

LAB BOOK 4

COMP41090 SQL PROGRAMMING

ANDREW DOYLE

STUDENT NUMBER: 12252388

MSc COMPUTER SCIENCE (CONVERSION)

16th April 2013

TABLE OF CONTENTS

TABLE OF CONTENTS	1
LIST OF FIGURES	1
INTRODUCTION	2
PREQUEL	3
QUESTION 1	5
QUESTION 2	6
QUESTION 3	8
QUESTION 4	10
QUESTION 5	12
QUESTION 6	13
Simple Method.....	13
Flexible Function Method	14
QUESTION 7	15

LIST OF FIGURES

Figure P.1	3
Figure P.2	3
Figure P.3	3
Figure P.4	3
Figure P.5	4
Figure P.6	4
Figure P.7	4
Figure 1.1 – Sample Procedure	5
Figure 1.2 – Sample Function.....	5
Figure 2.1 – Procedure Implementation.....	7
Figure 3.1 – Trim and VSIZE	9
Figure 4.1 – Output of character replacement procedure	11
Figure 5.1 – Number x 2.....	12
Figure 6.1 – Simple Function Method.....	13
Figure 6.2 – Flexible Function Method	14
Figure 7.1 – Creating Package	17
Figure 7.2 – Output functions from a package	18

INTRODUCTION

This introduction serves sets out the format of the author's submission for this Lab Book. For each question, the SQL code is provided in this document accompanied by explanatory notes which serve to demonstrate and reinforce knowledge of the topics covered. The notes may prove useful to the author as a reference point for future Lab Book or project submissions.

In each section screenshots are also provided which provide evidence of the SQL code being implemented in the database together with the actual output to the screen. Additionally, separate text files are included with the submission (one for each question), which the Examiner may wish to utilize to test the code. The exercises specified before question 1 are also included in the prequel section as part of the documentation process.

PREQUEL

This prequel section demonstrates introductory functions created to demonstrate capabilities to be used later in the lab.

Figure P.1

```
SELECT 'Firstname Surname' AS prequel_1_spaces FROM DUAL;
```

```
SQL> SELECT 'Firstname Surname' AS prequel_1_spaces FROM DUAL;
PREQUEL_1_SPACES
-----
Firstname Surname
```

Figure P.2

```
SELECT TRIM('Firstname Surname') AS prequel_2 FROM DUAL;
```

```
SQL> SELECT TRIM('Firstname Surname') AS prequel_2 FROM DUAL;
PREQUEL_2
-----
Firstname Surname
```

[Click here for definition on TRIM from Oracle Documentation](#)

Figure P.3

```
SELECT UPPER('Firstname Surname') AS prequel3_upper FROM DUAL;
```

```
SQL> SELECT UPPER('Firstname Surname') AS prequel3_upper FROM DUAL;
PREQUEL3_UPPER
-----
FIRSTNAME SURNAME
```

[Click here for definition on UPPER from Oracle Documentation](#)

Figure P.4

```
SELECT LOWER('FIRSTNAME Surname') AS prequel4_lower FROM DUAL;
```

```
SQL> SELECT LOWER('FIRSTNAME Surname') AS prequel4_lower FROM DUAL;
PREQUEL4_LOWER
-----
firstname surname
```

[Click here for definition on LOWER from Oracle Documentation](#)

Figure P.5

```
SELECT INITCAP('FIRSTNAME SURNAME') AS prequel_5_capital_1 FROM DUAL;
```

```
SQL> SELECT INITCAP('FIRSTNAME SURNAME') AS prequel_5_capital_1 FROM DUAL;
PREQUEL_5_CAPITAL
-----
Firstname Surname
```

[Click here for definition on INITCAP from Oracle Documentation](#)

Figure P.6

```
SELECT VSIZE('Firstname Surname') FROM DUAL;
```

```
SQL> SELECT VSIZE('Firstname Surname') FROM DUAL;
VSIZE('FIRSTNAMESURNAME')
-----
17
```

[Click here for definition on VSIZE from Oracle Documentation](#)

Figure P.7

```
SELECT REPLACE('This is an example', 'his', 'hat') AS prequel_6_replace FROM DUAL;
```

```
SQL> SELECT REPLACE('This is an example', 'his', 'hat') AS prequel_6_replace FROM DUAL;
PREQUEL_6_REPLACE
-----
That is an example
```

[Click here for definition on REPLACE from Oracle Documentation](#)

QUESTION 1

Sample Procedure

```

1| CREATE OR REPLACE PROCEDURE HelloWorldProc
2| IS
3| BEGIN
4|     DBMS_OUTPUT.PUT_LINE('Hello world');
5| END HelloWorldProc;
6| /
7|
8| BEGIN
9| HelloWorldProc();
10| END;
11| /

```

Figure 1.1 – Sample Procedure

```

SQL> connect ad Doyle;
Enter password:
Connected.
SQL> CREATE OR REPLACE PROCEDURE HelloWorldProc
2  IS
3  BEGIN
4  DBMS_OUTPUT.PUT_LINE('Hello world');
5  END HelloWorldProc;
6  /

Procedure created.

SQL> BEGIN
2  HelloWorldProc();
3  END;
4  /

PL/SQL procedure successfully completed.

```

Sample Function

```

12| CREATE OR REPLACE FUNCTION HelloWorldFunc
13| RETURN NUMBER
14| IS
15| BEGIN
16|     DBMS_OUTPUT.PUT_LINE('Hello world');
17| RETURN 1;
18| END HelloWorldFunc;
/

SELECT HelloWorldFunc() FROM DUAL;

```

Figure 1.2 – Sample Function

```

SQL> CREATE OR REPLACE FUNCTION HelloWorldFunc
2  RETURN NUMBER
3  IS
4  BEGIN
5  DBMS_OUTPUT.PUT_LINE('Hello world');
6  RETURN 1;
7  END HelloWorldFunc;
8  /

Function created.

SQL>
SQL> SELECT HelloWorldFunc() FROM DUAL;

HELLOWORLDFUNC()
-----
1

```

Creating a procedure allows us to store PL/SQL inside the database. When the procedure exists in the database you can call it at any time. The procedure is declared on line 1 (the **REPLACE** keyword means the procedure is overwritten if it already exists) and is named **HelloWorldProc**.

The **IS** keyword (Line 2) denotes the declaration section of the procedure. Declarations can be made which are local to the procedure; this section will be used in Question 3. The **BEGIN** keyword (Line 8) denotes the **executable section**, also known as the **body** of the procedure; where the main purpose of the procedure is specified. Exceptions may also be optionally specified in the body.

