Topics

1. Server Side Application Logic
2. PL/SQL Overview
3. Creating Standalone Procedures (CREATE PROCEDURE)
4. Passing data to and from a procedure
5. Anonymous Blocks
6. Conditional Statements (IF, CASE)
7. Select Into
8. Exception Handling

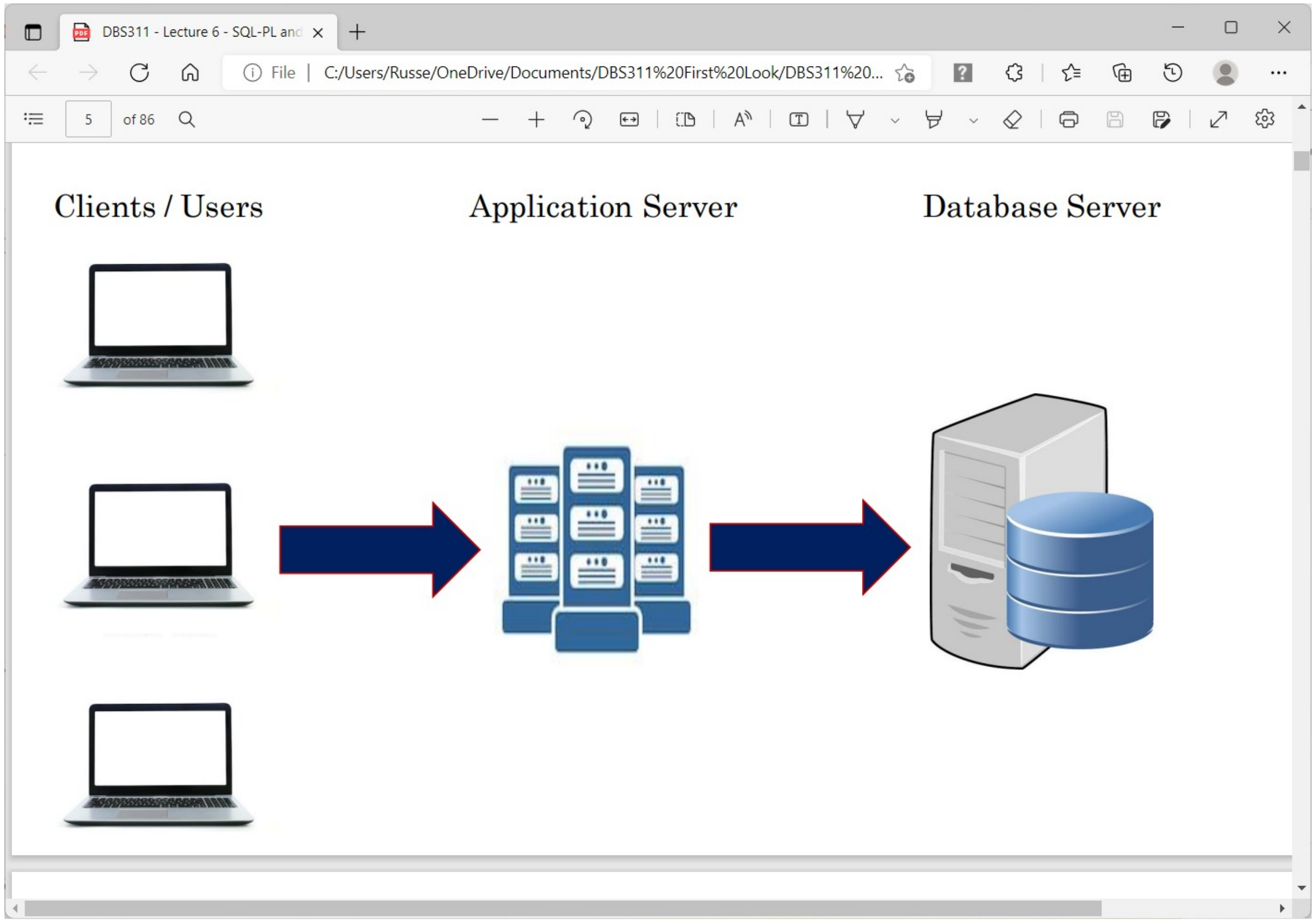
Server-Side Application Logic

- What is server-side application logic ?
  - This refers to components of applications which execute locally on the database server
  - This means the code is executing local to the data used by the application
- Why would we want to execute code on the database server ?
  - Minimize data movement between application location
  - Client / User environment
  - Application Server
  - Leverage data already in memory at the database server location

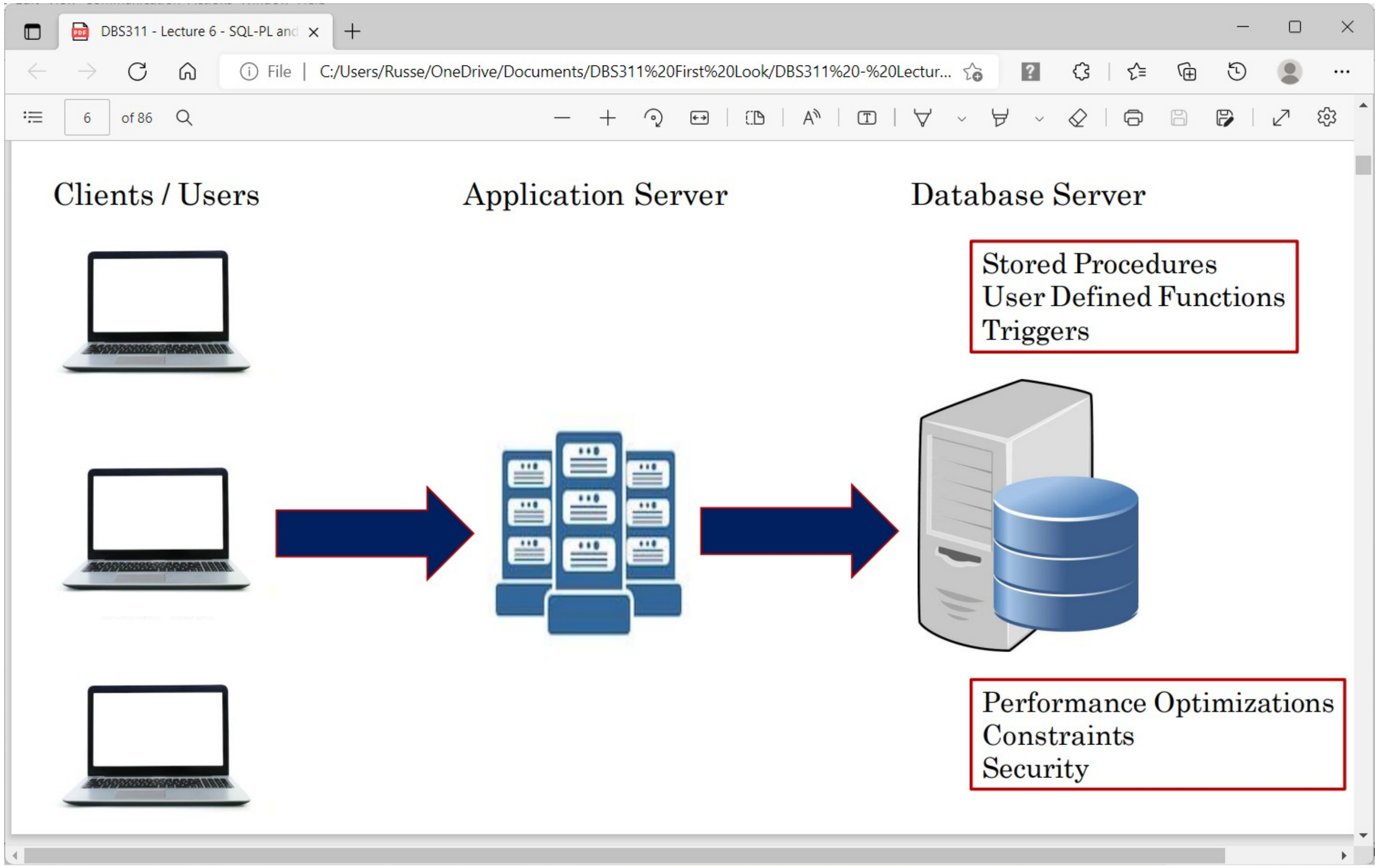
Server-Side Application Logic

- Why would we want to execute code on the database server ?
  - Massive performance benefits
  - Leverage optimizations from the database server
  - Parallelization
  - Shared-memory
  - Can be used by any application

Server-Side Application Logic


DBS311 - Lecture 6 - SQL-PL and
File | C:/Users/Russe/OneDrive/Documents/DBS311%20First%20Look/DBS311%20...
5
of 86
Clients / Users
Application Server
Database Server

Server-Side Application Logic


DBS311 - Lecture 6 - SQL-PL and
File
C:/Users/Russe/OneDrive/Documents/DBS311%20First%20Look/DBS311%20-%20Lectur...
6
of 86
Clients / Users
Application Server
Database Server

Stored Procedures
User Defined Functions
Triggers

Performance Optimizations
Constraints
Security

Server-Side Application Logic

- Stored Procedures
  - Application modules that perform functions
  - Execute <Procedure>
  - <Procedure>
  - CALL <Procedure>
- User Defined Function
  - A function that can be called from SQL
  - Built-in functions we have already discussed (MIN, AVG, SUBSTR, etc)
  - Now you can build your own
- SELECT FUNC(expression) FROM …..
- Trigger
  - Code to execute when an INSERT, UPDATE or DELETE take place
  - CREATE TRIGGER ON ……

Server-Side Application Logic

- Stored Procedures, User Defined Functions and Triggers can be developed using:
- General programming languages (ie: C, Java, etc)
- Database specific (native) languages
- Oracle (PL/SQL)
- Db2 (SQL/PL)
- Microsoft SQL Server (Transact-SQL)
- SQL PL is actually the SQL ANSII standard language
- PL/SQL is what you will use with Oracle
- There can be slight differences with what aspects of the stored procedure languages are available in each application object

PL/SQL Overview

- PL/SQL is a language with procedural constructs integrated with SQL that can be used to build complex applications.
- PL/SQL is executed in the database.
- PL/SQL can be used to create the following program units:
- Procedures
- Functions
- Packages

Creating Procedures/Functions

- In Oracle, a program can be written and stored in the database once and be accessed from any application program.
- There are two schema level standalone programs:
- Procedures
- are programs with no returning value
- Functions
- Are programs with a returning value
- When these programs are written and complied in a database they become the schema objects called stored procedures and stored functions.

Procedure/Function building Blocks

Procedures/Functions consists of the following basic PL/SQL block structures:

Declarative (optional)

Variables and constants are identified

Anonymous Block uses keyword DECLARE.

Executable (mandatory)

Contains the application logic. It starts with keyword BEGIN and finishes with the keyword END.

Exception handling (optional)

Starts with keyword EXCEPTION and handles error conditions that may occur in the executable part.

