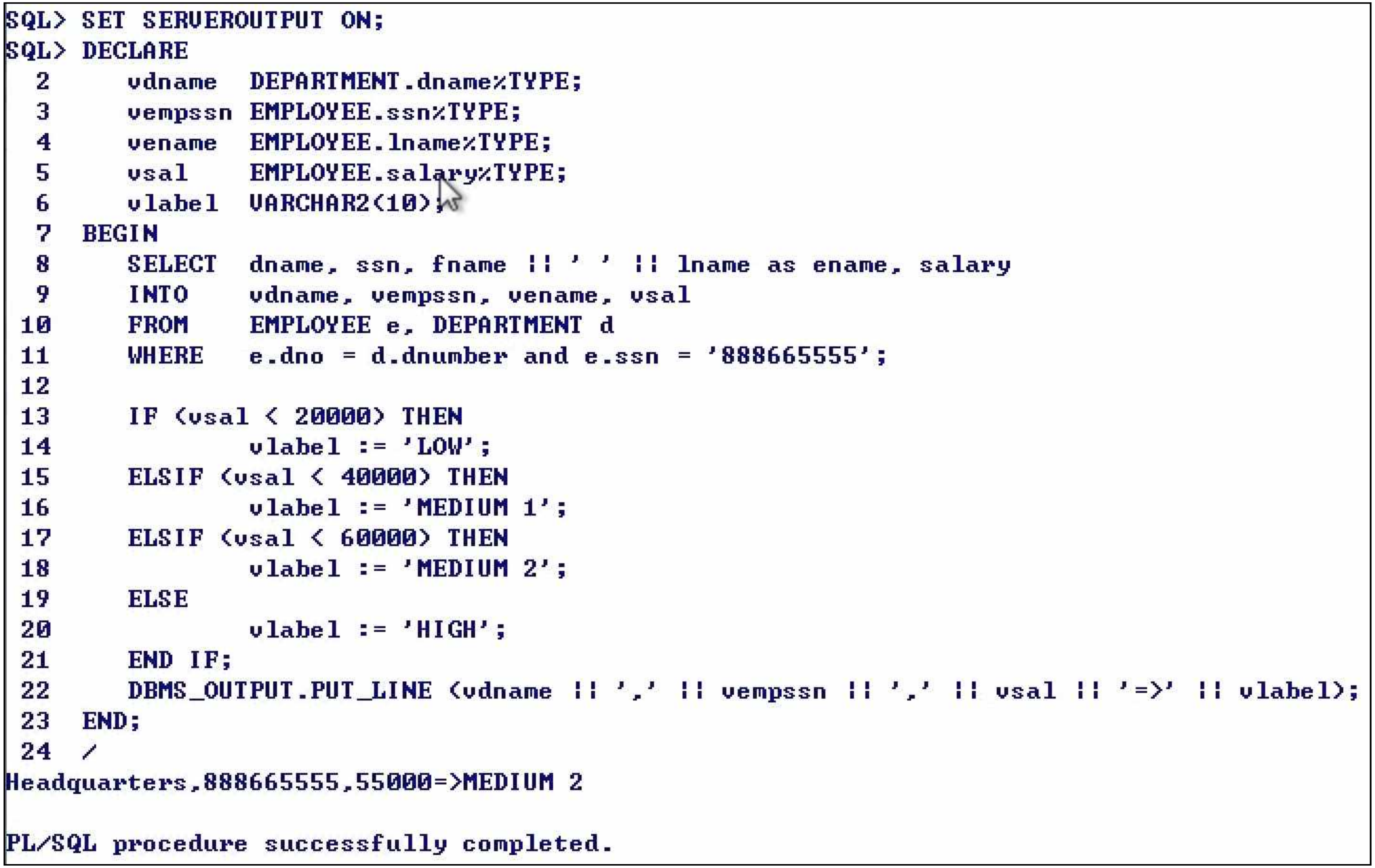
Control Flow: `IF` Statement Family

- Similar to the usage of `IF` (`THEN ELSE`) statement in GPL

Syntax

`IF condition_expr1 THEN`
`   Statement1;`
`[ELSIF condition_expr2 THEN`
`   Statement2;`
`ELSE`
`   Statement3;]`
`END IF;`

Control Flow: `IF` Example



```
SQL> SET SERVEROUTPUT ON;
SQL> DECLARE
  2     vdname   DEPARTMENT.dname%TYPE;
  3     vempssn  EMPLOYEE.ssn%TYPE;
  4     vename   EMPLOYEE.lname%TYPE;
  5     vsal     EMPLOYEE.salary%TYPE;
  6     vlabel   VARCHAR2(10);
  7  BEGIN
  8     SELECT   dname, ssn, fname || ' ' || lname as ename, salary
  9     INTO     vdname, vempssn, vename, vsal
 10     FROM     EMPLOYEE e, DEPARTMENT d
 11     WHERE    e.dno = d.dnumber and e.ssn = '888665555';
 12
 13     IF (vsal < 20000) THEN
 14             vlabel := 'LOW';
 15     ELSIF (vsal < 40000) THEN
 16             vlabel := 'MEDIUM 1';
 17     ELSIF (vsal < 60000) THEN
 18             vlabel := 'MEDIUM 2';
 19     ELSE
 20             vlabel := 'HIGH';
 21     END IF;
 22     DBMS_OUTPUT.PUT_LINE (vdname || ',' || vempssn || ',' || vsal || '=>' || vlabel);
 23  END;
 24  /
Headquarters,888665555,55000=>MEDIUM 2

PL/SQL procedure successfully completed.
```

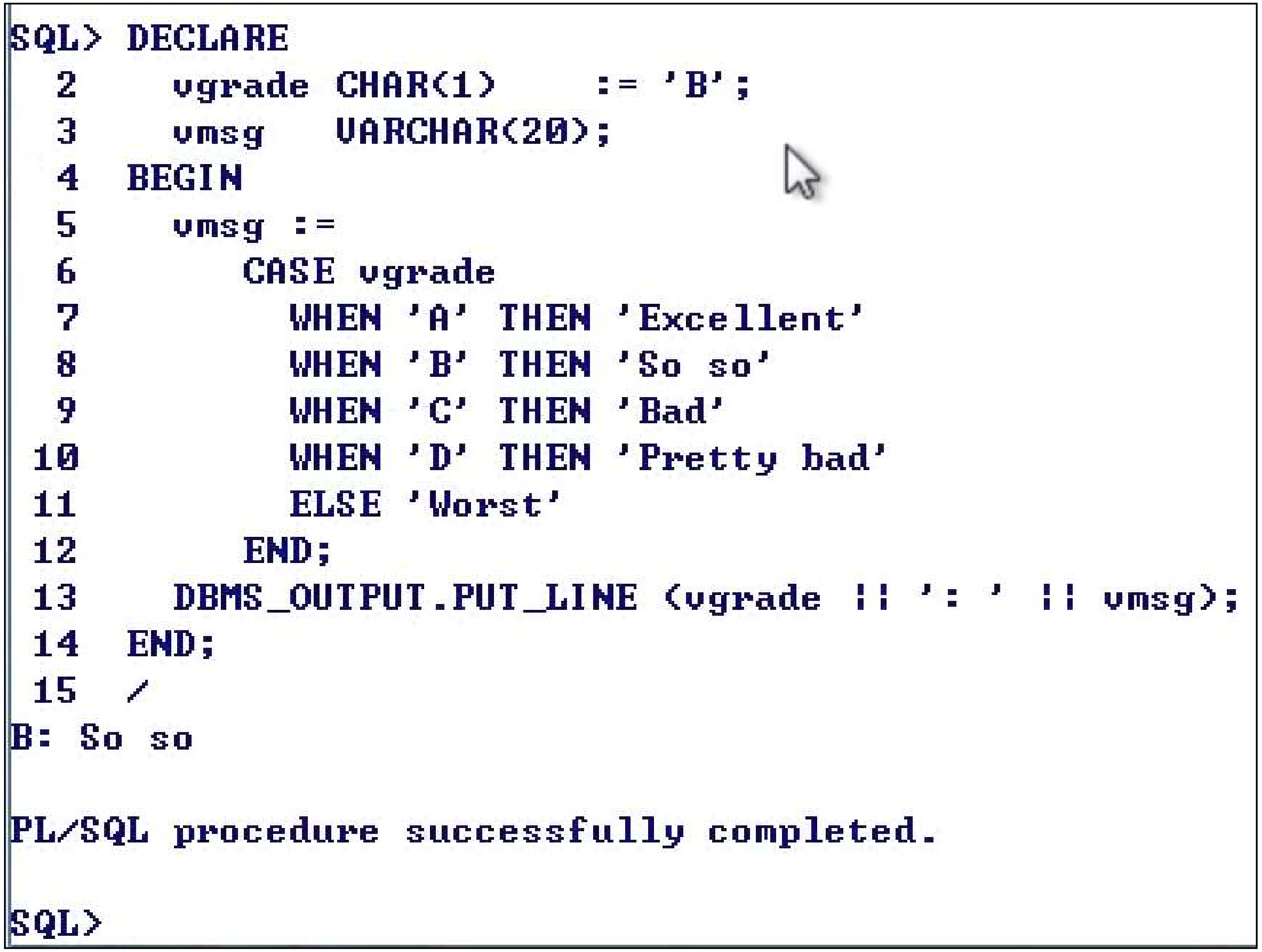
Control Flow: `CASE-WHEN` Statement

- Similar to `CASE-WHEN` in SQL

Syntax

```
CASE  expr1
   WHEN  value1 THEN  result1 ;
   WHEN  value2 THEN  result2 ;
   ...
   [ELSE  result3];
END IF;
```