The **END** keyword denotes the end of the procedure and the **/** character gives Oracle permission to complete the procedure (Lines 10 – 11).

The declaration of a function is the same as for a procedure. In the sample function above (Line 13) it specifies that a number is returned (this will be demonstrate in use for Question 5).

In the body of the function we use **DBMS_OUTPUT.PUT_LINE** to output the contents of the buffer ("Hello World") to the console, this is a useful debugging tool – information is not actually saved to the database. The return value is specified (Line 17) before the function is ended in the same way the procedure was ended.

Some important notes regarding functions and procedures include:

- Procedures **do not** return a value to the caller whereas functions **do**
- Functions can be called from SQL queries, procedures cannot
- Functions are considered expressions whereas procedures are not
- Procedures can modify parameters that are passed to it whereas functions cannot
- Therefore do not use functions for a block that will modify the database
- Functions can return a value whereas procedures cannot
- Procedures can be called from PL/SQL whereas functions can be called from SQL queries or PL/SQL

QUESTION 2

```
1| CREATE OR REPLACE PROCEDURE question_2_lab_4 (firstname IN NVARCHAR2, surname IN NVARCHAR2)
2| IS
3| BEGIN
4|     DBMS_OUTPUT.PUT_LINE (LOWER (firstname) || LOWER (surname));
5|     DBMS_OUTPUT.PUT_LINE (UPPER (firstname) || UPPER (surname));
6|     DBMS_OUTPUT.PUT_LINE (INITCAP (firstname) || INITCAP (surname));
7|
8| END question_2_lab_4;
9| /
10| set serveroutput on;
11| BEGIN
12| question_2_lab_4 ('THOMAS', 'Doyle');
13|     END;
14| /
```

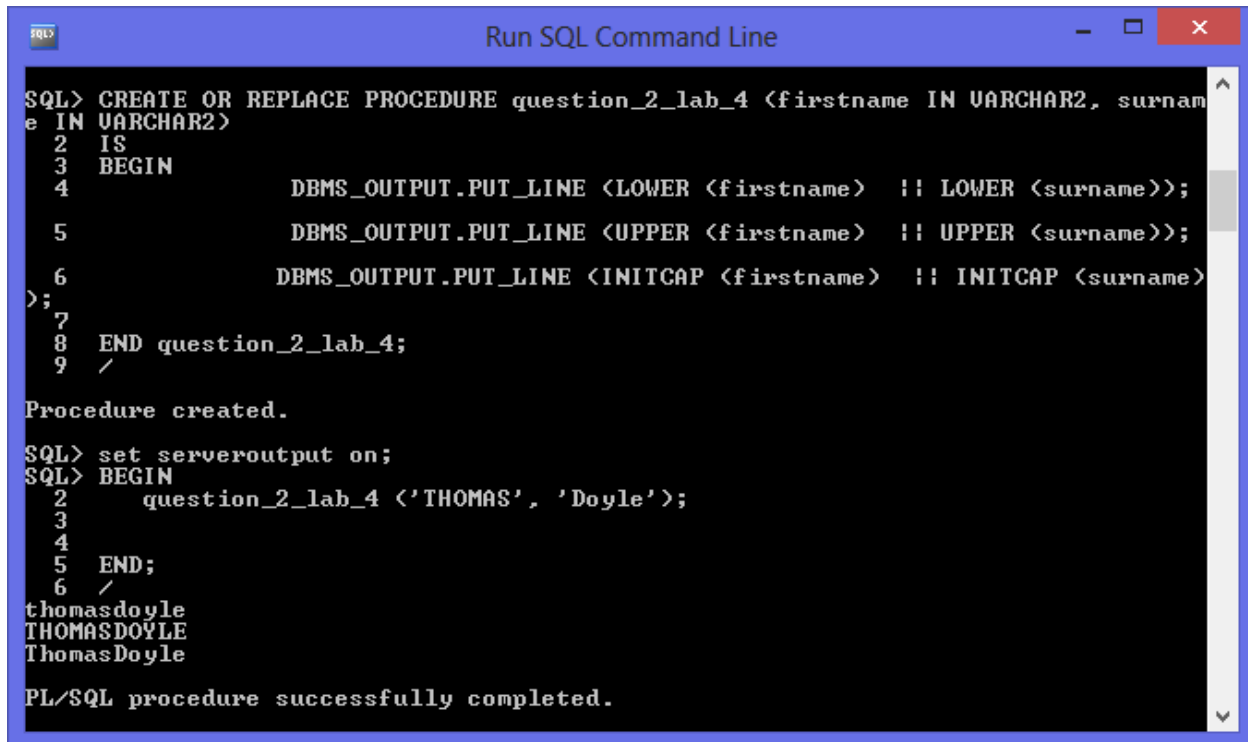
This question demonstrates the use, within a procedure, of **DBMS_OUTPUT.PUT_LINE** in conjunction with the **LOWER**, **UPPER**, and **INITCAP** functions demonstrated in the prequel exercises. As shown on Line 1, the procedure is created with two input parameters, *firstname* and *surname*.

In the body, on Line 4, **DBMS_OUTPUT.PUT_LINE** is used to output the contents of the buffer. The buffer contains the **LOWER** function applied to the variable *firstname* and *surname*. The variables are joined in the buffer using the double vertical bar symbol **||**. **UPPER** and **INITCAP** are implemented in the same manner.

Note, on line 10, the use of **set server output on;** this must be specified for output to be displayed on the screen. Once in a session you do not need to specify this line again. The function is called by using the **BEGIN** keyword followed by the function name(parameters).

On Line 12, 'THOMAS' is passed into the *firstname* variable and 'Doyle' into the *surname* variable of the function. The resulting output is showed in Figure 2.1 with the desired formatting consequences.