Procedure/Function building Blocks

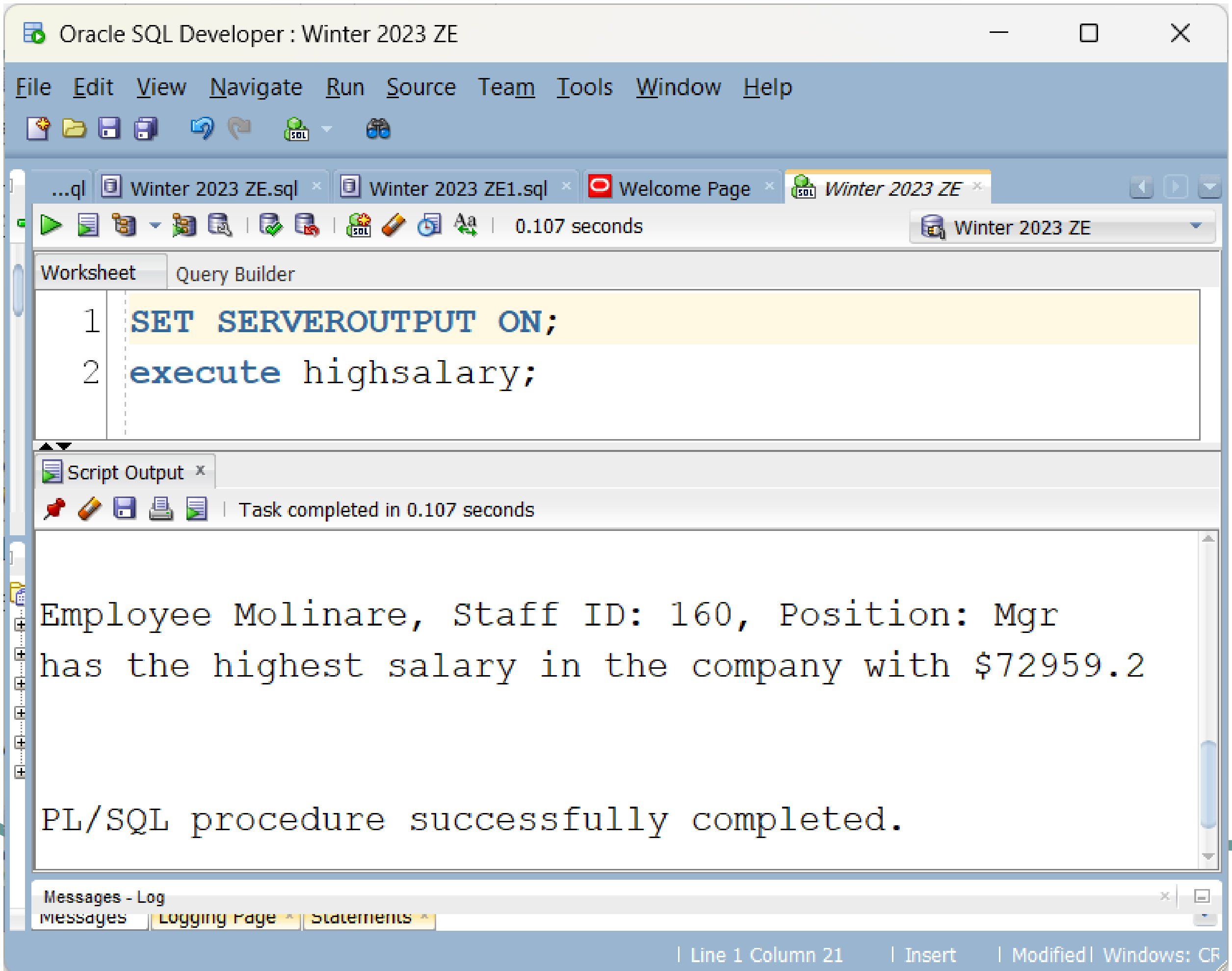
```
Header AS
[declaration statements
    ...]
BEGIN
    ...
[EXCEPTION
    ...]
END;
```

Create Procedures/Functions

- The following is the syntax to create a stored procedure or function:
- The keyword OR REPLACE recreates a function or a procedure if it already exists.
- Schema is the name of schema that contains the procedure/function
- CREATE OR REPLACE PROCEDURE schema.**procedure**_name
- (arg1 data_type, ...) AS
- BEGIN
- ....
- END procedure_name;
- CREATE OR REPLACE FUNCTION schema.**function**_name
- (arg1 data_type, ...) AS
- BEGIN
- ....
- RETURN END function_name;

PL/SQL

- A working procedure based on STAFF

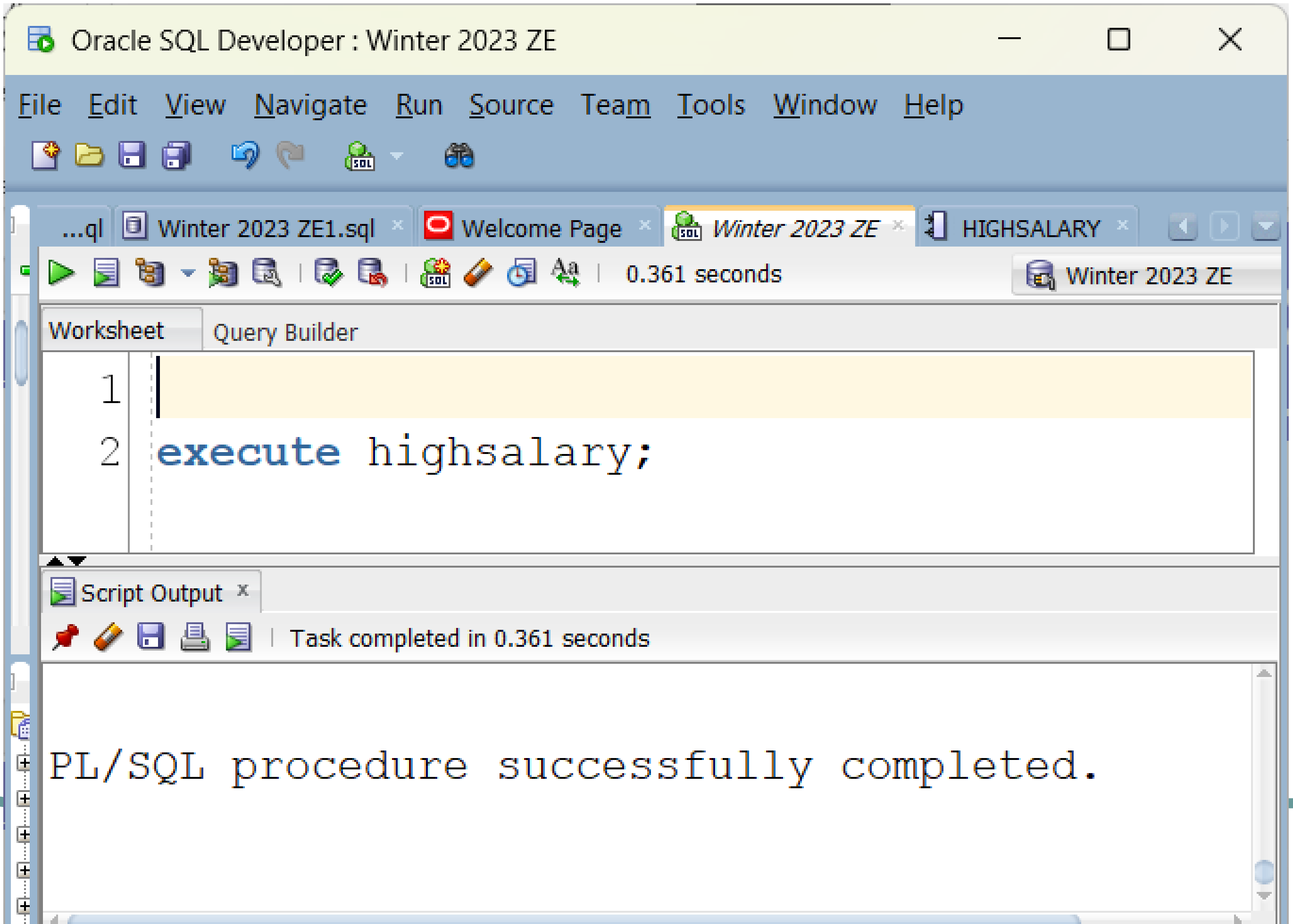


```
SET SERVEROUTPUT ON;
execute highsalary;
```

```
Employee Molinare, Staff ID: 160, Position: Mgr
has the highest salary in the company with $72959.2


PL/SQL procedure successfully completed.
```

Viewing the output

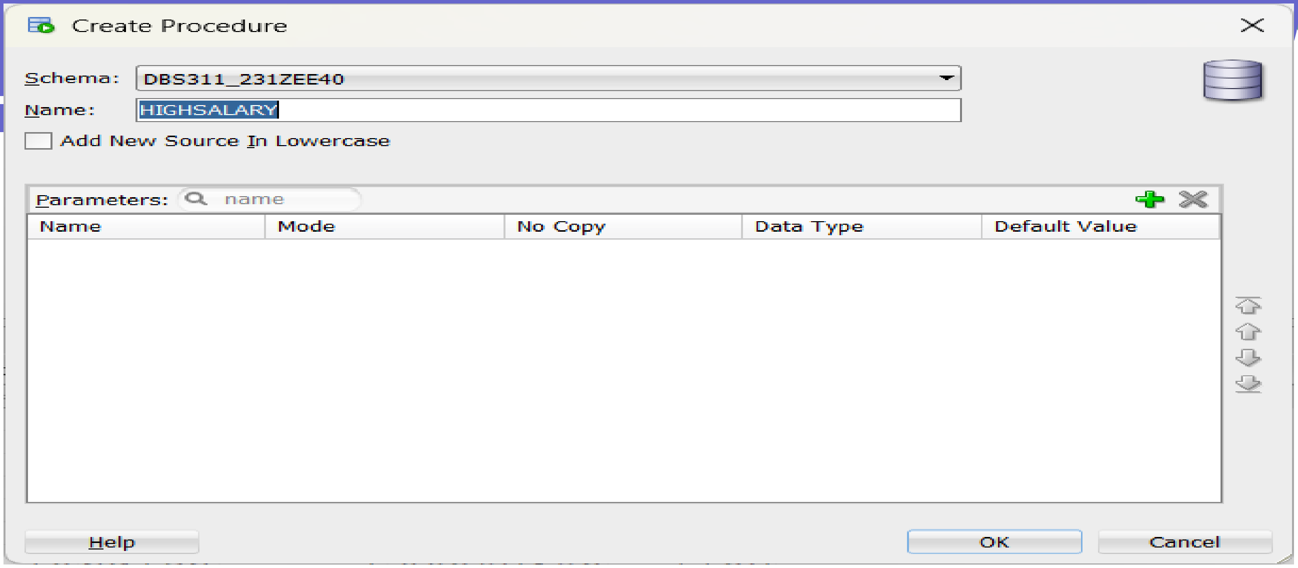
- Without SET SERVEROUTPUT ON;


Oracle SQL Developer : Winter 2023 ZE
File Edit View Navigate Run Source Team Tools Window Help
...ql
Winter 2023 ZE1.sql
Welcome Page
Winter 2023 ZE
HIGHSALARY
0.361 seconds
Winter 2023 ZE
Worksheet
Query Builder
Script Output
Task completed in 0.361 seconds

```

execute highsalary;
```

```
PL/SQL procedure successfully completed.
```