Control Flow: CASE Example



```
SQL> DECLARE
       vgrade CHAR(1)     := 'B';
       vmsg   VARCHAR(20);
     BEGIN
       vmsg :=
          CASE vgrade
            WHEN 'A' THEN 'Excellent'
            WHEN 'B' THEN 'So so'
            WHEN 'C' THEN 'Bad'
            WHEN 'D' THEN 'Pretty bad'
            ELSE 'Worst'
          END;
       DBMS_OUTPUT.PUT_LINE (vgrade || ': ' || vmsg);
     END;
     /
B: So so

PL/SQL procedure successfully completed.

SQL>
```

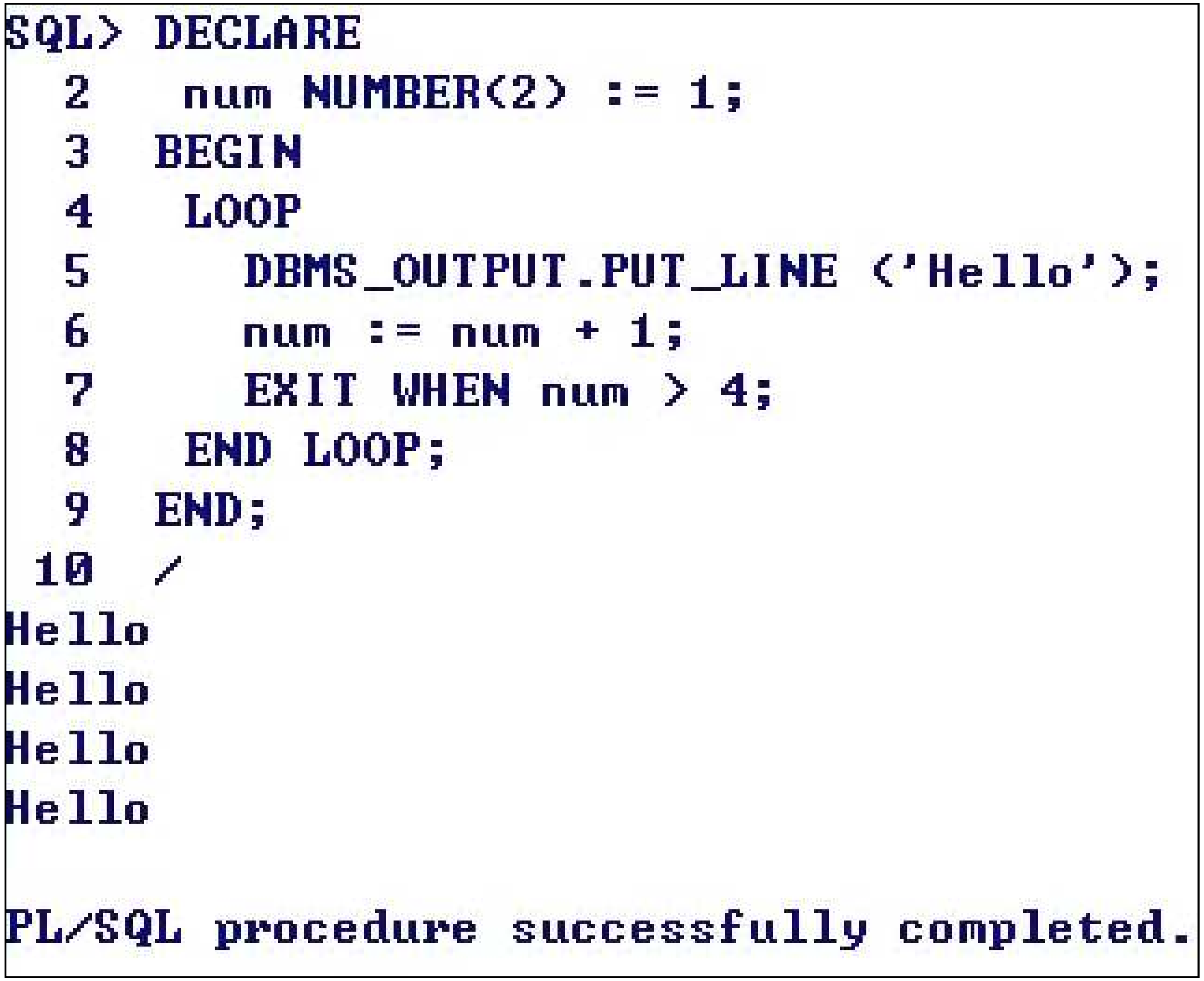
Control Flow: `LOOP` Statement

- Similar to `do-while` in GPL

Syntax

```
LOOP
   statement;
   ...
   EXIT [WHEN condition];
END LOOP;
```

Control Flow: `LOOP` Example



```
SQL> DECLARE
  2   num NUMBER(2) := 1;
  3  BEGIN
  4   LOOP
  5     DBMS_OUTPUT.PUT_LINE ('Hello');
  6     num := num + 1;
  7     EXIT WHEN num > 4;
  8   END LOOP;
  9  END;
 10  /
Hello
Hello
Hello
Hello

PL/SQL procedure successfully completed.
```

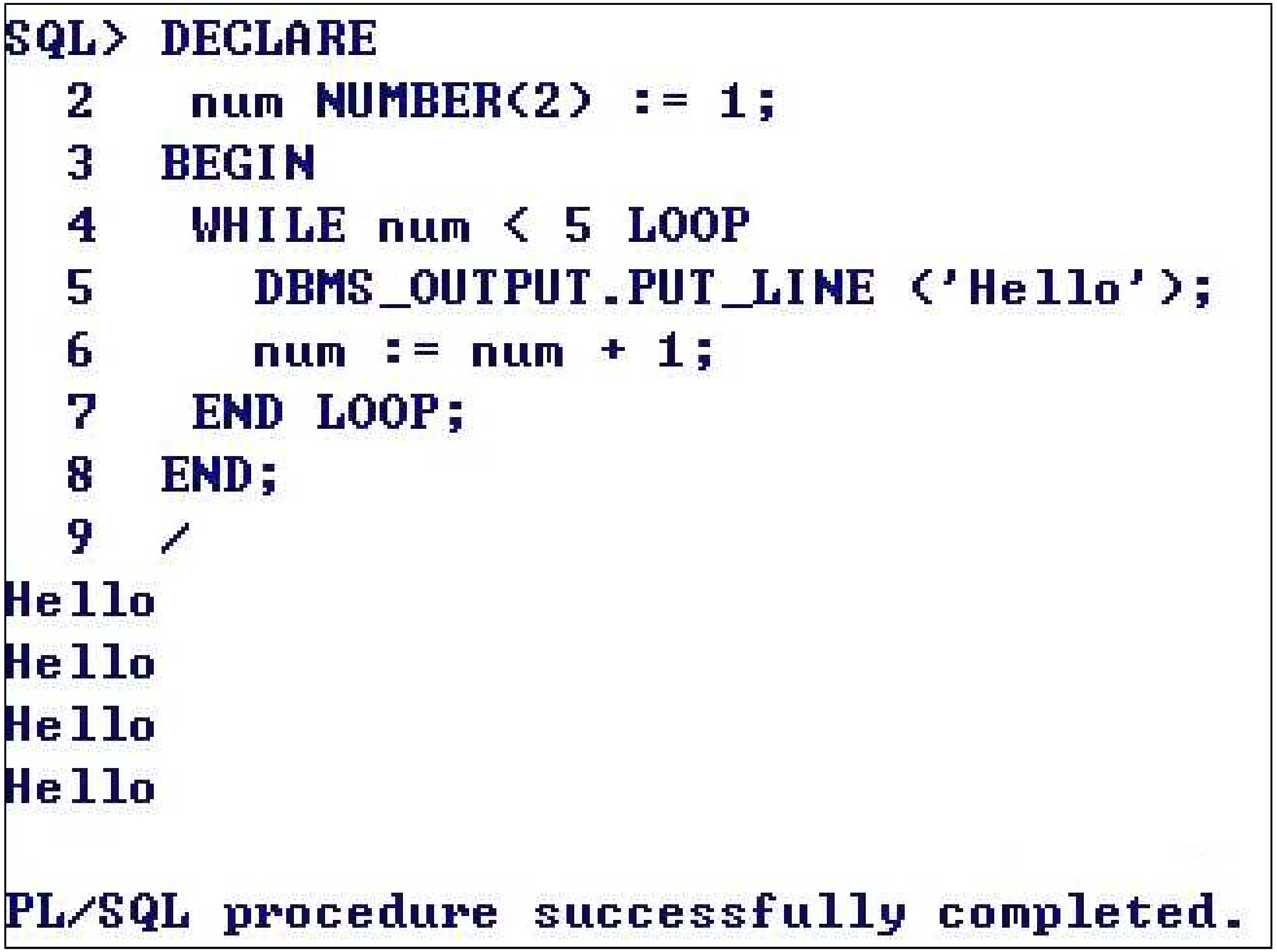
Control Flow: `WHILE` Statement

- Similar to `WHILE` in GPL

Syntax

```
WHILE condition LOOP
    statement1;
    statement2;
    ...
END LOOP;
```

Control Flow: `WHILE` Example



```
SQL> DECLARE
  2    num NUMBER(2) := 1;
  3   BEGIN
  4    WHILE num < 5 LOOP
  5      DBMS_OUTPUT.PUT_LINE ('Hello');
  6      num := num + 1;
  7    END LOOP;
  8   END;
  9   /
Hello
Hello
Hello
Hello

PL/SQL procedure successfully completed.
```