Figure 2.1 – Procedure Implementation



```
SQL> CREATE OR REPLACE PROCEDURE question_2_lab_4 (firstname IN VARCHAR2, surname IN VARCHAR2)
2  IS
3  BEGIN
4      DBMS_OUTPUT.PUT_LINE <LOWER <firstname> || LOWER <surname>>;
5      DBMS_OUTPUT.PUT_LINE <UPPER <firstname> || UPPER <surname>>;
6      DBMS_OUTPUT.PUT_LINE <INITCAP <firstname> || INITCAP <surname>>;
7  END;
8  END question_2_lab_4;
9  /

Procedure created.

SQL> set serveroutput on;
SQL> BEGIN
2     question_2_lab_4 ('THOMAS', 'Doyle');
3
4
5  END;
6  /

thomasdoyle
THOMASDOYLE
ThomasDoyle

PL/SQL procedure successfully completed.
```

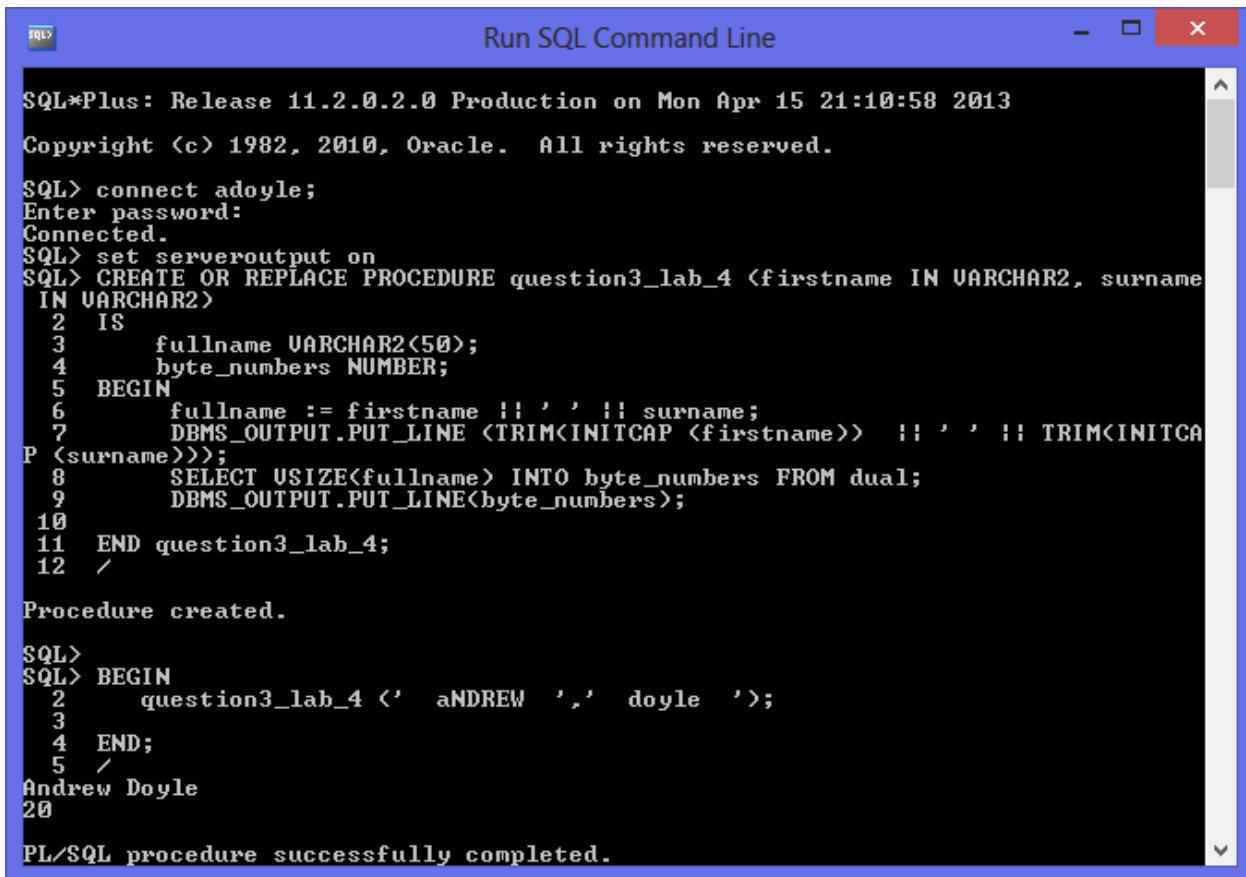

QUESTION 3

```
1| CREATE OR REPLACE PROCEDURE question3_lab_4 (firstname IN VARCHAR2, surname IN VARCHAR2)
2| IS
3|   fullname VARCHAR2(50);
4|   byte_numbers NUMBER;
5| BEGIN
6|   fullname := firstname || ' ' || surname;
7|   DBMS_OUTPUT.PUT_LINE (TRIM(INITCAP (firstname)) || ' ' || TRIM(INITCAP (surname)));
8|   SELECT VSIZE(fullname) INTO byte_numbers FROM dual;
9|   DBMS_OUTPUT.PUT_LINE(byte_numbers);
10|
11| END question3_lab_4;
12| /
13|
14| BEGIN
15|   question3_lab_4 (' aNDREW ', ' doyle ');
16|
17| END;
18| /
```

This question demonstrates the combined use of the **TRIM** and **INITCAP** functions as part of the **DBMS_OUTPUT.PUT_LINE** buffer. Within the buffer the **TRIM** function is the outer command, and within that, the variables *firstname* and *surname* are joined and altered using the **INITCAP** function.

Before this, in the declaration section (**IS** – line 2-3), the variable *fullname* is declared along with *byte_numbers*. In the body, *fullname* is assigned a join of the variables *firstname*, *surname* and a blank space to separate the names. This allows us to calculate the number of bytes in *fullname* using **VSIZE** in a **SELECT** statement. The **VSIZE** of *fullname* is selected into *byte_numbers* as shown on Line 7.

This allows us to **DBMS_OUTPUT.PUT_LINE** *byte_numbers* which represents the number of bytes in the internal representation of the person's name. See Figure 3.1 for output.

Figure 3.1 – Trim and VSIZE

```
SQL*Plus: Release 11.2.0.2.0 Production on Mon Apr 15 21:10:58 2013
Copyright (c) 1982, 2010, Oracle. All rights reserved.

SQL> connect ad Doyle;
Enter password:
Connected.
SQL> set serveroutput on
SQL> CREATE OR REPLACE PROCEDURE question3_lab_4 (firstname IN VARCHAR2, surname
IN VARCHAR2)
2 IS
3     fullname VARCHAR2(50);
4     byte_numbers NUMBER;
5 BEGIN
6     fullname := firstname || ' ' || surname;
7     DBMS_OUTPUT.PUT_LINE (TRIM(INITCAP (firstname)) || ' ' || TRIM(INITCA
P (surname)));
8     SELECT VSIZE(fullname) INTO byte_numbers FROM dual;
9     DBMS_OUTPUT.PUT_LINE(byte_numbers);
10
11 END question3_lab_4;
12 /

Procedure created.

SQL>
SQL> BEGIN
2     question3_lab_4 (' ANDREW ', ' doyle ');
3
4 END;
5 /
Andrew Doyle
20

PL/SQL procedure successfully completed.
```