PL/SQL

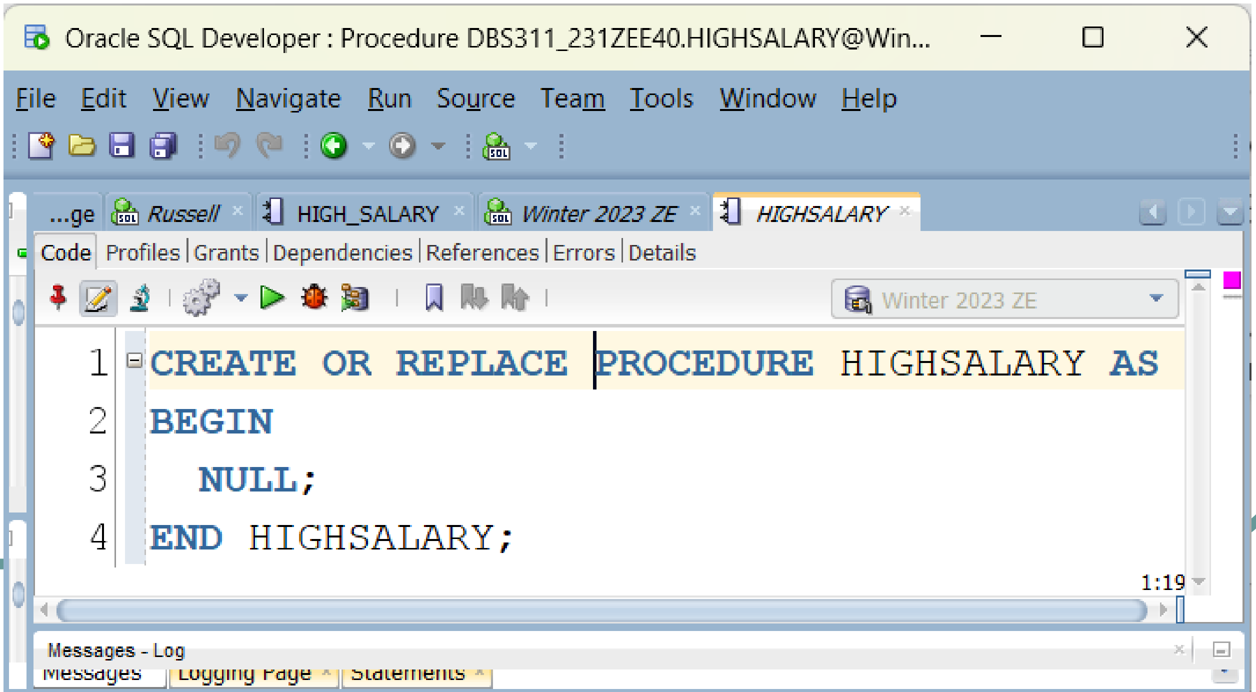

Create Procedure

Schema:	DBS311_231ZEE40
Name:	HIGHSALARY

☐ Add New Source In Lowercase

Parameters:

Name	Mode	No Copy	Data Type	Default Value

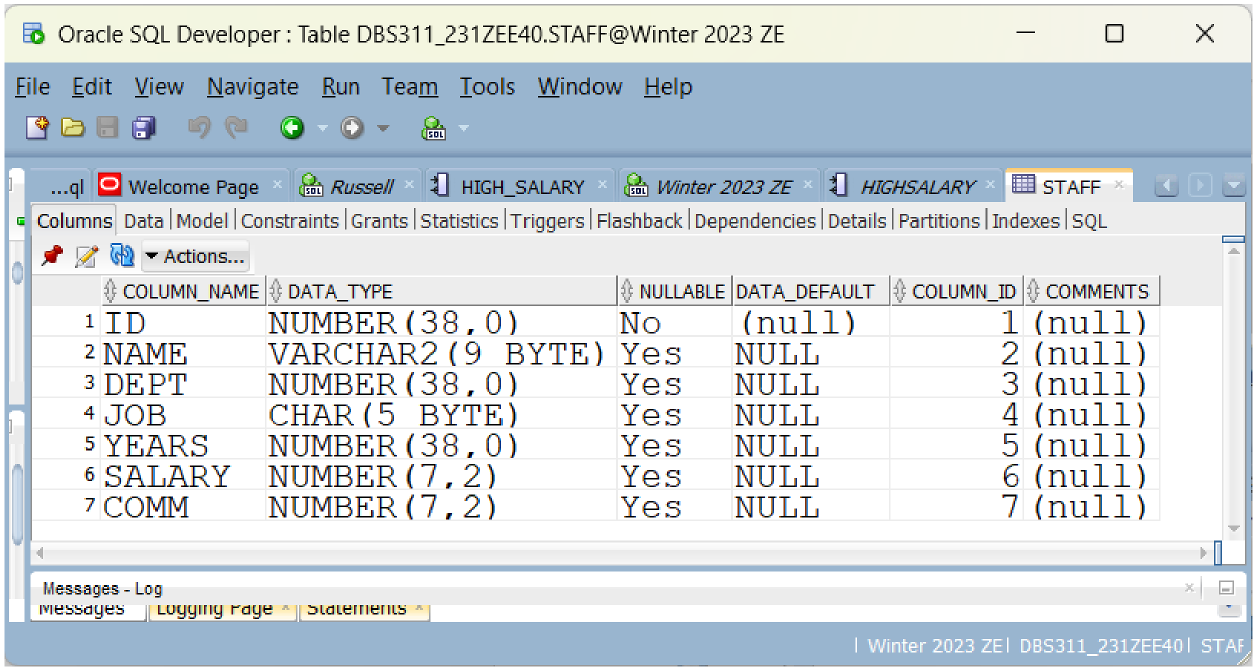

Oracle SQL Developer : Procedure DBS311_231ZEE40.HIGHSALARY@Win...

File Edit View Navigate Run Source Team Tools Window Help
...ge
Russell
HIGH_SALARY
Winter 2023 ZE
HIGHSALARY
Code Profiles Grants Dependencies References Errors Details
Winter 2023 ZE
1:19
Messages - Log
Messages
Logging Page
Statements

```
CREATE OR REPLACE PROCEDURE HIGHSALARY AS
BEGIN
  NULL;
END HIGHSALARY;
```

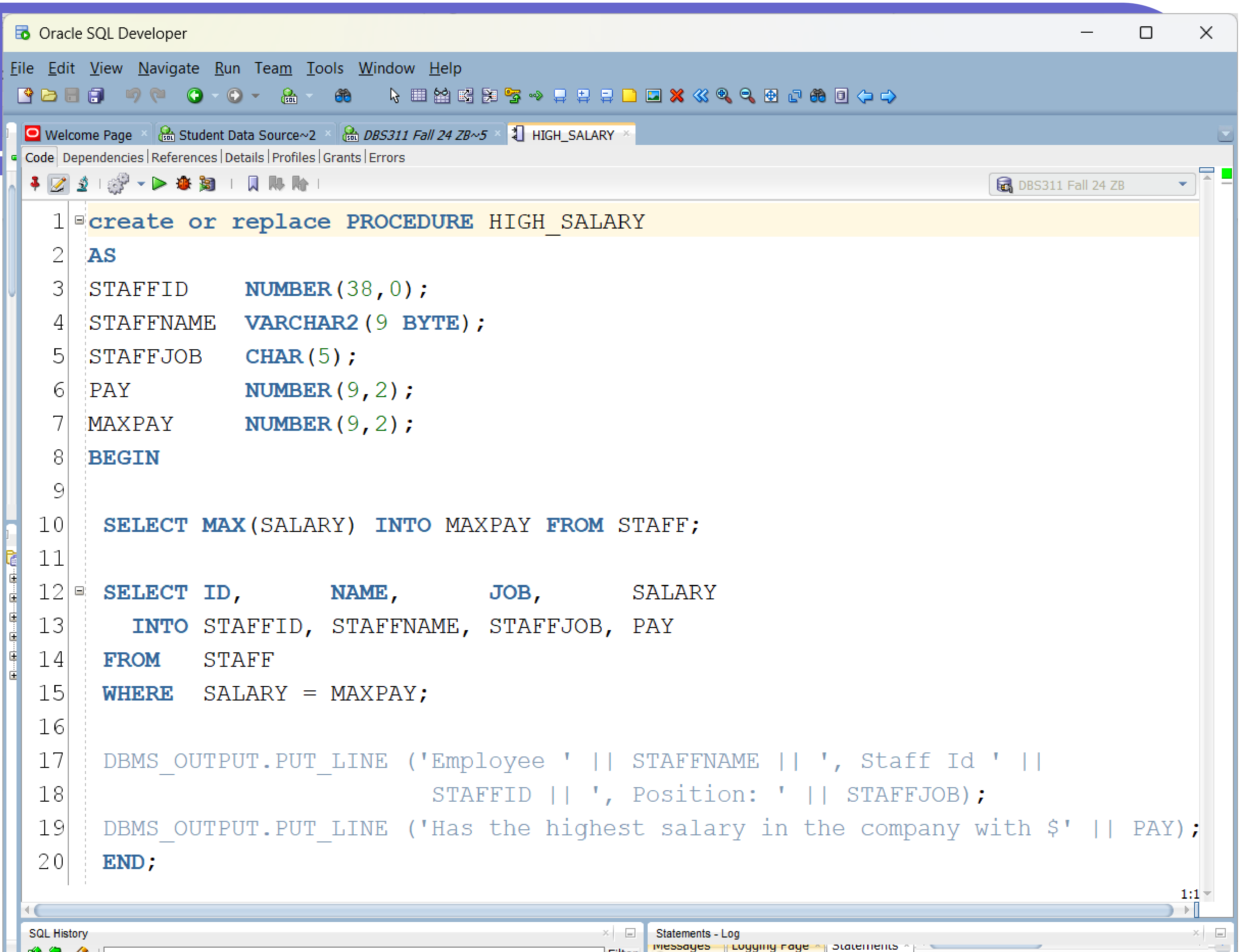
PL/SQL

- Variables in procedure should match variables in table



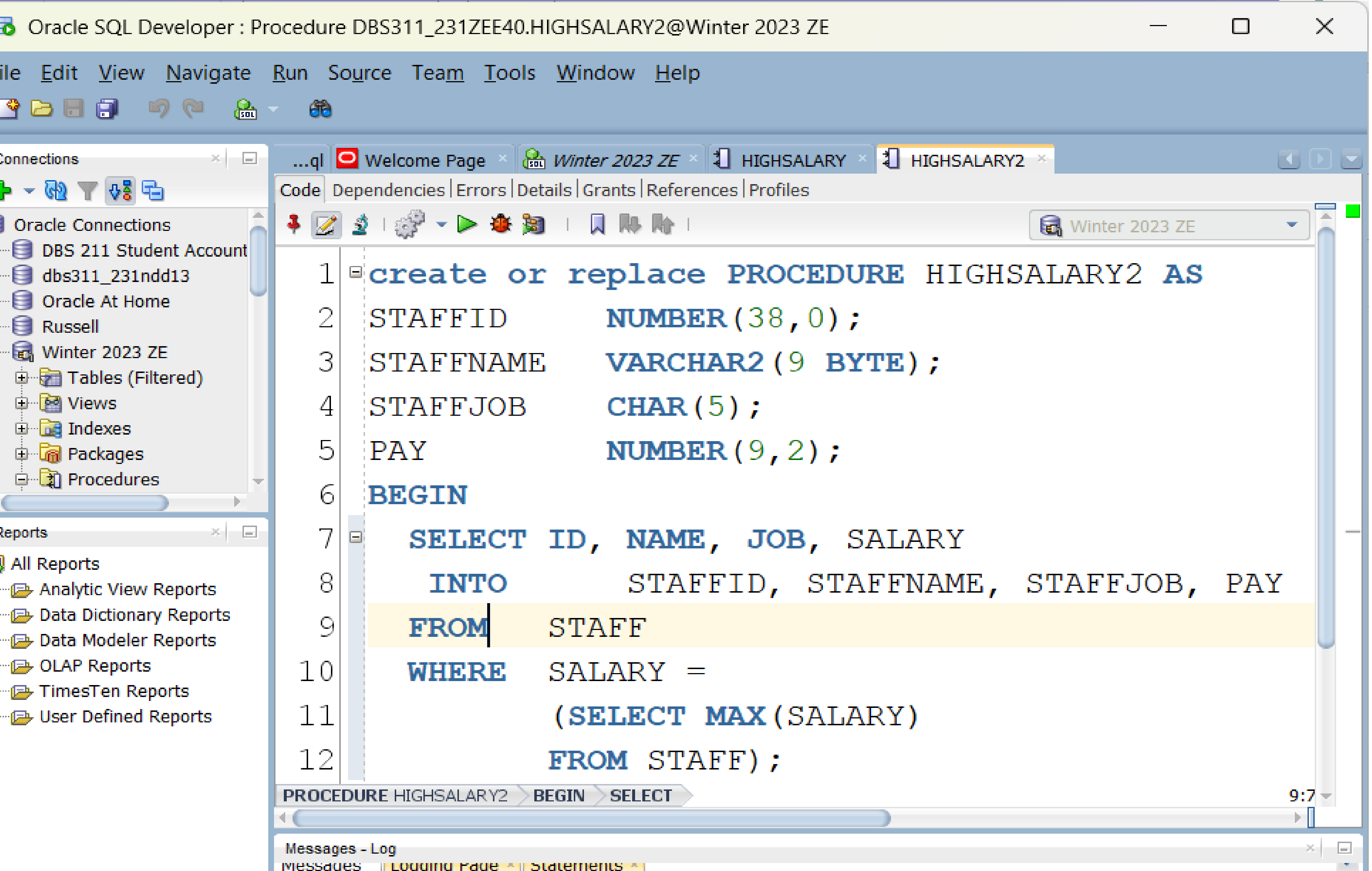
Oracle SQL Developer : Table DBS311_231ZEE40.STAFF@Winter 2023 ZE
File Edit View Navigate Run Team Tools Window Help
...ql | Welcome Page | Russell | HIGH_SALARY | Winter 2023 ZE | HIGHSALARY | STAFF
Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL
Actions...
Messages - Log
Messages | Logging Page | Statements
Winter 2023 ZE | DBS311_231ZEE40 | STAF