Control Flow: `FOR` Statement

- Similar to `FOR` in GPL

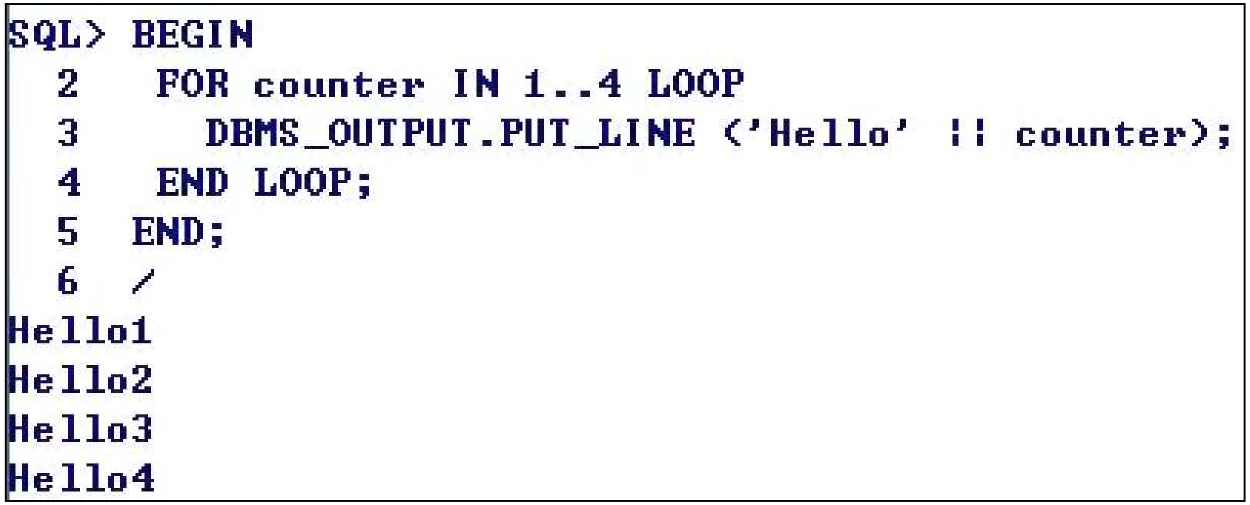
Syntax

```
FOR COUNTER IN [REVERSE] start...end LOOP
    statement1;
    statement2;
    ...
END LOOP;
```

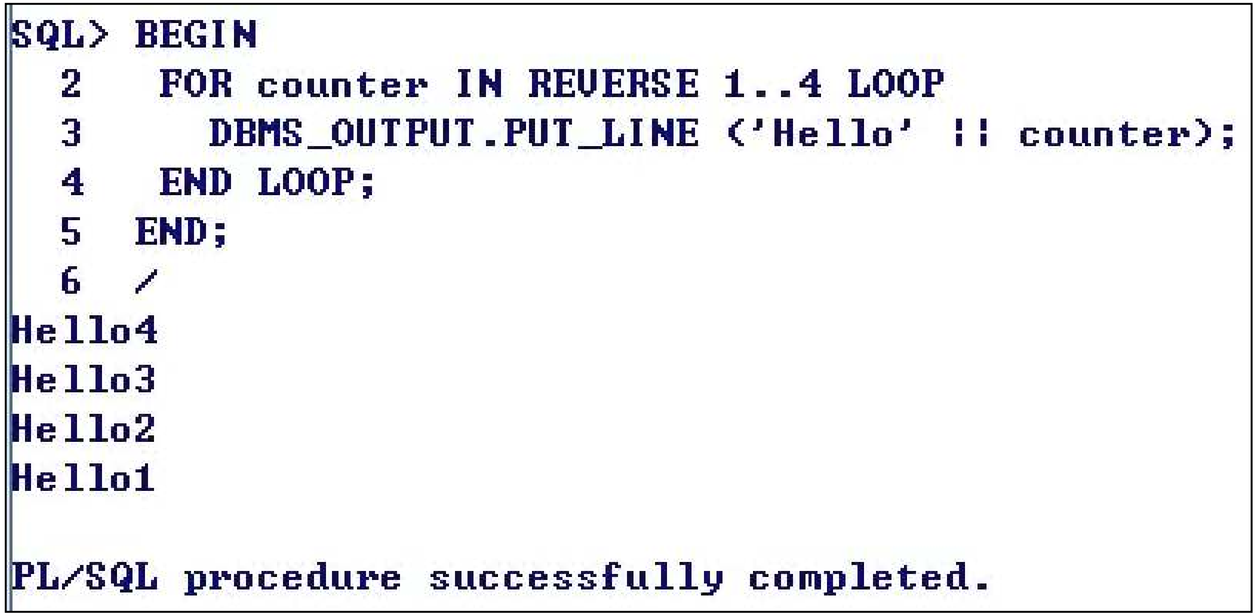
- `COUNTER`: implicitly incremented or decremented

- `REVERSE`: the looping order is reversed.

Control Flow: FOR Example



```
SQL> BEGIN
  2    FOR counter IN 1..4 LOOP
  3      DBMS_OUTPUT.PUT_LINE ('Hello' || counter);
  4    END LOOP;
  5  END;
  6  /
Hello1
Hello2
Hello3
Hello4
```



```
SQL> BEGIN
  2    FOR counter IN REVERSE 1..4 LOOP
  3      DBMS_OUTPUT.PUT_LINE ('Hello' || counter);
  4    END LOOP;
  5  END;
  6  /
Hello4
Hello3
Hello2
Hello1

PL/SQL procedure successfully completed.
```

Cursor

- Every time an SQL statement is executed, Oracle DBMS uses *specialized memory area* to store results from the statement that is interpreted and processed.
- Cursor is the one to refer to that area.

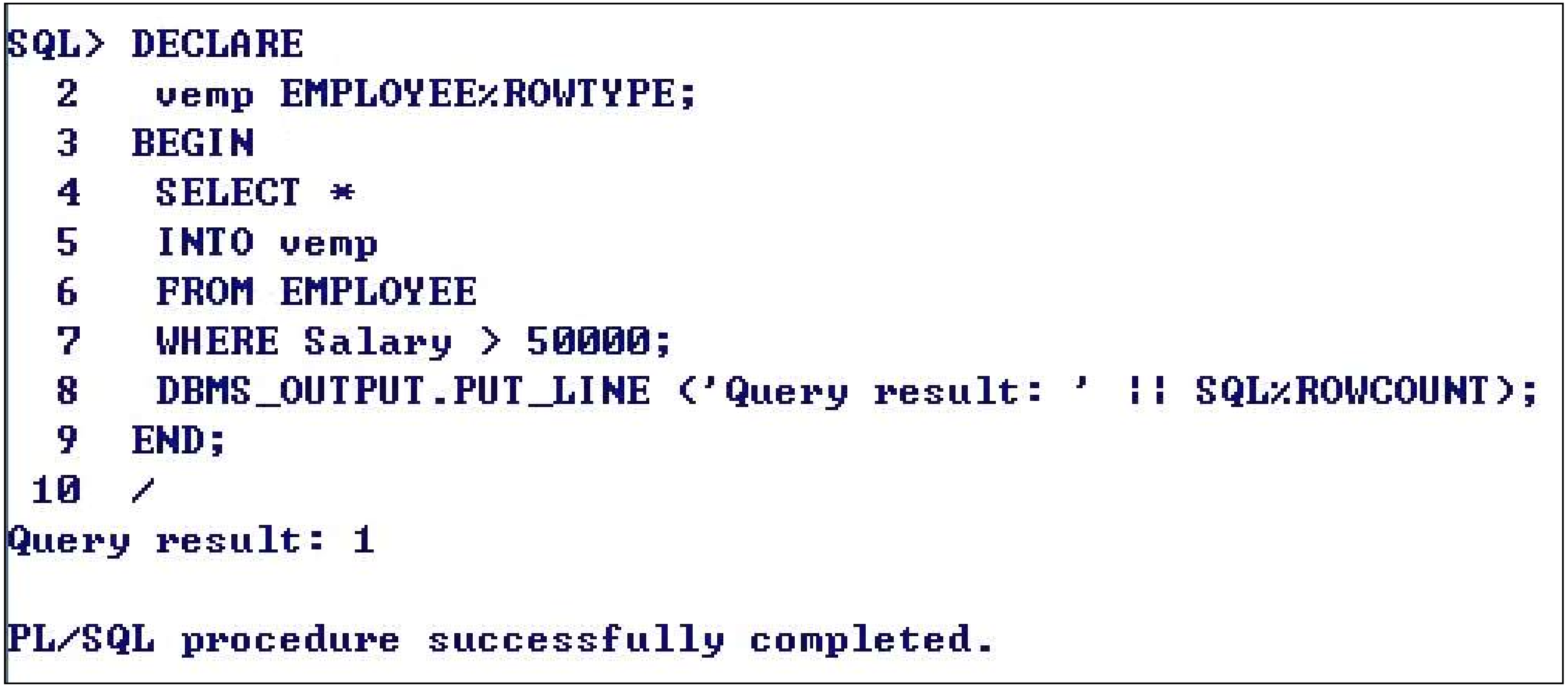
Cursor Type	Description
Implicit cursor	- Automatically declared by PL/SQL, for a `SELECT` statement returning one row or all DML statements - This type of cursor is used for your output that has been shown so far.
Explicit cursor	Declared by a user, for a `SELECT` statement returning multiple rows