QUESTION 4

```
1| CREATE OR REPLACE PROCEDURE question4_lab_4 (vowels IN VARCHAR2)
2| IS
3|   vowelreplace VARCHAR2(50);
4| BEGIN
5|
6|   SELECT TRANSLATE(vowels,'waeiou', 'w') INTO vowelreplace FROM dual;
7|   DBMS_OUTPUT.PUT_LINE (vowelreplace);
8|
9| END question4_lab_4;
10| /
11|
12| BEGIN
13|   question4_lab_4 (' Quick frozen fox jumps over the lazy dog');
14|
15| END;
16| /
```

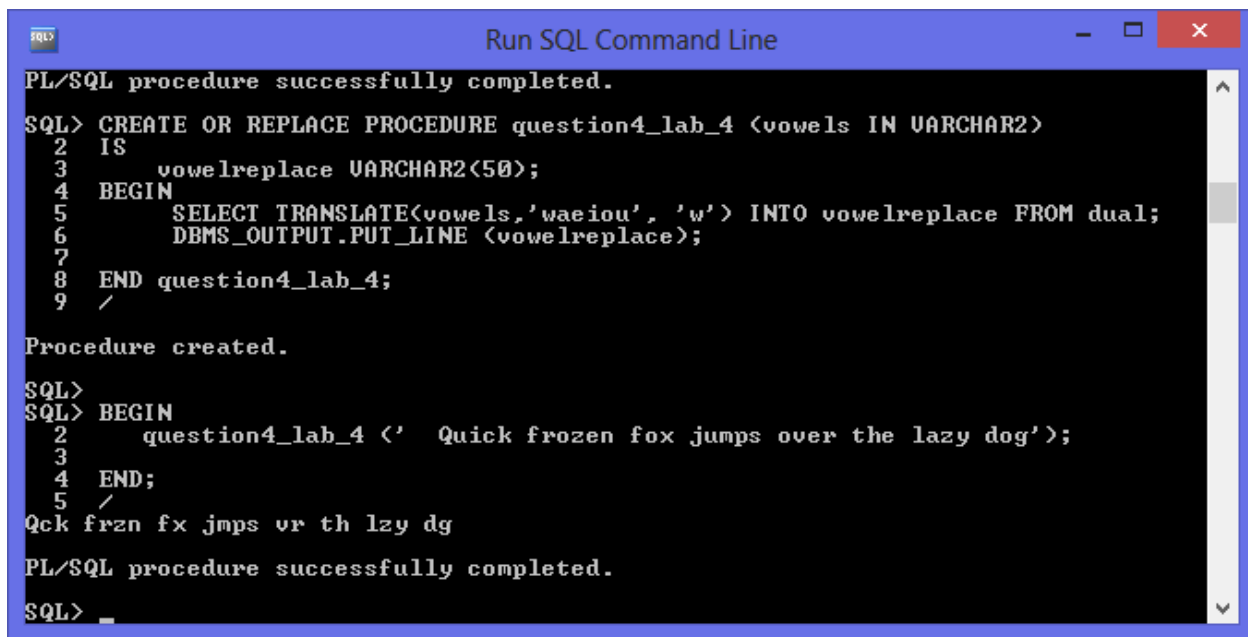
This question demonstrates the functionality of string or character replacement within a procedure. This could have been achieved with the **REPLACE** function as suggested in the Lab Book, however **REPLACE** lets you substitute a single string for another single string, whilst the function **TRANSLATE** allows several single-character one-one substitutions in one operation, as outlined in the Oracle Manual.

The procedure is created with one parameter, the variable **vowels** which is used to pass the input string to the procedure. The variable **vowelreplace** is declared on Line 3 in the declaration section (*/S*). The **TRANSLATE** function is applied to **vowels** in a **SELECT** statement.

As shown on Line 6, **TRANSLATE** returns **vowels** with all occurrences of 'waeiou' replaced with 'z'. In other words, all vowels are removed. The concatenation of the **w** is necessary as Oracle would interpret an empty string as NULL and would return NULL. The **w** is the author's chosen concatenated character; it could be any character.

The output of the aforementioned **TRANSLATE** function is copied into the variable **vowelreplace** which was declared earlier in the procedure. On line 7, the procedure **DBMS_OUTPUT.PUT_LINE**'s **vowelreplace**.

Outside of the procedure, the procedure is called in an anonymous block, where the input string is specified (Line 13).

Figure 4.1 – Output of character replacement procedure

```
Run SQL Command Line
PL/SQL procedure successfully completed.
SQL> CREATE OR REPLACE PROCEDURE question4_lab_4 (<vowels IN VARCHAR2>
2  IS
3      vowelreplace VARCHAR2(50);
4  BEGIN
5      SELECT TRANSLATE(<vowels,'waeiou','w') INTO vowelreplace FROM dual;
6      DBMS_OUTPUT.PUT_LINE (<vowelreplace>);
7
8  END question4_lab_4;
9  /
Procedure created.
SQL>
SQL> BEGIN
2      question4_lab_4 (<' Quick frozen fox jumps over the lazy dog'>);
3
4  END;
5  /
Qck frzn fx jmps vr th lzy dg
PL/SQL procedure successfully completed.
SQL> _
```