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	ID	NUMBER(38,0)	No	(null)	1	(null)
2	NAME	VARCHAR2(9 BYTE)	Yes	NULL	2	(null)
3	DEPT	NUMBER(38,0)	Yes	NULL	3	(null)
4	JOB	CHAR(5 BYTE)	Yes	NULL	4	(null)
5	YEARS	NUMBER(38,0)	Yes	NULL	5	(null)
6	SALARY	NUMBER(7,2)	Yes	NULL	6	(null)
7	COMM	NUMBER(7,2)	Yes	NULL	7	(null)



```sql
create or replace PROCEDURE HIGH_SALARY
AS
STAFFID     NUMBER(38,0);
STAFFNAME   VARCHAR2(9 BYTE);
STAFFJOB    CHAR(5);
PAY         NUMBER(9,2);
MAXPAY      NUMBER(9,2);
BEGIN

 SELECT MAX(SALARY) INTO MAXPAY FROM STAFF;

 SELECT ID,      NAME,      JOB,      SALARY
   INTO STAFFID, STAFFNAME, STAFFJOB, PAY
 FROM   STAFF
 WHERE  SALARY = MAXPAY;

 DBMS_OUTPUT.PUT_LINE ('Employee ' || STAFFNAME || ', Staff Id ' ||
                        STAFFID || ', Position: ' || STAFFJOB);
 DBMS_OUTPUT.PUT_LINE ('Has the highest salary in the company with $' || PAY);
 END;
```

PL/SQL - Different Solution



```
create or replace PROCEDURE HIGHSALARY2 AS
STAFFID      NUMBER(38,0);
STAFFNAME    VARCHAR2(9 BYTE);
STAFFJOB     CHAR(5);
PAY          NUMBER(9,2);
BEGIN
  SELECT ID, NAME, JOB, SALARY
   INTO       STAFFID, STAFFNAME, STAFFJOB, PAY
  FROM    STAFF
  WHERE   SALARY =
          (SELECT MAX(SALARY)
          FROM STAFF);
```

Arguments in a Procedure/Function

- A procedure/function may receive arguments.
- An argument has the following elements:
- Datatype
- Can be any datatype supported by PL/SQL.
- 1. IN 2. OUT 3. IN OUT
- IN indicates that the procedure has to receive a value for the argument.
- OUT indicates that the procedure/function passes a value for the argument back to the calling program.
- IN OUT indicates that procedure must receive a value for the argument and passes a value back to the calling program.
- By default, a parameter is an IN parameter with stored procedures
- Using DEFAULT keyword, you can define a value for an argument.

Stored Procedures with Parameters

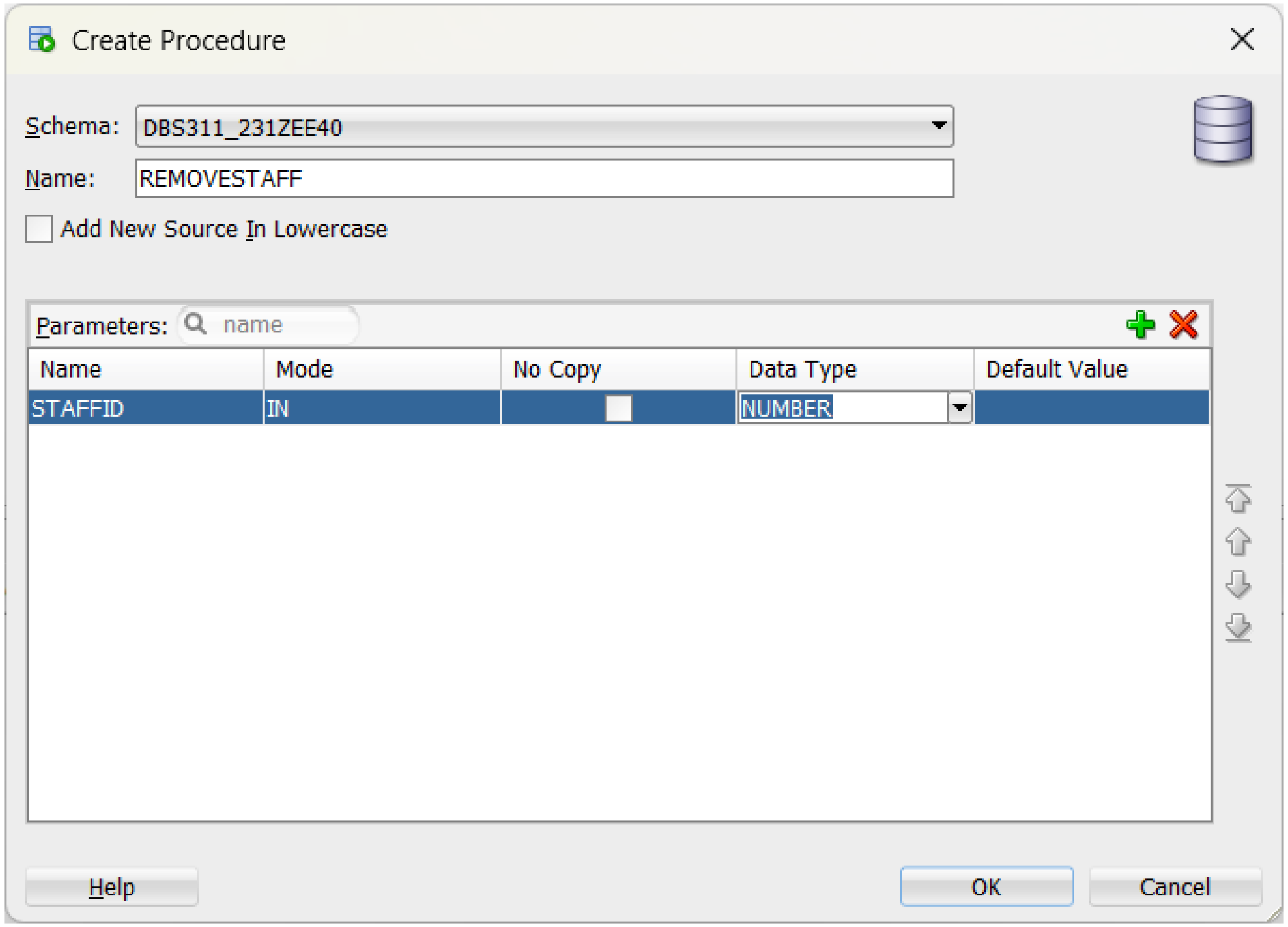
- In PL/SQL, we can pass parameters to stored procedures and functions.
- See the following syntax
- CREATE OR REPLACE PROCEDURE procedure_name(arg1 data_type, ...) AS
- BEGIN
- ....
- END procedure_name;
- The following stored procedure deletes the employee with Id given in the stored procedure parameter:
- CREATE PROCEDURE RemoveStaff
-             (STAFFID NUMBER) AS
- BEGIN
-    DELETE FROM STAFF
-    WHERE ID = STAFFID;
- END;

IN Parameters

- CREATE PROCEDURE RemoveStaff
-             (ID IN NUMBER) AS
- BEGIN
-    DELETE FROM STAFF
-    WHERE STAFF.ID = REMOVESTAFF.ID;
- END;

- When should IN be specified? – not necessary syntactically, but is clearer for readability
- This time ID needed to be qualified to differentiate between ID in the staff table from ID defined in the RemoveStaff table this is necessary

PL/SQL Paramater



Create Procedure

Schema: DBS311_231ZEE40

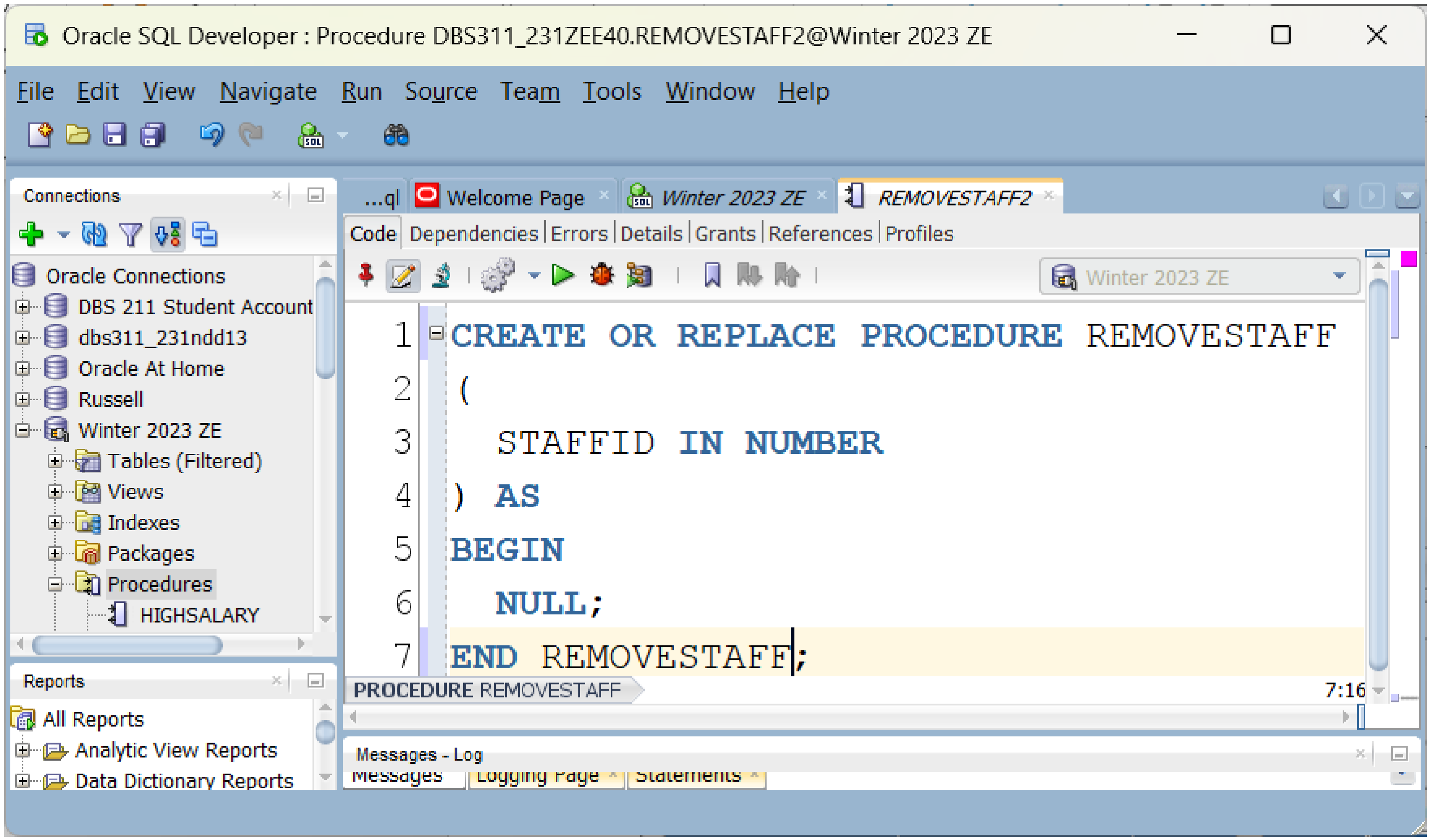
Name: REMOVESTAFF

☐ Add New Source In Lowercase

Parameters:

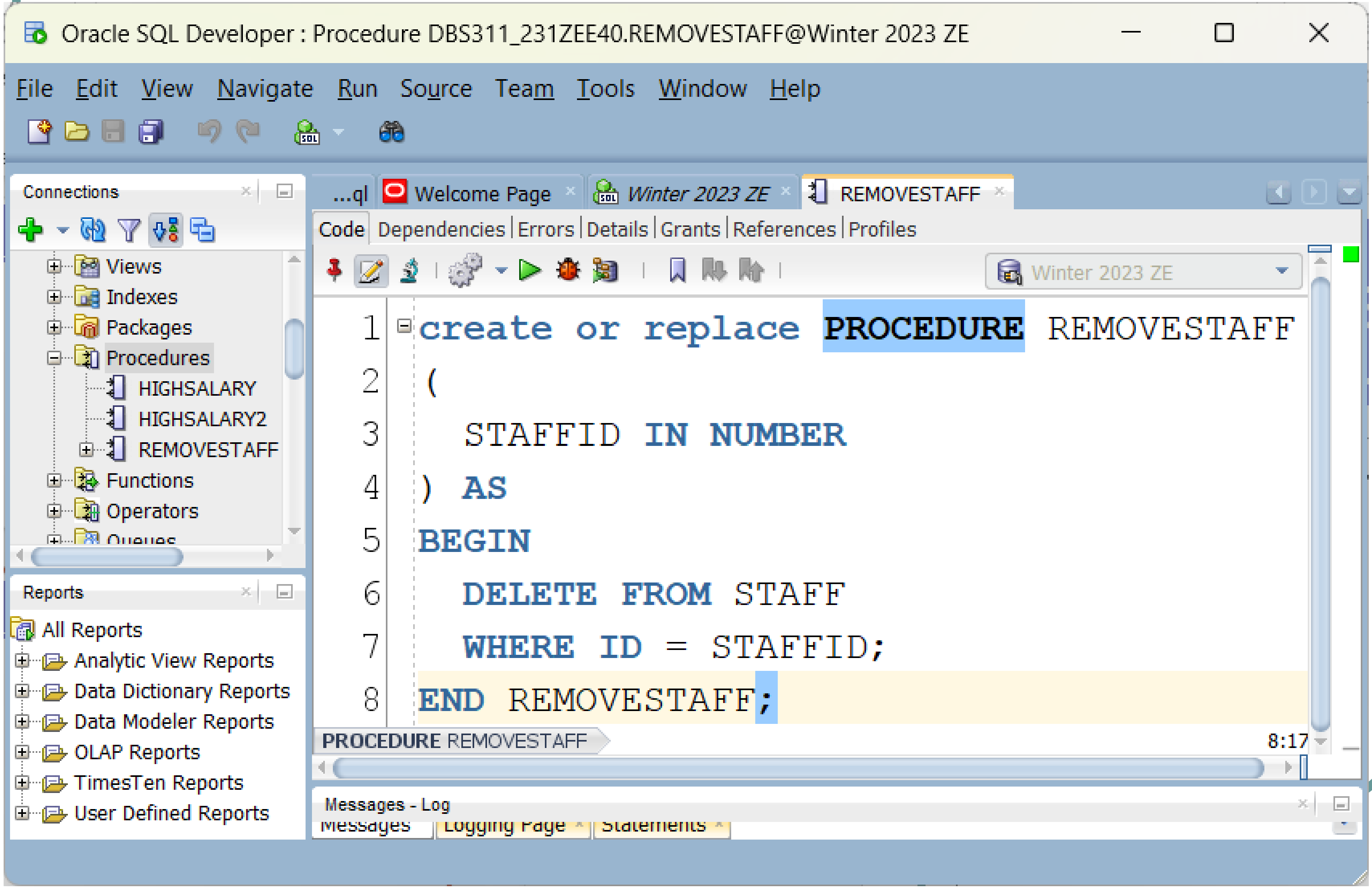
Name	Mode	No Copy	Data Type	Default Value
STAFFID	IN	☐	NUMBER	

PL/SQL Parameter



```
CREATE OR REPLACE PROCEDURE REMOVESTAFF
(
  STAFFID IN NUMBER
) AS
BEGIN
  NULL;
END REMOVESTAFF;
```

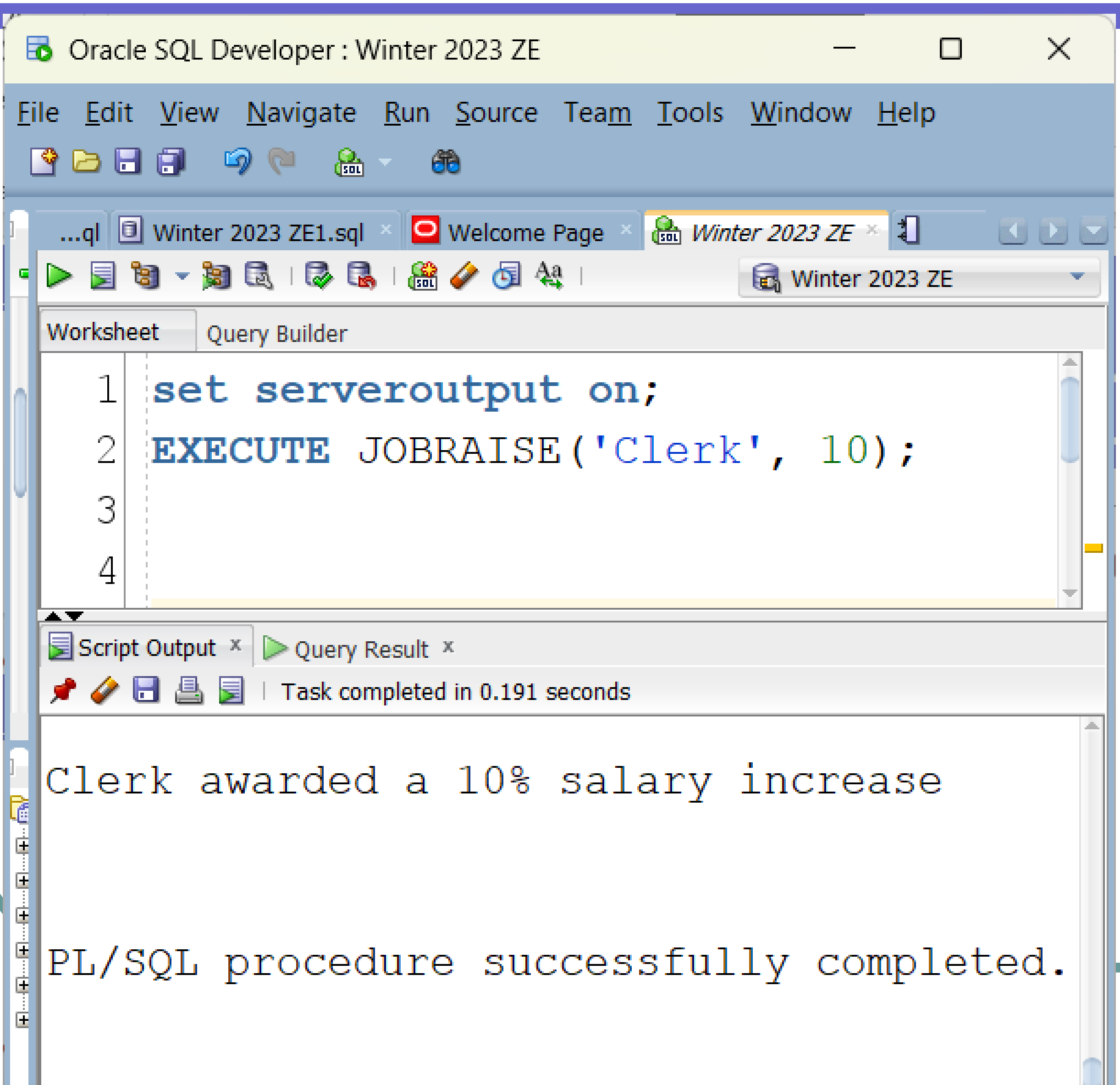
PL/SQL Parameter



Oracle SQL Developer : Procedure DBS311_231ZEE40.REMOVESTAFF@Winter 2023 ZE

```
create or replace PROCEDURE REMOVESTAFF
(
  STAFFID IN NUMBER
) AS
BEGIN
  DELETE FROM STAFF
  WHERE ID = STAFFID;
END REMOVESTAFF;
```