Implicit Cursor

- Properties

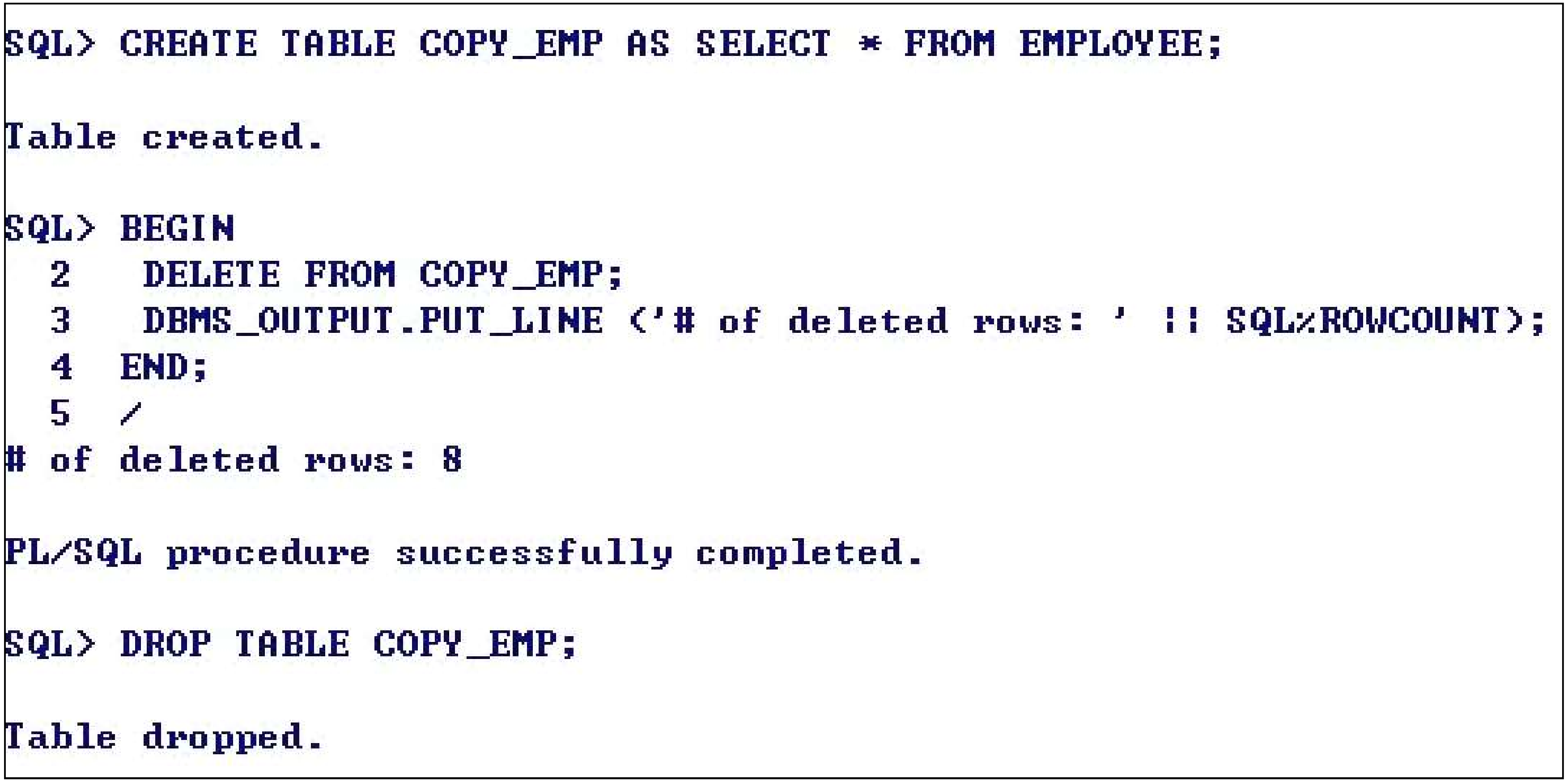
Property Name	Description
`SQL%ROWCOUNT`	Represents # of rows affected by the most recent SQL statement
`SQL%FOUND`	Returns `TRUE` if there exists a row(s) affected by the most recent SQL statement
`SQL%NOTFOUND`	Returns `TRUE` if there exist no row affected by the most recent SQL statement
`SQL%ISOPEN`	Always set to be `FALSE` for an implicit cursor, as it is closed after being executed.

Implicit Cursor Example: `SELECT`



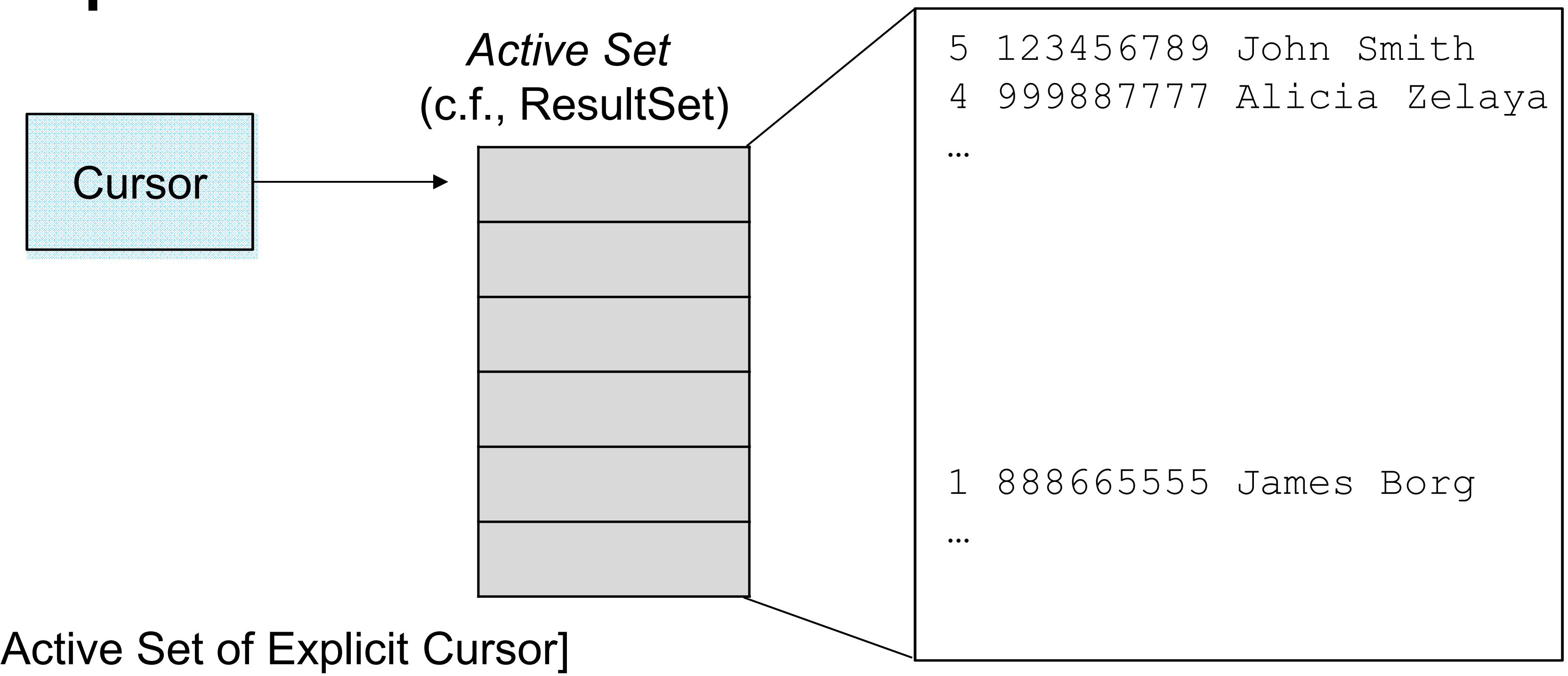
```
SQL> DECLARE
  2   vemp EMPLOYEE%ROWTYPE;
  3  BEGIN
  4   SELECT *
  5   INTO vemp
  6   FROM EMPLOYEE
  7   WHERE Salary > 50000;
  8   DBMS_OUTPUT.PUT_LINE ('Query result: ' || SQL%ROWCOUNT);
  9  END;
 10  /
Query result: 1

PL/SQL procedure successfully completed.
```

Implicit Cursor Example: DELETE



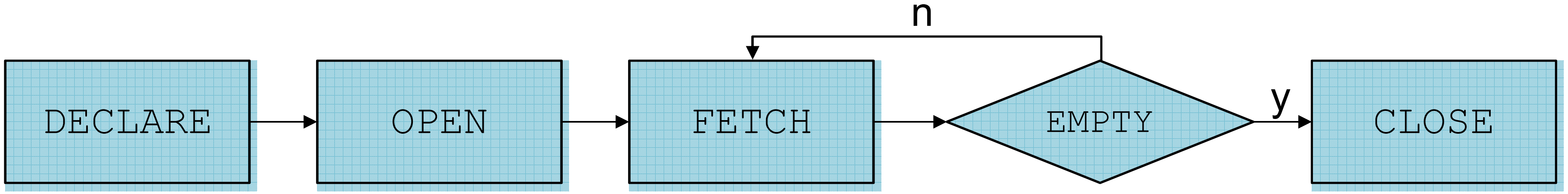
```
SQL> CREATE TABLE COPY_EMP AS SELECT * FROM EMPLOYEE;

Table created.

SQL> BEGIN
  2    DELETE FROM COPY_EMP;
  3    DBMS_OUTPUT.PUT_LINE ('# of deleted rows: ' || SQL%ROWCOUNT);
  4   END;
  5   /
# of deleted rows: 8

PL/SQL procedure successfully completed.

SQL> DROP TABLE COPY_EMP;

Table dropped.
```

Explicit Cursor



```
5 123456789 John Smith
4 999887777 Alicia Zelaya
…
1 888665555 James Borg
…
```

[Active Set of Explicit Cursor]