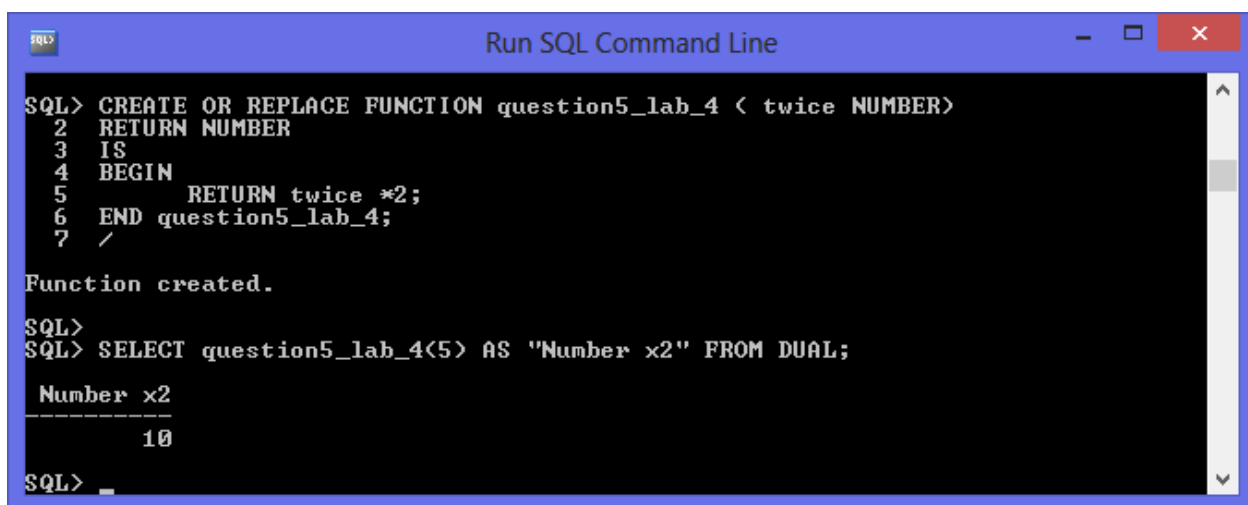
QUESTION 5

```
1| CREATE OR REPLACE FUNCTION question5_lab_4 ( twice NUMBER)
2| RETURN NUMBER
3| IS
4| BEGIN
5|     RETURN twice *2;
6| END question5_lab_4;
7| /
8|
9| SELECT question5_lab_4(5) AS "Number x2" FROM DUAL;
```

This question demonstrates a function that inputs a number and returns the double value of that number. The function is declared with one parameter, **twice**, which is used to pass the number into the function. In the function body, **twice *2** is returned.

As shown on line 9, the function is called using a **SELECT** statement with the value 5 passed into its parentheses. The **AS** keyword is used to provide a descriptive column heading for the output as shown in Figure 5.1.

Figure 5.1 – Number x 2



The screenshot shows a window titled "Run SQL Command Line" with a black background and white text. The SQL commands and their output are as follows:

```
SQL> CREATE OR REPLACE FUNCTION question5_lab_4 ( twice NUMBER)
2  RETURN NUMBER
3  IS
4  BEGIN
5      RETURN twice *2;
6  END question5_lab_4;
7  /

Function created.

SQL>
SQL> SELECT question5_lab_4(5) AS "Number x2" FROM DUAL;

Number x2
-----
       10

SQL> _
```

QUESTION 6

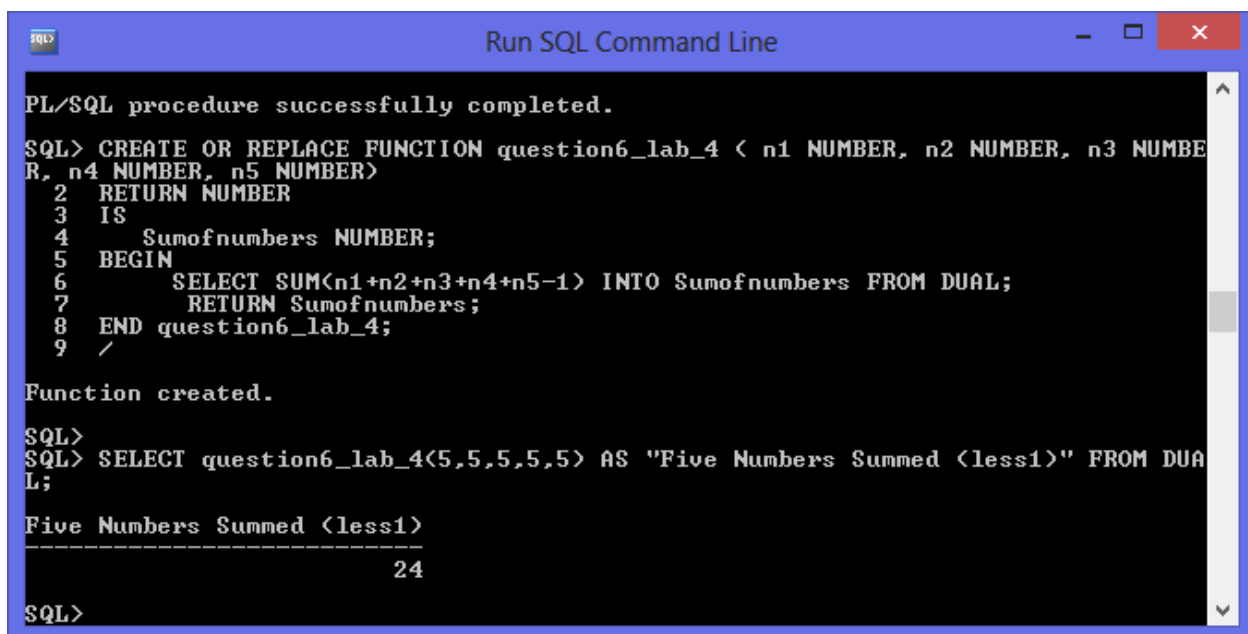
Simple Method

```
1| CREATE OR REPLACE FUNCTION question6_lab_4 ( n1 NUMBER, n2 NUMBER, n3 NUMBER, n4 NUMBER, n5  
NUMBER)  
2| RETURN NUMBER  
3| IS  
4| Sumofnumbers NUMBER;  
5| BEGIN  
6| SELECT SUM(n1+n2+n3+n4+n5-1) INTO Sumofnumbers FROM DUAL;  
7| RETURN Sumofnumbers;  
8| END question6_lab_4;  
9| /  
10|  
11| SELECT question6_lab_4(5,5,5,5,5) AS "Five Numbers Summed (less1)" FROM DUAL;
```

This question demonstrates a function that takes 5 numbers as input and returns the sum of them all, minus 1. The function is declared with five parameters, one for each number. On line 2 it is said the function will have a type **RETURN NUMBER**. On line 4, the variable *Sumofnumbers* is declared.

In the body, as shown on line 6, the number variables are summed with 1 subtracted at the end. The **SUM** is selected and inserted **INTO** the *Sumofnumber* variable, which is returned from the function on line 7. The function is then called on line 11 using a **SELECT** statement, with the desired number passed into the parentheses and the column heading renamed using **AS**.