PL/SQL – Multiple parameters

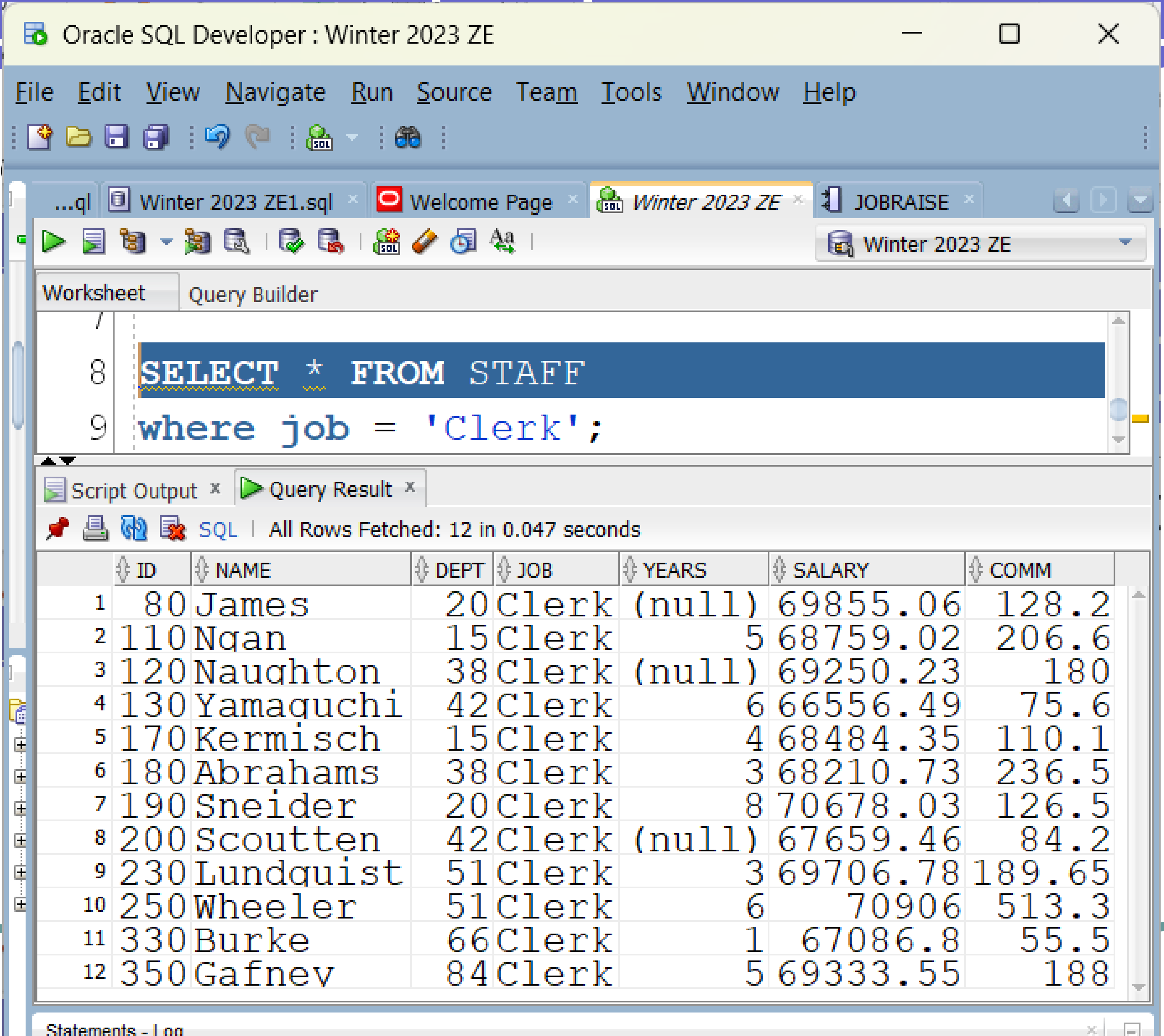

Oracle SQL Developer : Winter 2023 ZE
File Edit View Navigate Run Source Team Tools Window Help
...ql
Winter 2023 ZE1.sql
Welcome Page
Winter 2023 ZE
Winter 2023 ZE
Worksheet
Query Builder
Script Output
Query Result
Task completed in 0.191 seconds

```
set serveroutput on;
EXECUTE JOBRAISE('Clerk', 10);
```

```
Clerk awarded a 10% salary increase


PL/SQL procedure successfully completed.
```

PL/SQL – Multiple parameters

- Cl



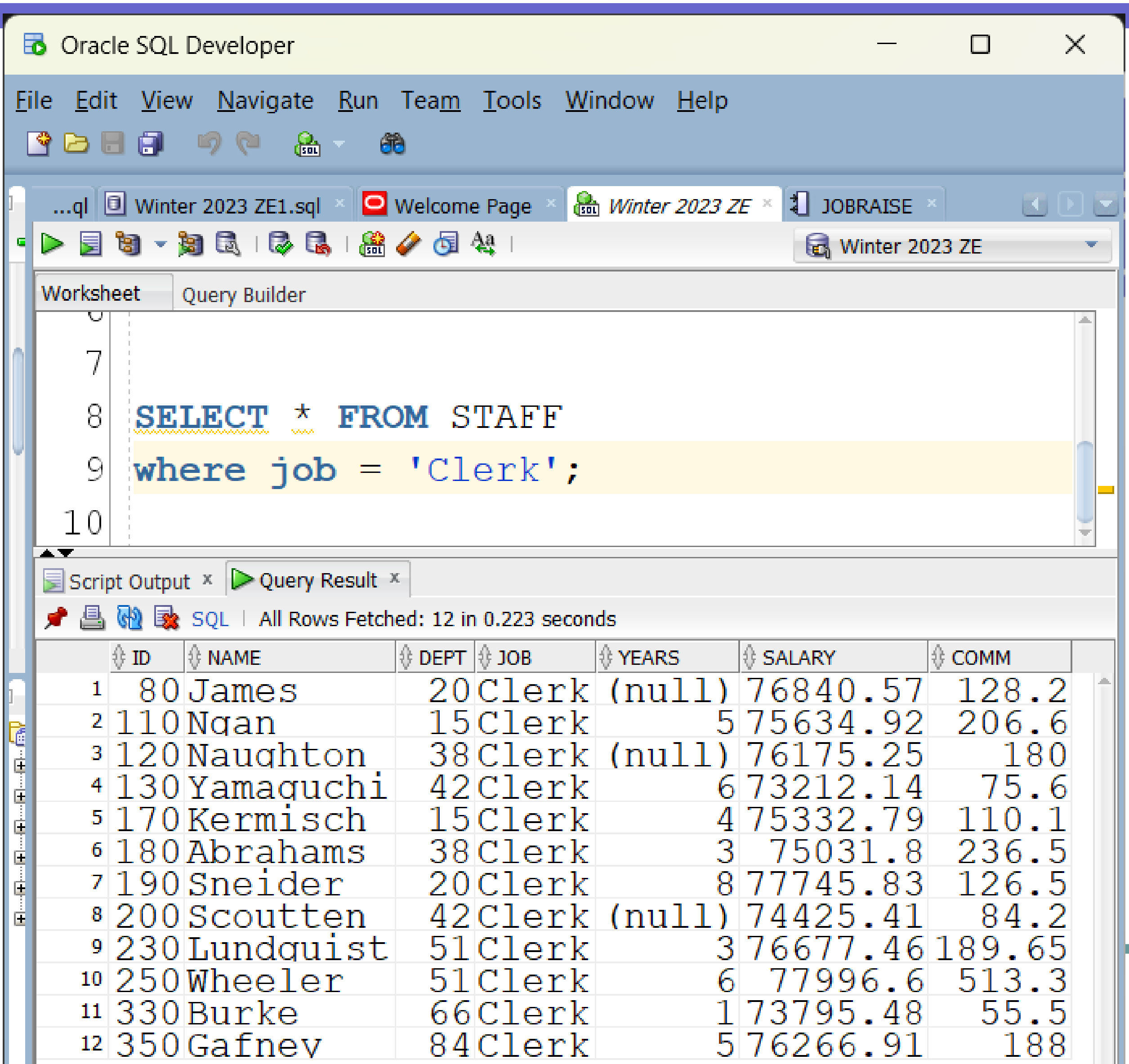
```
SELECT * FROM STAFF
where job = 'Clerk';
```

All Rows Fetched: 12 in 0.047 seconds

	ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
1	80	James	20	Clerk	(null)	69855.06	128.2
2	110	Ngan	15	Clerk	5	68759.02	206.6
3	120	Naughton	38	Clerk	(null)	69250.23	180
4	130	Yamaguchi	42	Clerk	6	66556.49	75.6
5	170	Kermisch	15	Clerk	4	68484.35	110.1
6	180	Abrahams	38	Clerk	3	68210.73	236.5
7	190	Sneider	20	Clerk	8	70678.03	126.5
8	200	Scoutten	42	Clerk	(null)	67659.46	84.2
9	230	Lundquist	51	Clerk	3	69706.78	189.65
10	250	Wheeler	51	Clerk	6	70906	513.3
11	330	Burke	66	Clerk	1	67086.8	55.5
12	350	Gafney	84	Clerk	5	69333.55	188

PL/SQL – Multiple parameters

re run



Oracle SQL Developer

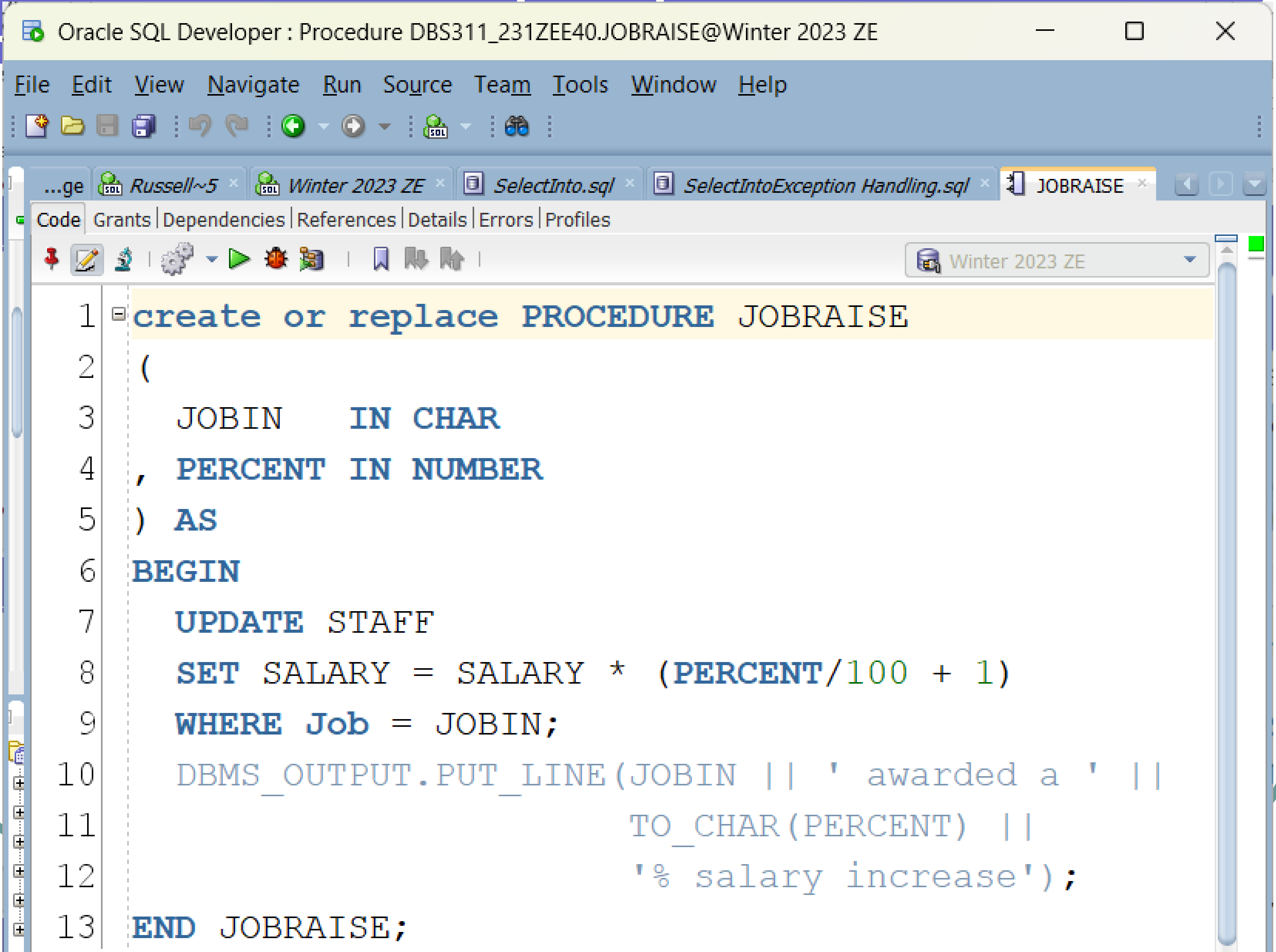
```
SELECT * FROM STAFF
where job = 'Clerk';
```