- DECLARE: - Named SQL area generation (or, cursor name)
- OPEN: - Identification of active set
- FETCH: - Loading a current row into a variable
- EMPTY: - Test the existing row
- Go to `FETCH` if a remaining row exists
- CLOSE: - Deactivation of active set

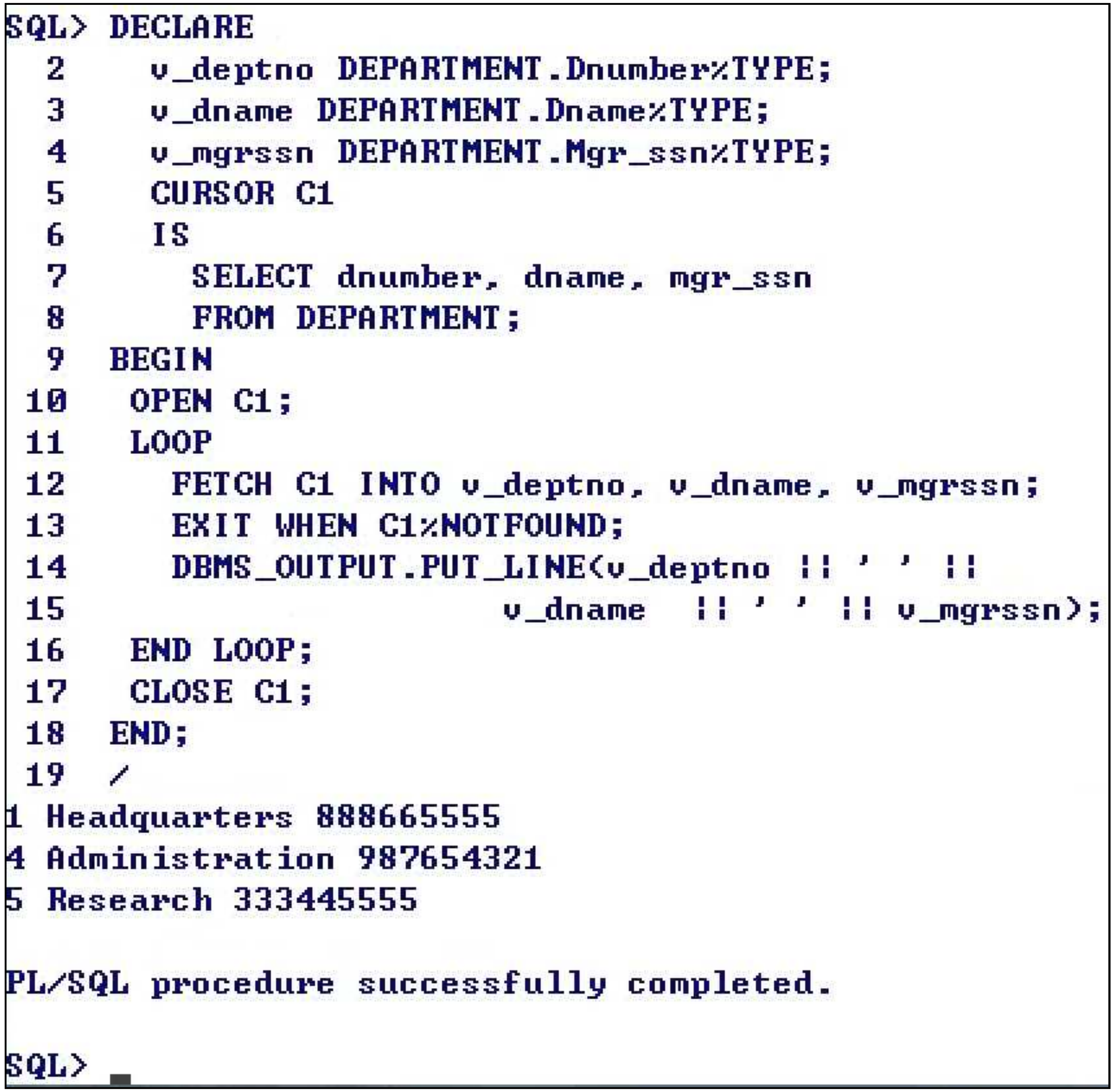
[Processing Steps of Explicit Cursor]

Explicit Cursor (Cont'd)

Syntax

```
DECLARE
   ...
   CURSOR cursor_name
   IS
   SELECT Statement;
   ... -- multiple cursors possible

OPEN cursor_name;

LOOP
   FETCH cursor_name INTO var1, var2, …;
   EXIT WHEN cursor_name%NOT_FOUND;
END LOOP;

CLOSE cursor_name;
```

Explicit Cursor: Example (Cont'd)



```
SQL> DECLARE
  v_deptno DEPARTMENT.Dnumber%TYPE;
  v_dname DEPARTMENT.Dname%TYPE;
  v_mgrssn DEPARTMENT.Mgr_ssn%TYPE;
  CURSOR C1
  IS
    SELECT dnumber, dname, mgr_ssn
    FROM DEPARTMENT;
BEGIN
 OPEN C1;
 LOOP
   FETCH C1 INTO v_deptno, v_dname, v_mgrssn;
   EXIT WHEN C1%NOTFOUND;
   DBMS_OUTPUT.PUT_LINE(v_deptno || ' ' ||
                   v_dname  || ' ' || v_mgrssn);
 END LOOP;
 CLOSE C1;
END;
/
1 Headquarters 888665555
4 Administration 987654321
5 Research 333445555

PL/SQL procedure successfully completed.

SQL>
```

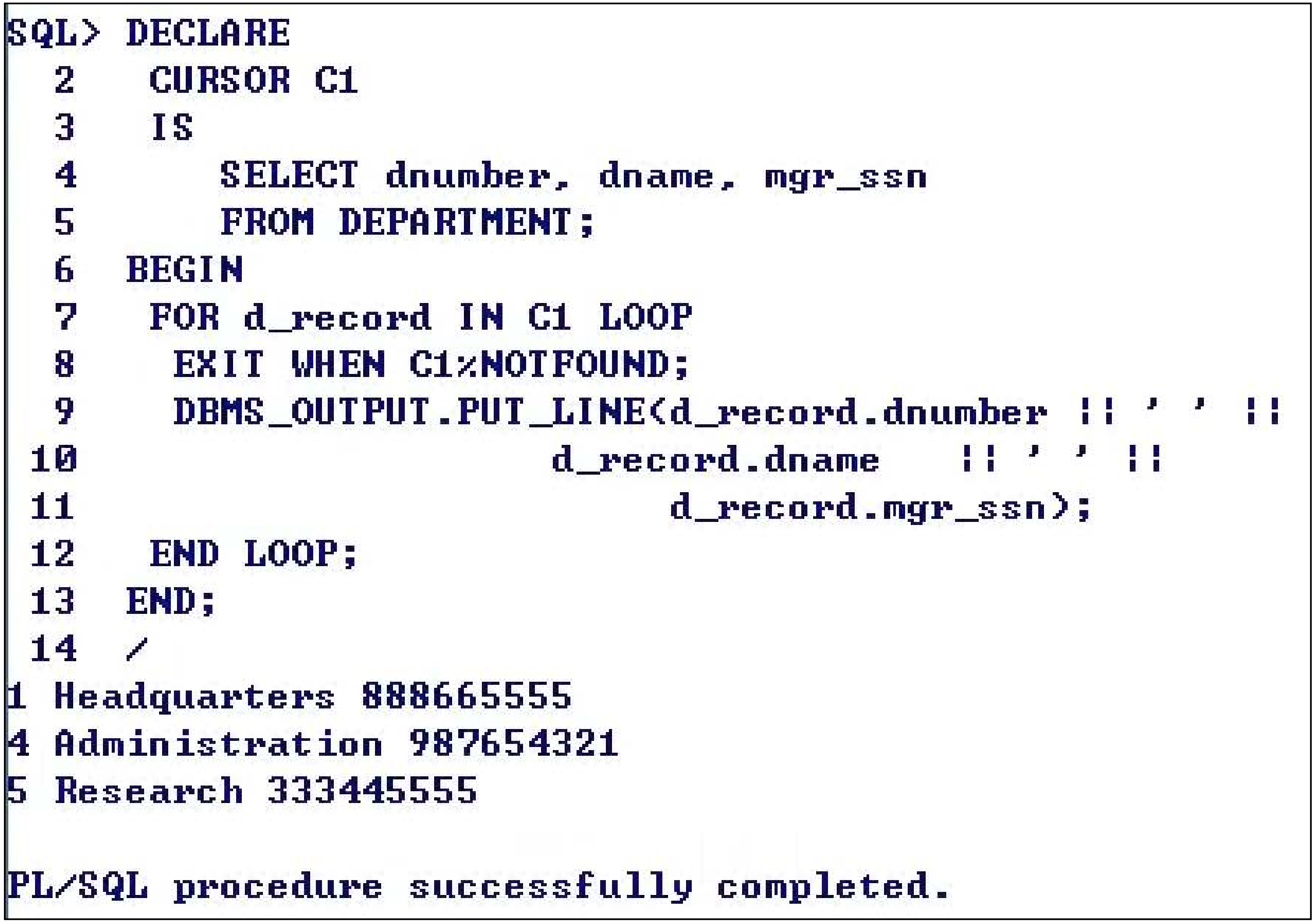
Cursor FOR LOOP Statement

Syntax

```
FOR record_name IN cursor_name LOOP
   Statement1;
   Statement2;
   . . .
END LOOP;
```

Much more convenient and simplified!