Figure 6.1 – Simple Function Method



```
Run SQL Command Line

PL/SQL procedure successfully completed.

SQL> CREATE OR REPLACE FUNCTION question6_lab_4 ( n1 NUMBER, n2 NUMBER, n3 NUMBER, n4 NUMBER, n5 NUMBER)
2| RETURN NUMBER
3| IS
4| Sumofnumbers NUMBER;
5| BEGIN
6| SELECT SUM(n1+n2+n3+n4+n5-1) INTO Sumofnumbers FROM DUAL;
7| RETURN Sumofnumbers;
8| END question6_lab_4;
9| /

Function created.

SQL>
SQL> SELECT question6_lab_4(5,5,5,5,5) AS "Five Numbers Summed (less1)" FROM DUAL;

Five Numbers Summed (less1)
-----
24

SQL>
```

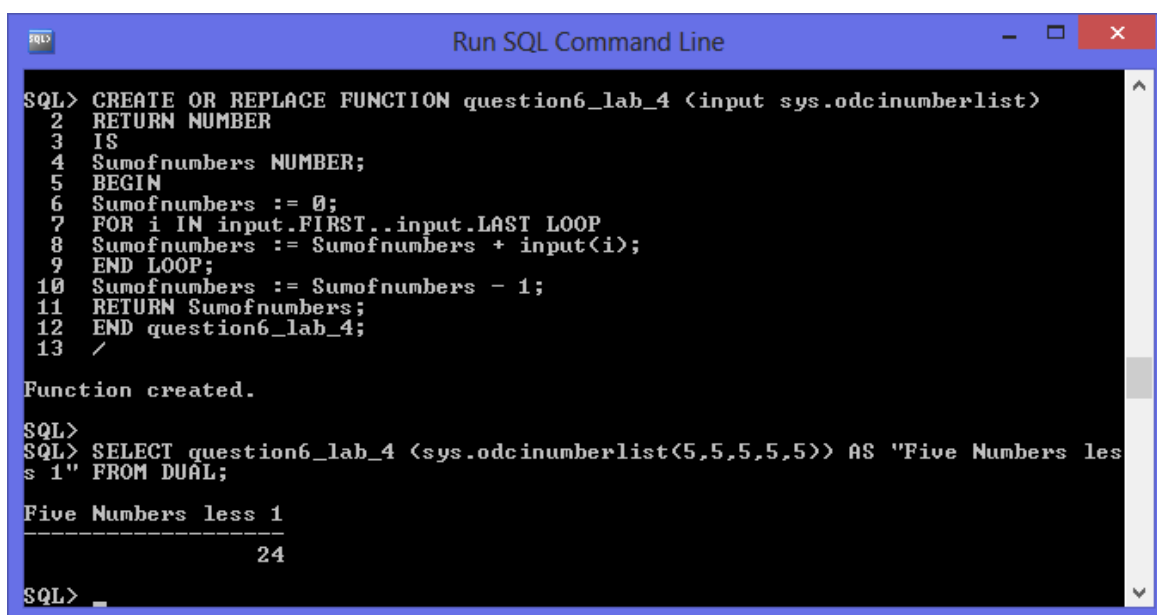
Flexible Function Method

This method is perhaps a better solution, as the number of parameters does not have to be specified in the function, instead **input sys.odcinumberlist** is specified as the input parameter to pass an array of numbers to the function. [Click here for more information on this capability](#). The variable **Sumofnumbers** is declared on line 4 and initialised to zero on line 6 (in the body).

On line 7, a for loop is implemented to run through the numbers and on line 8, the numbers are summed with the final sum stored in **Sumofnumbers** before the loop is ended on line 9. 1 is subtracted from the sum on line 10 and **Sumofnumbers** is returned. On Line 14, the function is called, with the desired input parameters (5,5,5,5,5); with the **AS** keyword being utilised again to provide a meaningful column heading.

```
1| CREATE OR REPLACE FUNCTION question6_lab_4 (input sys.odcinumberlist)
2| RETURN NUMBER
3| IS
4| Sumofnumbers NUMBER;
5| BEGIN
6| Sumofnumbers := 0;
7| FOR i IN input.FIRST..input.LAST LOOP
8| Sumofnumbers := Sumofnumbers + input(i);
9| END LOOP;
10| Sumofnumbers := Sumofnumbers - 1;
11| RETURN Sumofnumbers;
12| END question6_lab_4;
13| /
14| SELECT question6_lab_4 (sys.odcinumberlist(5,5,5,5,5)) AS "Five Numbers less 1" FROM DUAL;
```

Figure 6.2 – Flexible Function Method



```
Run SQL Command Line

SQL> CREATE OR REPLACE FUNCTION question6_lab_4 <input sys.odcinumberlist>
2  RETURN NUMBER
3  IS
4  Sumofnumbers NUMBER;
5  BEGIN
6  Sumofnumbers := 0;
7  FOR i IN input.FIRST..input.LAST LOOP
8  Sumofnumbers := Sumofnumbers + input(i);
9  END LOOP;
10 Sumofnumbers := Sumofnumbers - 1;
11 RETURN Sumofnumbers;
12 END question6_lab_4;
13 /

Function created.

SQL>
SQL> SELECT question6_lab_4 <sys.odcinumberlist(5,5,5,5,5)> AS "Five Numbers less 1" FROM DUAL;

Five Numbers less 1
-----
                24

SQL> _
```

QUESTION 7

-- Package Specification

```
1| CREATE OR REPLACE PACKAGE q7_package
2| AS
3|   FUNCTION q7_overloading ( n1 NUMBER, n2 NUMBER) RETURN NUMBER;
4|   FUNCTION q7_overloading ( n1 NUMBER, n2 NUMBER, n3 NUMBER) RETURN NUMBER;
5|   FUNCTION q7_overloading ( n1 NUMBER, n2 NUMBER, n3 NUMBER, n4 NUMBER) RETURN NUMBER;
6|   FUNCTION q7_overloading ( n1 NUMBER, n2 NUMBER, n3 NUMBER, n4 NUMBER, n5 NUMBER) RETURN
NUMBER;
7| END;
8| /
9|
10| CREATE OR REPLACE PACKAGE BODY q7_package
11| AS
12|   FUNCTION q7_overloading ( n1 NUMBER, n2 NUMBER)
13|   RETURN NUMBER
14|   IS
15|     Sumofnumbers NUMBER;
16|   BEGIN
17|     SELECT SUM(n1+n2-1) INTO Sumofnumbers FROM DUAL;
18|     RETURN Sumofnumbers;
19|   END q7_overloading;
20|
21|   FUNCTION q7_overloading (n1 NUMBER, n2 NUMBER, n3 NUMBER)
22|   RETURN NUMBER
23|   IS
24|     Sumofnumbers NUMBER;
25|   BEGIN
26|     SELECT SUM(n1+n2+n3-1) INTO Sumofnumbers FROM DUAL;
27|     RETURN Sumofnumbers;
28|   END q7_overloading;
29|
30|   FUNCTION q7_overloading (n1 NUMBER, n2 NUMBER, n3 NUMBER, n4 NUMBER)
31|   RETURN NUMBER
32|   IS
33|     Sumofnumbers NUMBER;
34|   BEGIN
35|     SELECT SUM(n1+n2+n3+n4-1) INTO Sumofnumbers FROM DUAL;
36|     RETURN Sumofnumbers;
37|   END q7_overloading;
```



```
38| FUNCTION q7_overloading (n1 NUMBER, n2 NUMBER, n3 NUMBER, n4 NUMBER, n5 NUMBER)
39| RETURN NUMBER
40| IS
41|   Sumofnumbers NUMBER;
42| BEGIN
43|   SELECT SUM(n1+n2+n3+n4+n5-1) INTO Sumofnumbers FROM DUAL;
44|   RETURN Sumofnumbers;
45| END q7_overloading;
46|
47| END;
48| /
```