Script Output | Query Result

All Rows Fetched: 12 in 0.223 seconds

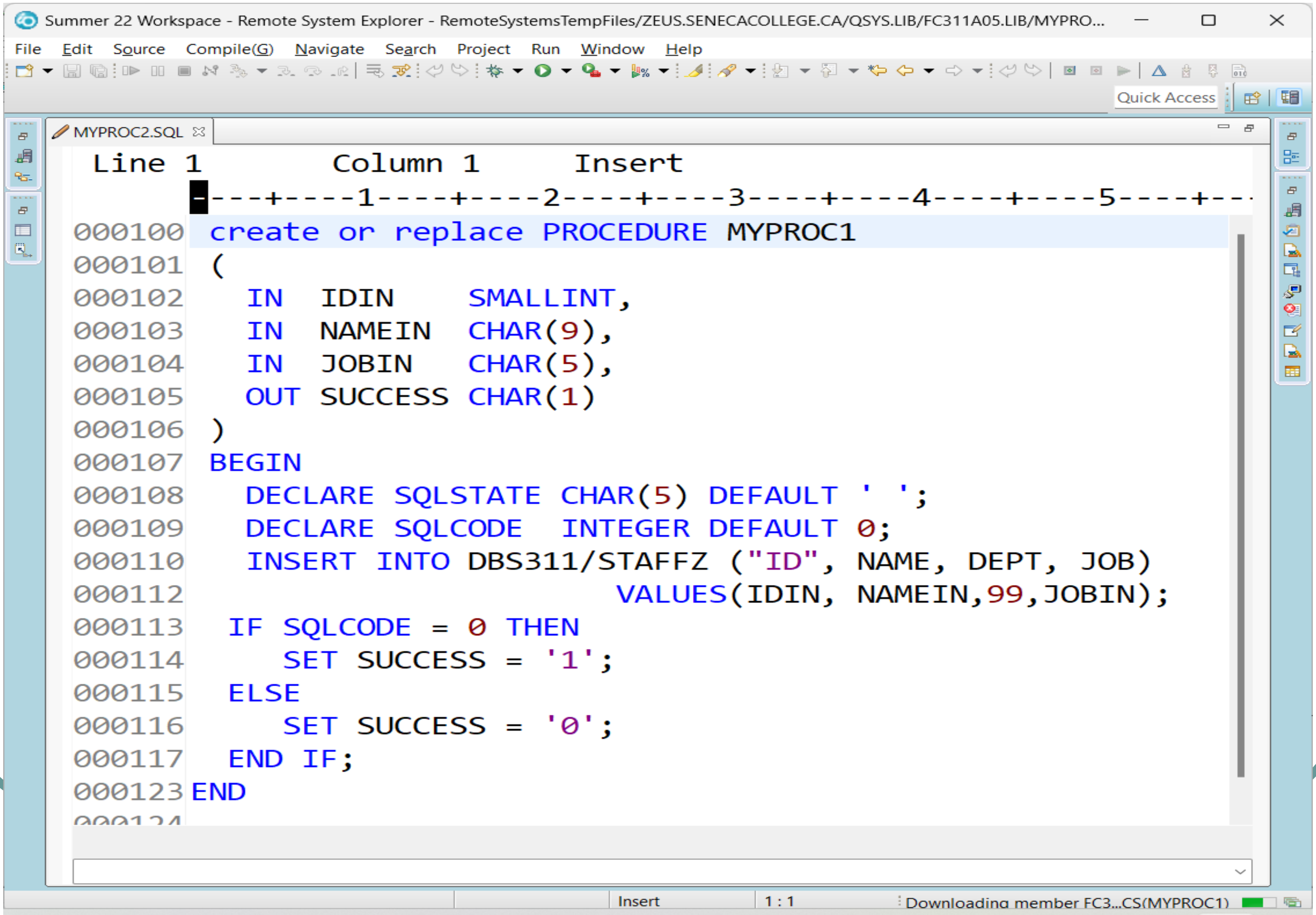
	ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
1	80	James	20	Clerk	(null)	76840.57	128.2
2	110	Ngan	15	Clerk	5	75634.92	206.6
3	120	Naughton	38	Clerk	(null)	76175.25	180
4	130	Yamaguchi	42	Clerk	6	73212.14	75.6
5	170	Kermisch	15	Clerk	4	75332.79	110.1
6	180	Abrahams	38	Clerk	3	75031.8	236.5
7	190	Sneider	20	Clerk	8	77745.83	126.5
8	200	Scoutten	42	Clerk	(null)	74425.41	84.2
9	230	Lundquist	51	Clerk	3	76677.46	189.65
10	250	Wheeler	51	Clerk	6	77996.6	513.3
11	330	Burke	66	Clerk	1	73795.48	55.5
12	350	Gafney	84	Clerk	5	76266.91	188

PL/SQL – Multiple parameters


Oracle SQL Developer : Procedure DBS311_231ZEE40.JOBRAISE@Winter 2023 ZE
File Edit View Navigate Run Source Team Tools Window Help
...ge Russell~5 Winter 2023 ZE SelectInto.sql SelectIntoException Handling.sql JOBRAISE
Code Grants Dependencies References Details Errors Profiles
Winter 2023 ZE

```
create or replace PROCEDURE JOBRAISE
(
  JOBIN    IN CHAR
, PERCENT IN NUMBER
) AS
BEGIN
  UPDATE STAFF
  SET SALARY = SALARY * (PERCENT/100 + 1)
  WHERE Job = JOBIN;
  DBMS_OUTPUT.PUT_LINE(JOBIN || ' awarded a ' ||
                       TO_CHAR(PERCENT) ||
                       '% salary increase');
END JOBRAISE;
```

Out Paramater used with SQL/PL (DB2)



Summer 22 Workspace - Remote System Explorer - RemoteSystemsTempFiles/ZEUS.SENECACOLLEGE.CA/QSYS.LIB/FC311A05.LIB/MYPRO...

File Edit Source Compile(G) Navigate Search Project Run Window Help
Quick Access
MYPROC2.SQL

```
 Line 1        Column 1       Insert
       ----+----1----+----2----+----3----+----4----+----5----+--
create or replace PROCEDURE MYPROC1
(
   IN   IDIN     SMALLINT,
   IN   NAMEIN   CHAR(9),
   IN   JOBIN    CHAR(5),
   OUT  SUCCESS  CHAR(1)
)
BEGIN
   DECLARE SQLSTATE CHAR(5) DEFAULT ' ';
   DECLARE SQLCODE  INTEGER DEFAULT 0;
   INSERT INTO DBS311/STAFFZ ("ID", NAME, DEPT, JOB)
                    VALUES(IDIN, NAMEIN,99,JOBIN);
  IF SQLCODE = 0 THEN
     SET SUCCESS = '1';
  ELSE
     SET SUCCESS = '0';
  END IF;
END
```

Out Paramater used with SQL/PL (DB2)

- SELECT * FROM STAFFZ WHERE NAME = 'Pangborn’

```
    ID    NAME           DEPT    JOB      YEARS       SALARY          COMM
 *******   End of data   ********
```

- CREATE VARIABLE DBS311.SUCCESS CHAR(1) DEFAULT NULL
- SELECT DBS311.SUCCESS FROM SYSIBM.SYSDUMMY1

```
SUCCESS
   -
********   End of data
```

- Set up debug mode for procedure
- call myproc1(444,'Pangborn','Prof',DBS311.SUCCESS)

Out Paramater used with SQL/PL (DB2)

- Stepping through procedure

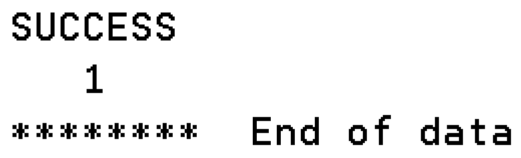
```
Program:    MYPROC1         Library:    FC311A05        Module:    MYPROC1
       1   CREATE OR REPLACE PROCEDURE MYPROC1 ( IN IDIN SMALLINT , IN NAMEIN CHA
       2   BEGIN
       3   DECLARE SQLSTATE CHAR ( 5 ) DEFAULT ' ';
       4   DECLARE SQLCODE INTEGER DEFAULT 0;
       5   INSERT INTO DBS311 / STAFFZ ( "ID" , NAME , DEPT , JOB ) VALUES ( IDIN
       6   IF SQLCODE = 0
       7   THEN
       8   SET SUCCESS = '1';
       9   ELSE
      10   SET SUCCESS = '0';
      11   END IF;
```

- After procedure finishes

```
....+....1....+....2....+....3....+....4....+....5....+....6....+..
    ID    NAME          DEPT    JOB      YEARS       SALARY        COMM
   444    Pangborn        99    Prof         -            -           -
********  End of data  ********
```

Out Paramater used with SQL/PL (DB2)

- SELECT DBS311.SUCCESS FROM SYSIBM.SYSDUMMY1



```
SUCCESS
   1
********  End of data
```

PL/SQL Anonymous Blocks

- A block without a name is an anonymous block.
- An anonymous block is not saved in the Oracle Database server, so it is just for one-time use. However, PL/SQL anonymous blocks can be useful for testing purposes.
- You can save the SQL code as a file on your local machine.

PL/SQL Anonymous Blocks

- BEGIN
- DBMS_OUTPUT.PUT_LINE (‘Welcome to DBS311!’);
- END;
- To see the output:
- Execute the following statement first:
- SET SERVEROUTPUT ON;
- To execute the block, in your SQL developer worksheet, put your cursor on any line of the block and press Ctrl + Enter
- Output:
- Welcome to DBS311!
- PL/SQL procedure successfully completed.

Procedure vs Anonymous Block

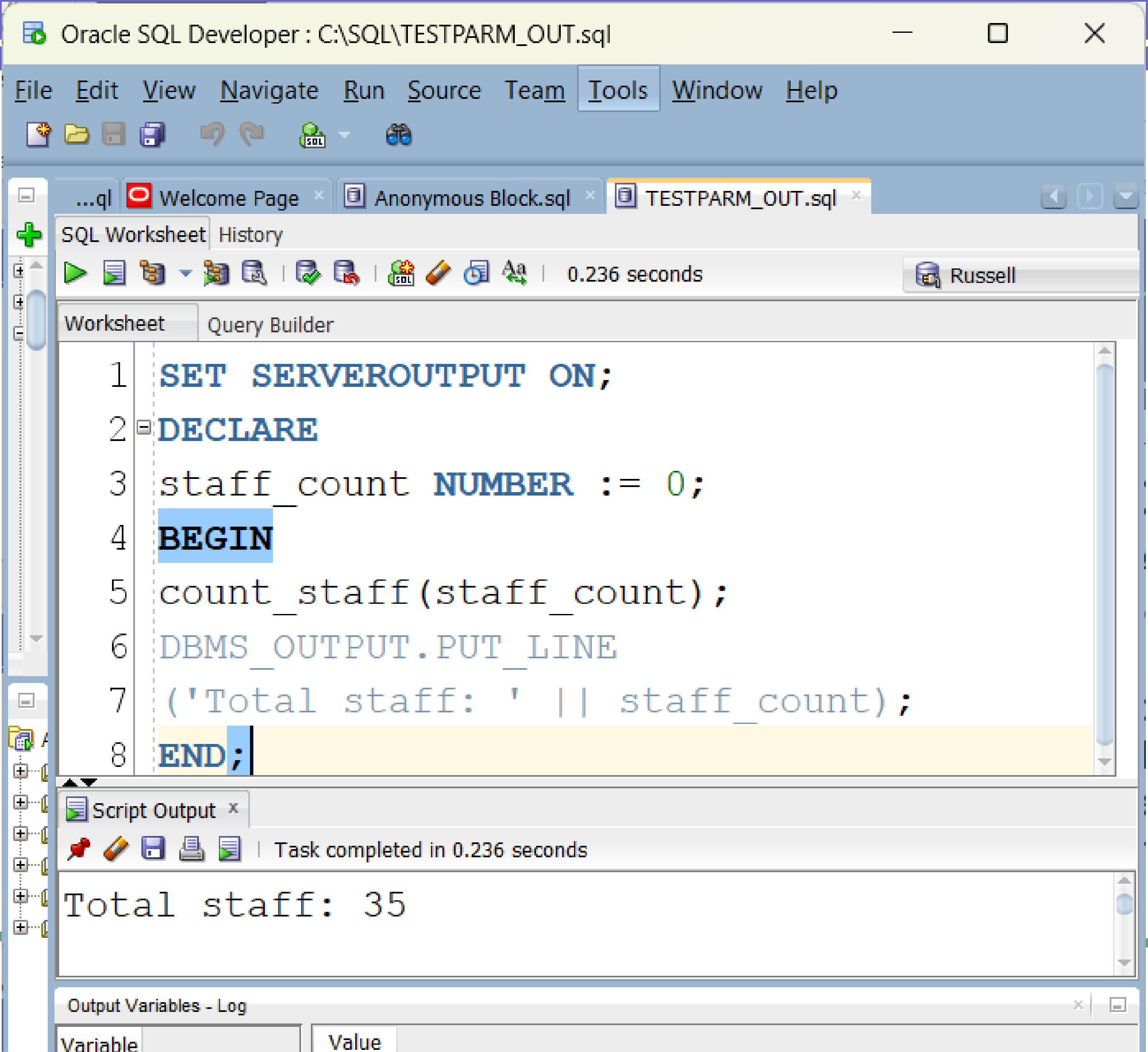
- CREATE OR REPLACE PROCEDURE NOTANONYMOUS
- AS
- BEGIN
-    DBMS_OUTPUT.PUT_LINE('WELCOME TO DBS311');
- END;
- This is a permanent database object on the server, not a file on your PC
- EXECUTE NOTANONYMOUS;
- Welcome to DBS311
- To remove the database object
- DROP PROCEDURE NOTANONYMOUS;