Cursor FOR LOOP Statement (Cont'd)



```
SQL> DECLARE
  2   CURSOR C1
  3   IS
  4      SELECT dnumber, dname, mgr_ssn
  5      FROM DEPARTMENT;
  6  BEGIN
  7   FOR d_record IN C1 LOOP
  8    EXIT WHEN C1%NOTFOUND;
  9    DBMS_OUTPUT.PUT_LINE(d_record.dnumber || ' ' ||
 10                        d_record.dname    || ' ' ||
 11                             d_record.mgr_ssn);
 12   END LOOP;
 13  END;
 14  /
1 Headquarters 888665555
4 Administration 987654321
5 Research 333445555

PL/SQL procedure successfully completed.
```

Exception Handling

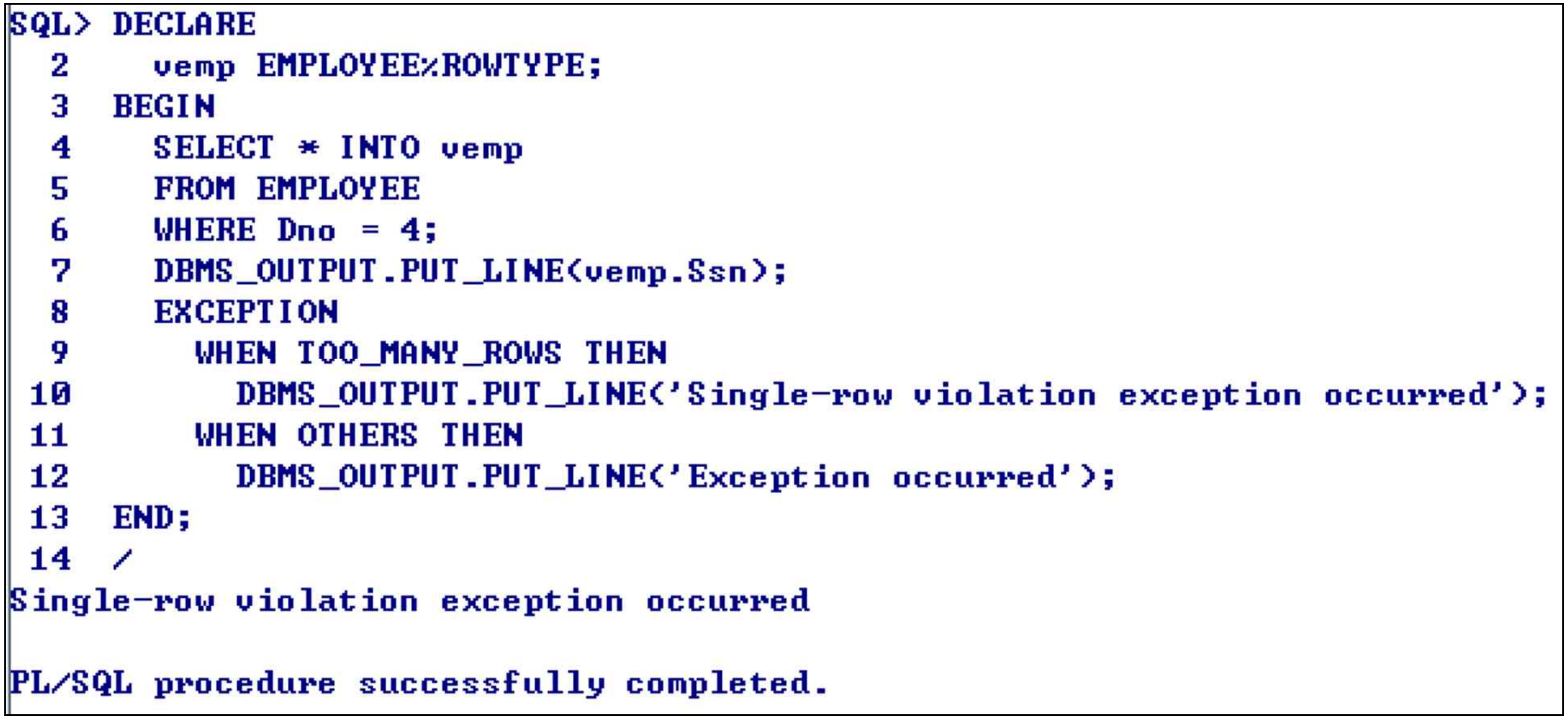
Syntax

```
EXCEPTION
  WHEN exception_name [OR exception_name] THEN
    Statement1;
    Statement2;
  WHEN exception_name [OR exception_name] THEN
    Statement1;
    Statement2;
  . . .
  [WHEN OTHERS THEN
    Statement1;
    Statement2;]
```

Exception Handling (Cont'd)

Cursor Property Name	Error Number	Description
NO_DATA_FOUND	ORA-01403	Exception occurring when no data returned by SELECT
TOO_MANY_ROWS	ORA-01422	Exception occurring when more than two rows data returned by SELECT
INVALID_CURSOR	ORA-01001	Exception occurring when wrong cursor is used
ZERO_DIVIDE	ORA-01476	Exception occurring when division by 0
VALUE_ERROR	ORA-06502	Exception occurring when wrong arithmetic operation or conversion or size constraint
DUP_VAL_ON_INDEX	ORA-00001	Exception occurring when duplicate values are set to unique attributes
ACCESS_INTO_NULL	ORA-06530	Exception occurring when a value is set to an attribute of an uninitialized object

Exception Handling Examples



```
SQL> DECLARE
  2    vemp EMPLOYEE%ROWTYPE;
  3  BEGIN
  4    SELECT * INTO vemp
  5    FROM EMPLOYEE
  6    WHERE Dno = 4;
  7    DBMS_OUTPUT.PUT_LINE(vemp.Ssn);
  8    EXCEPTION
  9      WHEN TOO_MANY_ROWS THEN
 10        DBMS_OUTPUT.PUT_LINE('Single-row violation exception occurred');
 11      WHEN OTHERS THEN
 12        DBMS_OUTPUT.PUT_LINE('Exception occurred');
 13  END;
 14  /
Single-row violation exception occurred

PL/SQL procedure successfully completed.
```

Stored Procedure (저장 프로시저)

- Implements program logic (in PL/SQL).
- Exists as an object and is used in a DBMS (like Oracle).
- Similar to a function in GPL; indicates an independent program with a specified task order in execution
  - Includes `PROCEDURE`, `FUNCTION`, and `TRIGGER`.
- Once defined, it is stored in the DBMS.
  - That's why it's also called *stored procedure* or persistent stored module
  - No need to recompile; possible to invoke multiple times
- May return result (via `RETURN)` or not, as opposed to function in the DBMS (in Oracle).

CREATE PROCEDURE

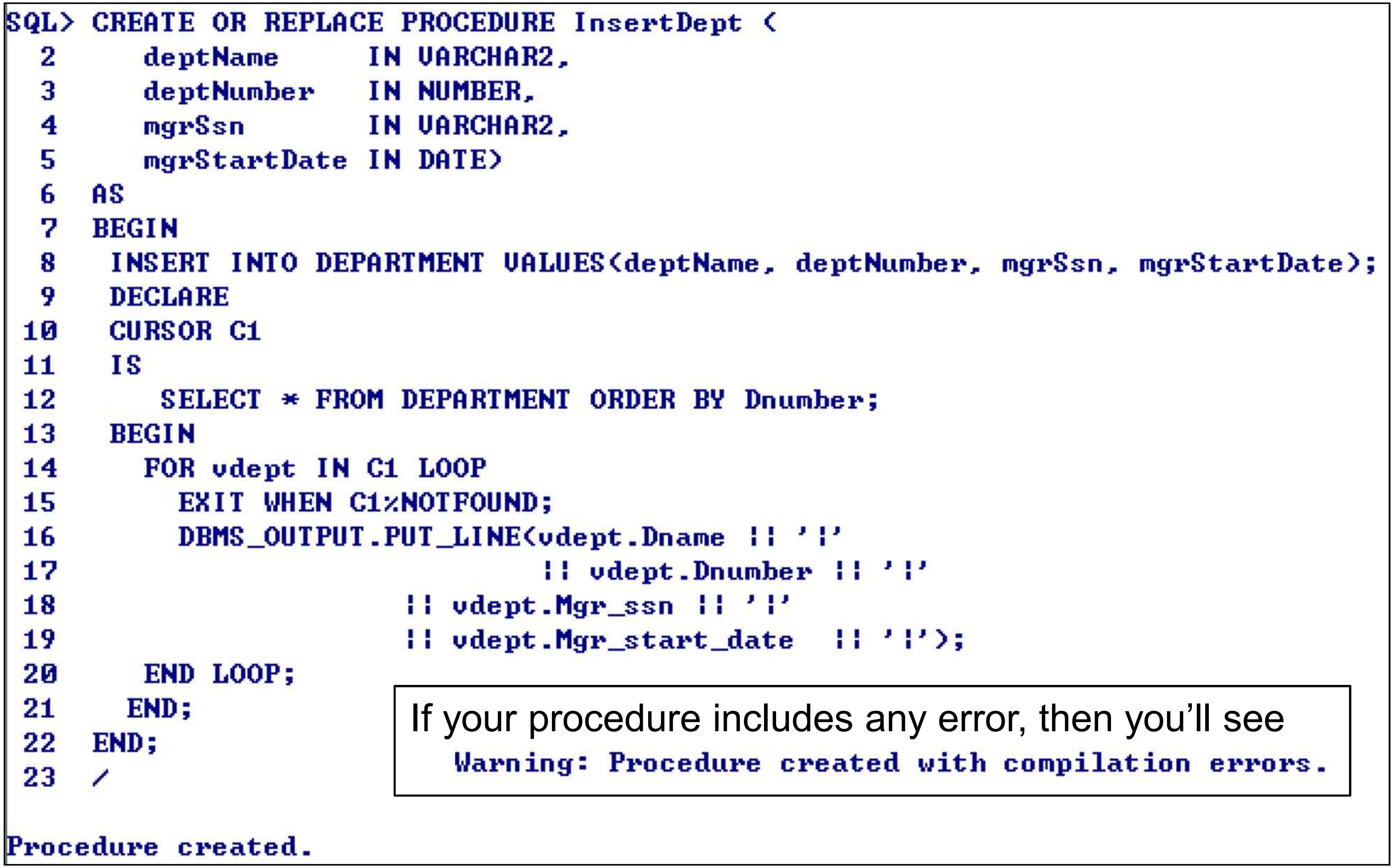
- Defines a procedure.
- How to define a procedure?
  - A procedure in PL/SQL consists of `BEGIN`-`END` .
    - In `BEGIN`, we declare variables and parameters.
    - In `END`, a specific program logic (procedural actions) is implemented.
  - A certain parameter is a value to be passed in the (stored) procedure when the procedure is invoked.
  - A variable is used within the stored procedure or trigger.
  - Comments can be specified between `/*` and `*/`.
    - Like in SQL, a single-line comment can be specified after '`--`'.