Calling on functions created in the package

```
49| DECLARE
50|   result1 NUMBER;
51|   result2 NUMBER;
52|   result3 NUMBER;
53|   result4 NUMBER;
54| BEGIN
55|   result1 := q7_package.q7_overloading (5,5);
56|   result2 := q7_package.q7_overloading (5,5,5);
57|   result3 := q7_package.q7_overloading (5,5,5,5);
58|   result4 := q7_package.q7_overloading5,5,5,5,5);
59|   DBMS_OUTPUT.PUT_LINE(result1);
60|   DBMS_OUTPUT.PUT_LINE(result2);
61|   DBMS_OUTPUT.PUT_LINE(result3);
62|   DBMS_OUTPUT.PUT_LINE(result4);
62| END;
63| /
```

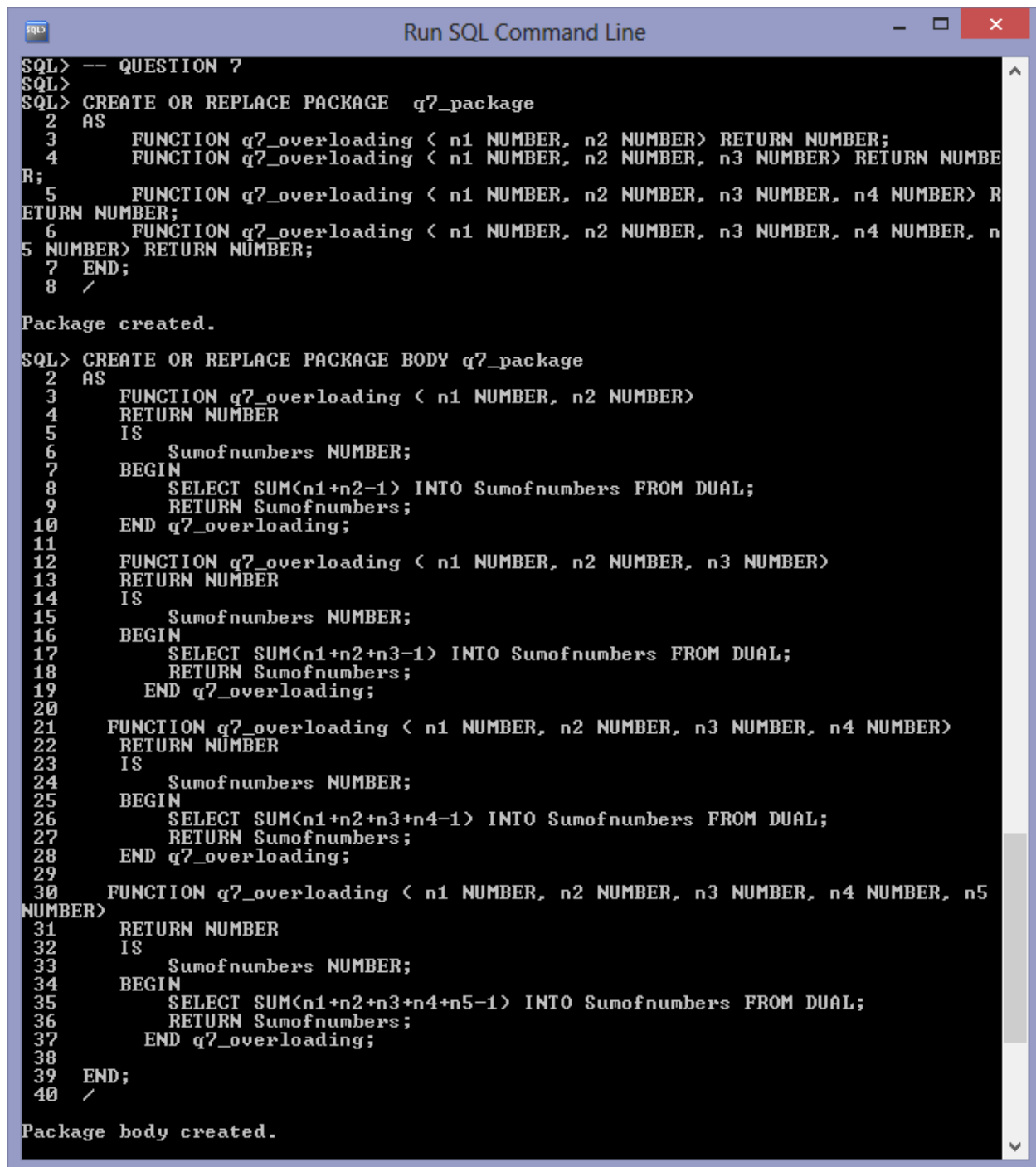
This question demonstrates the use of a package. A package is a container for SQL code which may contain functions and procedures to help manage them into logical groups; very useful when a system may have hundreds or thousands of functions.

There are two objects in the creation of a package. The first is the package specification, which is the visible part of the function which contains all information that code accessing the function needs to know (such as the function name, parameters, return type). The function code itself is placed within the package body.

The package specification is found between lines 1 – 8. The package is declared and named **q7_package**, and in the **AS** section, functions and their parameters are listed. Note that whilst the function names are the same, the amount of parameters passed to the function differ, so the functions are distinguishable.

In the package body, from Line 10 to 48, the function code is specified, with the same simple method utilised in question 6 for each of the four functions.

Outside of the package, we can call on the functions within the package as shown on lines 49 – 63. Variables are declared e.g. **result1** which are assigned the returned value of the packagename.functionname(parameters passed) – see line 55 for the first example; **result1** is then output on line 59. See Figure 7.2 for the output to the screen.