CREATE PROCEDURE (Cont’d)

Syntax

```
CREATE [OR REPLACE] PROCEDURE procedure_name
  [(param_name mode [IN | OUT | IN OUT] data_type,
    param_name mode [IN | OUT | IN OUT] data_type, ...)]
IS
 [Variable;]
AS
BEGIN
 ...
END;
```

1) Procedure for Insert

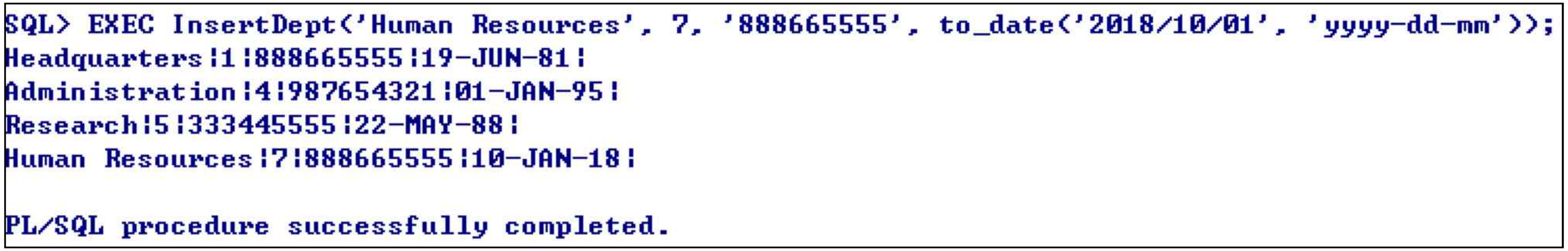


```
SQL> CREATE OR REPLACE PROCEDURE InsertDept (
  2      deptName      IN VARCHAR2,
  3      deptNumber    IN NUMBER,
  4      mgrSsn        IN VARCHAR2,
  5      mgrStartDate IN DATE)
  6  AS
  7  BEGIN
  8   INSERT INTO DEPARTMENT VALUES(deptName, deptNumber, mgrSsn, mgrStartDate);
  9   DECLARE
 10   CURSOR C1
 11   IS
 12      SELECT * FROM DEPARTMENT ORDER BY Dnumber;
 13   BEGIN
 14     FOR vdept IN C1 LOOP
 15       EXIT WHEN C1%NOTFOUND;
 16       DBMS_OUTPUT.PUT_LINE(vdept.Dname || '|'
 17                            || vdept.Dnumber || '|'
 18                    || vdept.Mgr_ssn || '|'
 19                    || vdept.Mgr_start_date  || '|');
 20     END LOOP;
 21    END;
 22  END;
 23  /

Procedure created.
```

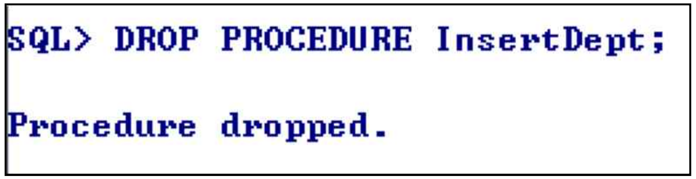
If your procedure includes any error, then you'll see

```
Warning: Procedure created with compilation errors.
```

1) Procedure for Insert – Execution & Drop

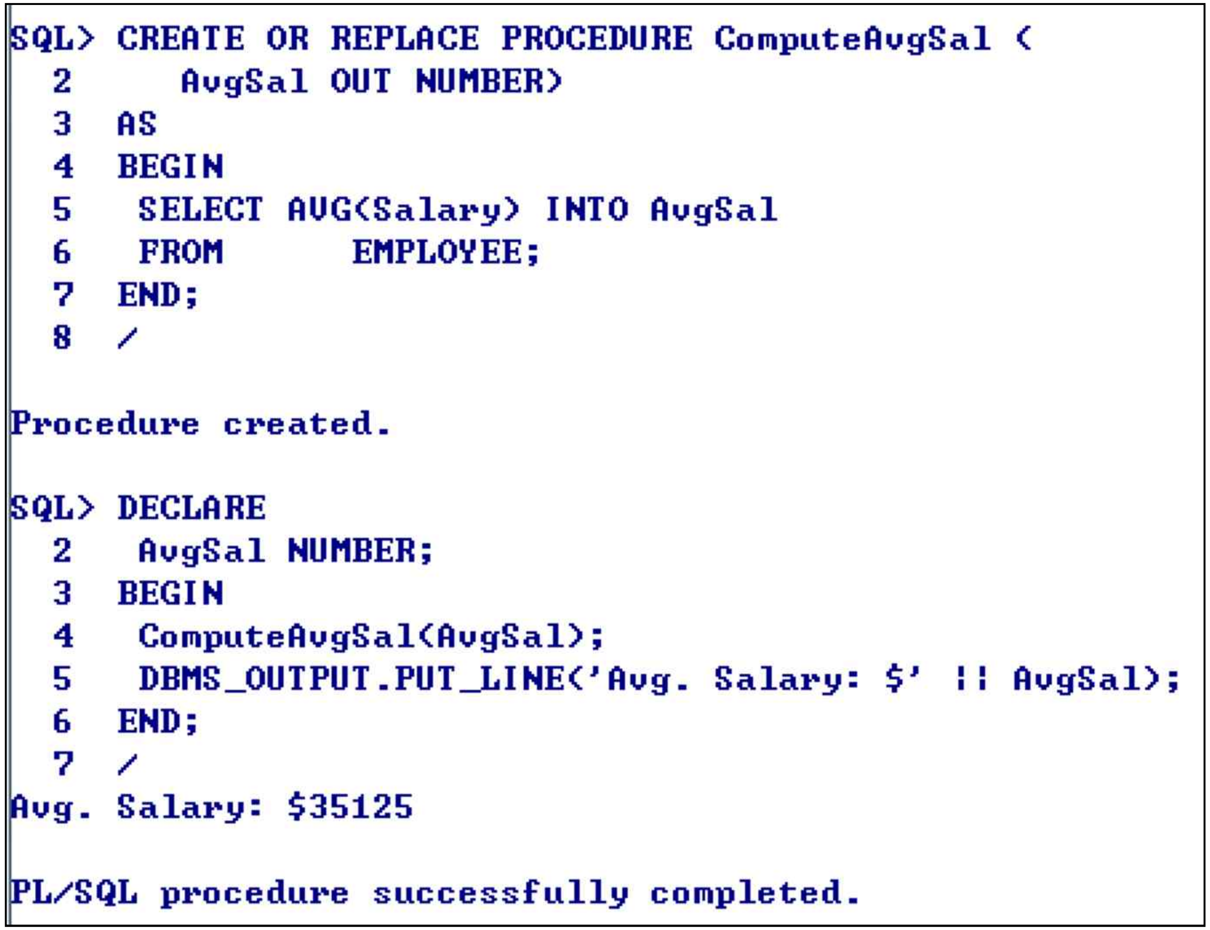


```
SQL> EXEC InsertDept('Human Resources', 7, '888665555', to_date('2018/10/01', 'yyyy-dd-mm'));
Headquarters|1|888665555|19-JUN-81|
Administration|4|987654321|01-JAN-95|
Research|5|333445555|22-MAY-88|
Human Resources|7|888665555|10-JAN-18|

PL/SQL procedure successfully completed.
```



```
SQL> DROP PROCEDURE InsertDept;

Procedure dropped.
```

2) Procedure for Returning a Scalar Value



```
SQL> CREATE OR REPLACE PROCEDURE ComputeAvgSal (
           AvgSal OUT NUMBER)
     AS
     BEGIN
      SELECT AVG(Salary) INTO AvgSal
      FROM       EMPLOYEE;
     END;
     /

Procedure created.

SQL> DECLARE
      AvgSal NUMBER;
     BEGIN
      ComputeAvgSal(AvgSal);
      DBMS_OUTPUT.PUT_LINE('Avg. Salary: $' || AvgSal);
     END;
     /
Avg. Salary: $35125

PL/SQL procedure successfully completed.
```

CREATE FUNCTION

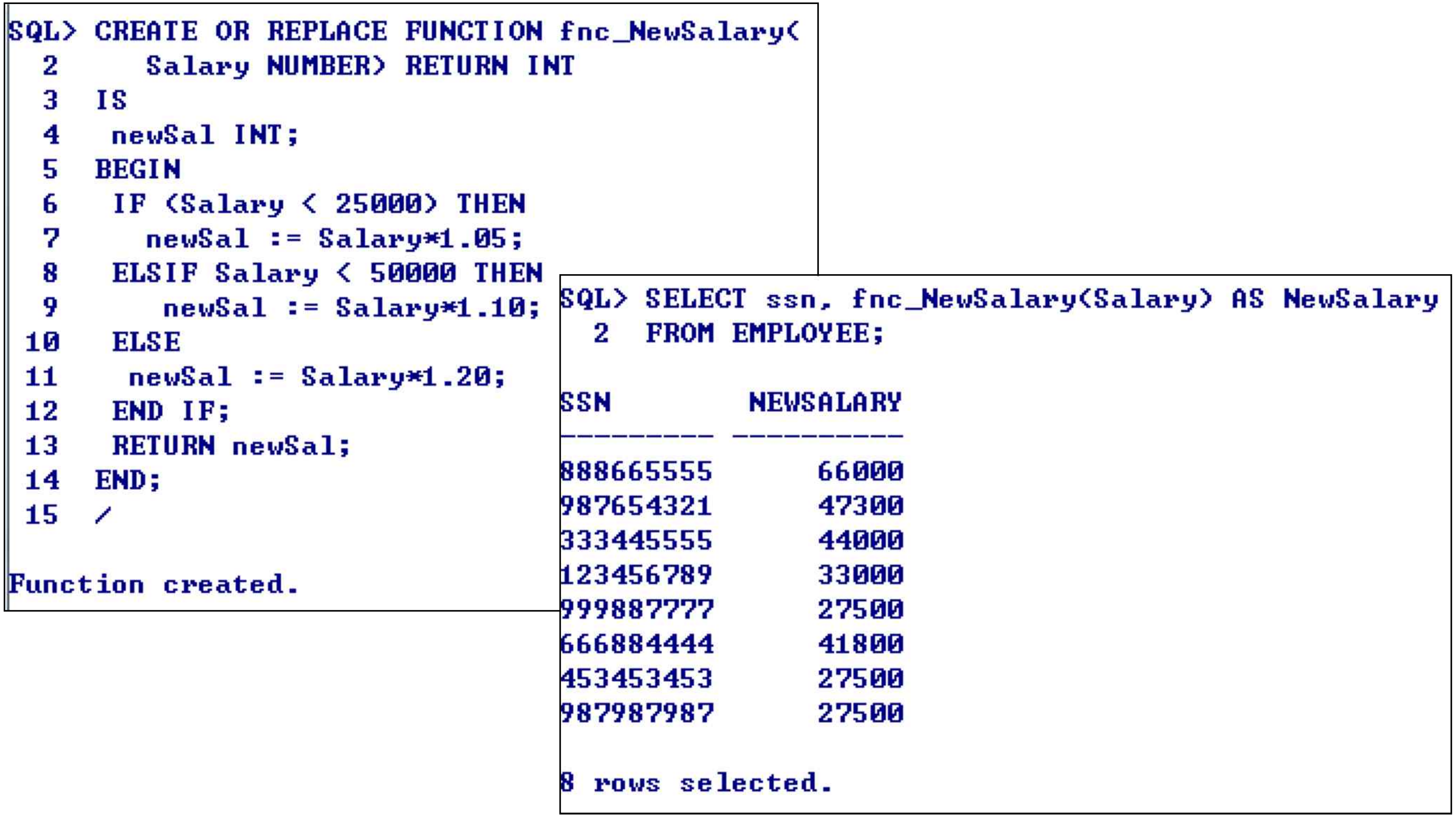
- Defines a user-defined function.
- Compute a value in the body and returns it as in a math function.
- Invoked typically in an SQL statement or another procedure

CREATE FUNCTION (Cont'd)

Syntax

```
CREATE [OR REPLACE] FUNCTION function_name
 [(param_name data_type,
  param_name data_type, …)]
RETURN data_type
IS
 [Variable;  ...]
BEGIN
 …
 RETURN return_value
END;
```

Function for Returning a Scalar Value



```
SQL> CREATE OR REPLACE FUNCTION fnc_NewSalary(
  2      Salary NUMBER) RETURN INT
  3  IS
  4   newSal INT;
  5  BEGIN
  6   IF (Salary < 25000) THEN
  7     newSal := Salary*1.05;
  8   ELSIF Salary < 50000 THEN
  9      newSal := Salary*1.10;
 10   ELSE
 11    newSal := Salary*1.20;
 12   END IF;
 13   RETURN newSal;
 14  END;
 15  /

Function created.
```

```
SQL> SELECT ssn, fnc_NewSalary(Salary) AS NewSalary
  2  FROM EMPLOYEE;

SSN         NEWSALARY
--------- ----------
888665555      66000
987654321      47300
333445555      44000
123456789      33000
999887777      27500
666884444      41800
453453453      27500
987987987      27500

8 rows selected.
```

Comparison of Procedure, Trigger, and Function (in Oracle)

	Procedure	Trigger	(User-Defined) Function
Similarities	All are classified as **stored procedures** and implemented by PL/SQL in Oracle.		
How to Define?	`CREATE PROCEDURE`	`CREATE TRIGGER`	`CREATE FUNCTION`
How to Execute?	Invoked via `EXEC`	Automatically executed when `INSERT`, `DELETE`, `UPDATE` statements are executed	Invoked via `SELECT`
Differences	Performs a complex logic that cannot be specified by SQL.	Sets a default value, Keeps schema constraints, Modifies a view, Detects a referential integrity constraint violation	Returns a computed value; Can be used in an SQL statement

Lab #7: PSM Practice in PL/SQL

- Deadline: **Friday midnight** (10/19/2018)

- Task 1: Write a (stored) Procedure, named `ComputeMaxAvgHours`, in PL/SQL to perform the following:
  - Receive a **department number** as an **`IN`** parameter.
  - For each project controlled by the given department,

    compute and **print out** the averaged (working) hours, rounded up to two decimal points (소수점 둘째 자리에서 반올림), of employees working on that project.
  - Find the maximum (average) hours spent on the projects in the given department and put the **maximum hour number** into an **`OUT`** parameter.

- Task 2: For testing, write a simple PL/SQL block to invoke your procedure and to **print out** the maximum hour number.

Lab #7: PSM Practice in PL/SQL (Cont'd)

[Requirements]

(1) Print what department number is received.

(2) Use aggregate functions for calculating the averaged hours, rounding up the results, and finding the max hours.

- The grouping attributes: `DEPARTMENT.Dnumber,PROJECT.Pnumber`.

(3) Sort the result by `PROJECT.Pnumber` in ascending order and include the received department number in the result.

(4) Use the Cursor `FOR LOOP` statement in Task 1.

(5) For the output, provide a proper column header and use `DBMS_OUTPUT.PUT_LINE()`. (For your reference, see the next slide.)

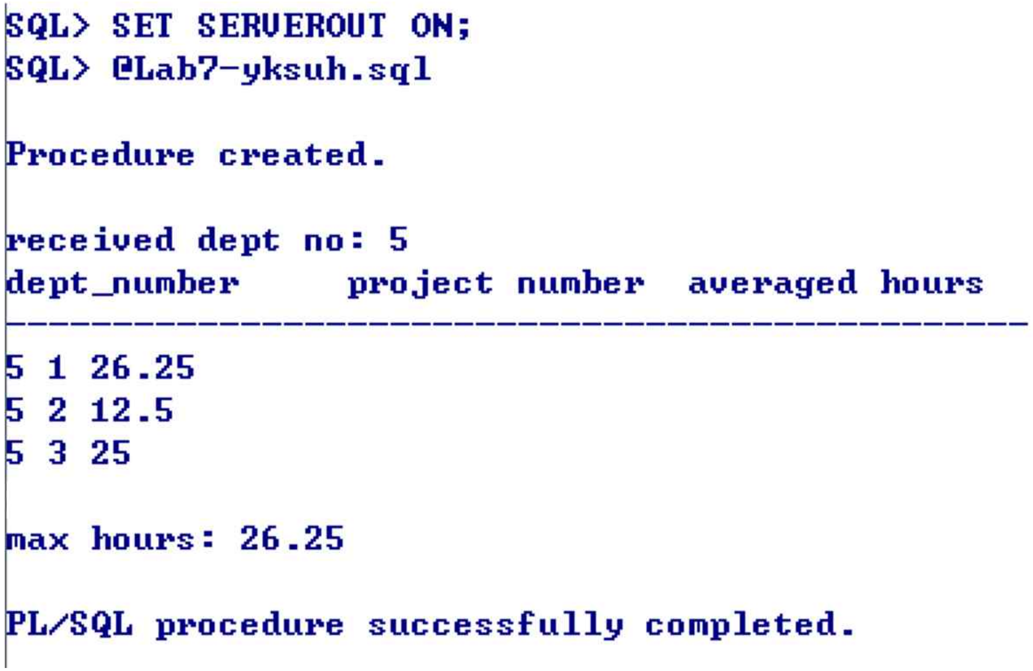
(6) In Task 1: Department number => `IN` mode, Maximum hour => `OUT` mode

[Submission]

- Name your file as '`lab7-`Your_Student_ID`.sql`'.
- Include in the `sql` file all your code for Tasks 1 and 2 in sequence.
- Upload the `sql` file into LMS.

Lab #7: Practice of a Stored Procedure in PL/SQL (Cont'd)

- Expected output
  - If you pass "dept no. = 5" into your procedure, then the output would be like:



```
SQL> SET SERVEROUT ON;
SQL> @Lab7-yksuh.sql

Procedure created.

received dept no: 5
dept_number     project number  averaged hours
------------------------------------------------
5 1 26.25
5 2 12.5
5 3 25

max hours: 26.25

PL/SQL procedure successfully completed.
```