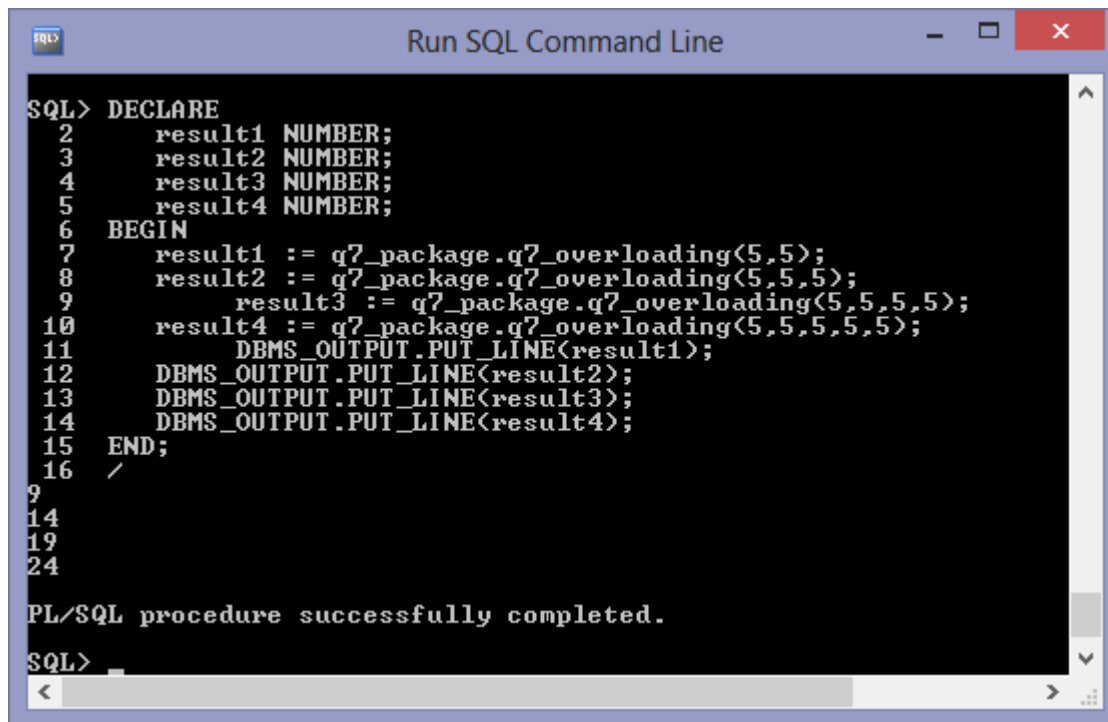
Figure 7.1 – Creating Package

```
SQL> -- QUESTION 7
SQL>
SQL> CREATE OR REPLACE PACKAGE q7_package
2 AS
3     FUNCTION q7_overloading < n1 NUMBER, n2 NUMBER> RETURN NUMBER;
4     FUNCTION q7_overloading < n1 NUMBER, n2 NUMBER, n3 NUMBER> RETURN NUMBE
R;
5     FUNCTION q7_overloading < n1 NUMBER, n2 NUMBER, n3 NUMBER, n4 NUMBER> R
ETURN NUMBER;
6     FUNCTION q7_overloading < n1 NUMBER, n2 NUMBER, n3 NUMBER, n4 NUMBER, n
5 NUMBER> RETURN NUMBER;
7 END;
8 /

Package created.

SQL> CREATE OR REPLACE PACKAGE BODY q7_package
2 AS
3     FUNCTION q7_overloading < n1 NUMBER, n2 NUMBER>
4     RETURN NUMBER
5     IS
6         Sumofnumbers NUMBER;
7     BEGIN
8         SELECT SUM(n1+n2-1) INTO Sumofnumbers FROM DUAL;
9         RETURN Sumofnumbers;
10    END q7_overloading;
11
12    FUNCTION q7_overloading < n1 NUMBER, n2 NUMBER, n3 NUMBER>
13    RETURN NUMBER
14    IS
15        Sumofnumbers NUMBER;
16    BEGIN
17        SELECT SUM(n1+n2+n3-1) INTO Sumofnumbers FROM DUAL;
18        RETURN Sumofnumbers;
19    END q7_overloading;
20
21    FUNCTION q7_overloading < n1 NUMBER, n2 NUMBER, n3 NUMBER, n4 NUMBER>
22    RETURN NUMBER
23    IS
24        Sumofnumbers NUMBER;
25    BEGIN
26        SELECT SUM(n1+n2+n3+n4-1) INTO Sumofnumbers FROM DUAL;
27        RETURN Sumofnumbers;
28    END q7_overloading;
29
30    FUNCTION q7_overloading < n1 NUMBER, n2 NUMBER, n3 NUMBER, n4 NUMBER, n5
NUMBER>
31    RETURN NUMBER
32    IS
33        Sumofnumbers NUMBER;
34    BEGIN
35        SELECT SUM(n1+n2+n3+n4+n5-1) INTO Sumofnumbers FROM DUAL;
36        RETURN Sumofnumbers;
37    END q7_overloading;
38
39 END;
40 /

Package body created.
```

Figure 7.2 – Output functions from a package

The screenshot shows a window titled "Run SQL Command Line" with a black background and white text. The text displays a PL/SQL procedure being executed. The procedure declares four variables (result1, result2, result3, result4) of type NUMBER. It then begins a block where each variable is assigned the result of a call to the q7_package.q7_overloading function with varying numbers of arguments (2, 3, 4, and 5). After each assignment, the DBMS_OUTPUT.PUT_LINE function is called to output the value of the variable. The procedure ends with an END statement. The output shows the procedure completed successfully, with line numbers 9, 14, 19, and 24 visible on the left margin.

```
SQL> DECLARE
2   result1 NUMBER;
3   result2 NUMBER;
4   result3 NUMBER;
5   result4 NUMBER;
6 BEGIN
7   result1 := q7_package.q7_overloading(5,5);
8   result2 := q7_package.q7_overloading(5,5,5);
9   result3 := q7_package.q7_overloading(5,5,5,5);
10  result4 := q7_package.q7_overloading(5,5,5,5,5);
11  DBMS_OUTPUT.PUT_LINE(result1);
12  DBMS_OUTPUT.PUT_LINE(result2);
13  DBMS_OUTPUT.PUT_LINE(result3);
14  DBMS_OUTPUT.PUT_LINE(result4);
15 END;
16 /
9
14
19
24
PL/SQL procedure successfully completed.
SQL>
```