Out Paramater used with PL/SQL (Oracle)

- Will test procedure (saved object) with anonymous block (saved file)
- Procedure (saved object)
- create or replace PROCEDURE count_staff (staff_count OUT NUMBER) AS

```
BEGIN
      select count(*) INTO staff_count
      from staff;
END;
```

Out Paramater used with PL/SQL (Oracle)



```
SET SERVEROUTPUT ON;
DECLARE
staff_count NUMBER := 0;
BEGIN
count_staff(staff_count);
DBMS_OUTPUT.PUT_LINE
('Total staff: ' || staff_count);
END;
```

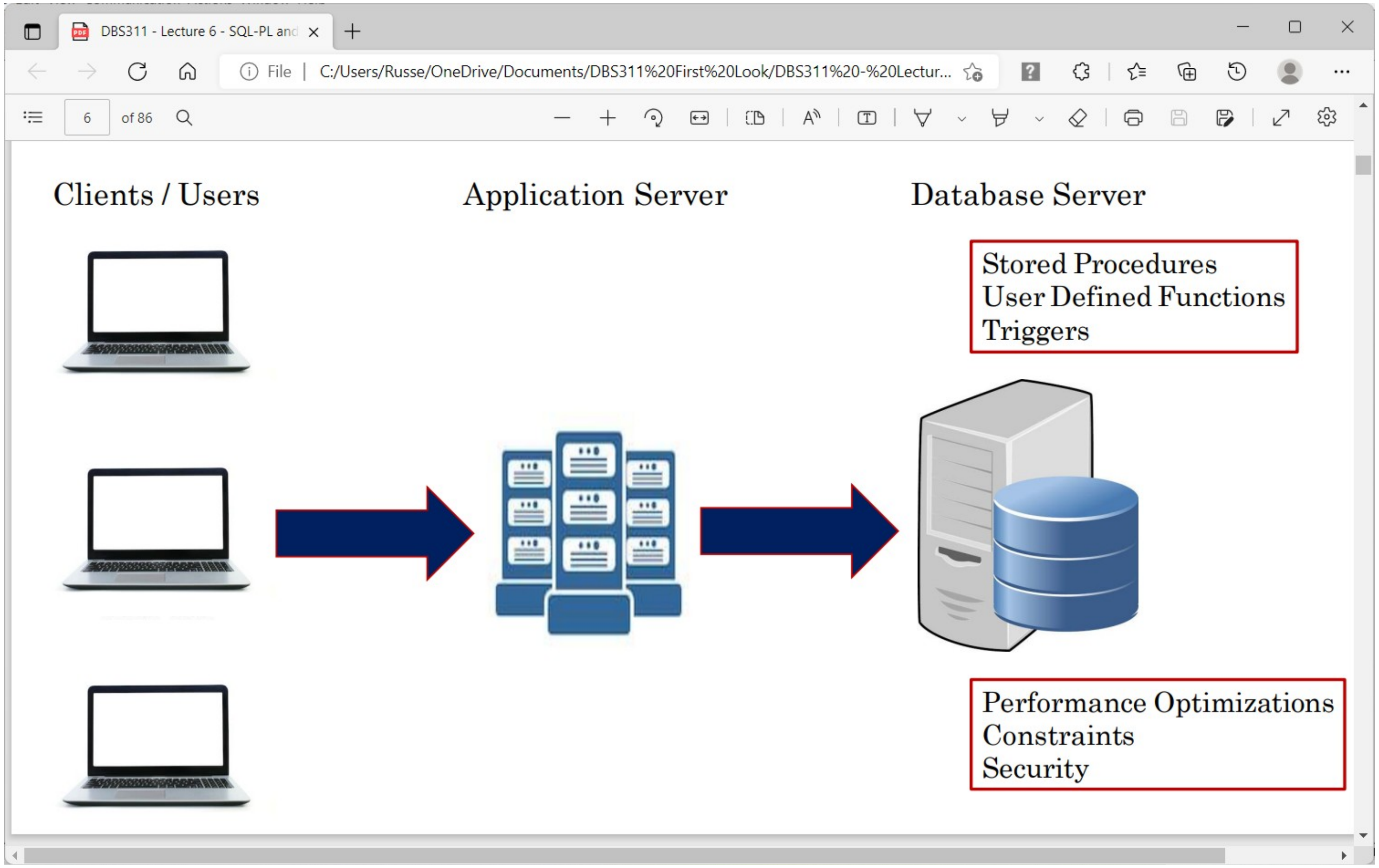
```
Total staff: 35
```

IN OUT Parameters

- CREATE OR REPLACE PROCEDURE procedure_name(arg1 IN OUT data_type, ...) AS
- BEGIN ....
- END procedure_name;
- The following procedure gets a salary and increases the salary by 20%

```
CREATE OR REPLACE PROCEDURE new_salary
        (salary IN OUT FLOAT)
AS
BEGIN
                salary := salary * 1.20;
END;
-- Anonymous Block to test
DECLARE salary_A FLOAT := 10000;
BEGIN
   new_salary (salary_A);
  DBMS_OUTPUT.PUT_LINE (salary_A);
END;
RESULTS IN              12000
```

Anonymous Blocks Stored on Client side


DBS311 - Lecture 6 - SQL-PL and
File | C:/Users/Russe/OneDrive/Documents/DBS311%20First%20Look/DBS311%20-%20Lectur...
6 of 86
Clients / Users
Application Server
Database Server
Stored Procedures
User Defined Functions
Triggers
Performance Optimizations
Constraints
Security

SQL/PL (DB2)

- Using Debug to investigate IF ELSE ELSEIF statements

```
CREATE OR REPLACE PROCEDURE PROCIFELSE
BEGIN
  DECLARE AGE INT;
  SET AGE = 35;
  IF AGE = 35
     THEN SET AGE = 45;
  END IF;
  IF AGE = 45
    THEN SET AGE = 55
  ELSE
    SET AGE = 65;
  END IF;
                      IF AGE = 55
                        THEN SET AGE = 65;
                      ELSEIF AGE = 65
                        THEN SET AGE = 45;
                      ELSE
                        SET AGE = 35;
                      END IF;
                      INSERT INTO FORTEST1/TEST1  VALUES(AGE,
                                                         CURRENT TIMESTAMP,
                                                         CURRENT USER);
                      END
```

- What is age?
- Look at STRDBG

SQL/PL (DB2)

```
SELECT * FROM FORTEST1.TEST1

          AGE   TIMEENTERED                        PROFILE
 ********  End of data  ********

 CALL PROCIFELSE
 CALL statement complete.

SELECT * FROM FORTEST1.TEST1


            AGE   TIMEENTERED                 PROFILE
             65   2023-02-13-10.50.36.896051  FC311A05
   ********  End of data  ********
```

```
CREATE OR REPLACE PROCEDURE PROCCASE
BEGIN
DECLARE AGE1 INT;
DECLARE AGE2 INT;
DECLARE AGE3 INT;
SET AGE1 = 45;
```

```
CASE AGE1
  WHEN 35
    THEN SET AGE2 = 35;
  WHEN 45
    THEN SET AGE2 = 45;
  WHEN 55
    THEN SET AGE3 = 55;
  ELSE
    SET AGE2 = 0;
END CASE;
```

```
SET AGE1 = 35;
SET AGE3 = 35;
CASE
  WHEN AGE1 > 35
    THEN SET AGE2 = 35;
  WHEN AGE1 > 45
    THEN SET AGE2 = 45;
  WHEN AGE1 > 55
    THEN SET AGE2 = 55;
  ELSE
    SET AGE2 = 0;
  END CASE;
```

Case continued

```
 CASE
   WHEN AGE1 BETWEEN 35 AND 55
     THEN SET AGE2 = 35;
   WHEN AGE1 BETWEEN 45 AND 65
     THEN SET AGE2 = 45;
   WHEN AGE1 BETWEEN 75 AND 100
     THEN SET AGE2 = 55;
   ELSE
     SET AGE2 = 0;
   END CASE;
   INSERT INTO FORTEST1/TEST1  VALUES(AGE2,
                              CURRENT TIMESTAMP,
                              CURRENT USER);
END
```

SQL/PL (DB2)

```
  DELETE FROM FORTEST1.TEST1

  CALL PROCCASE
  CALL statement complete.

SELECT * FROM FORTEST1.TEST1




            AGE   TIMEENTERED                 PROFILE
             35   2023-02-13-10.53.25.727554  FC311A05
  ********  End of data  ********
```

SQL/PL (DB2)

```
CREATE OR REPLACE PROCEDURE UPDATE_SAL (IN EMPNUM SMALLINT,
                                        IN RATING INTEGER)
LANGUAGE SQL
BEGIN
  IF RATING = 1 THEN
      UPDATE DBS311J.STAFF2
         SET SALARY = SALARY *1.10, COMM = COMM + 1000
             WHERE ID = EMPNUM;
  ELSEIF RATING = 2 THEN
      UPDATE DBS311J.STAFF2
         SET SALARY = SALARY *1.05, COMM = COMM + 500
             WHERE ID = EMPNUM;
  ELSE
      UPDATE DBS311J.STAFF2
         SET SALARY = SALARY *1.03
             WHERE ID = EMPNUM;
  END IF;
 END;
```

```
ID   NAME        DEPT   JOB      YEARS      SALARY       COMM
10   Sanders       20   Mgr          7   68,357.50          -
20   Pernal        20   Sales        8   68,171.25     612.45
30   Marenghi      38   Mgr          5   67,506.75          -
40   O'Brien       38   Sales        6   68,006.00     846.55
50   Hanes         15   Mgr         10   70,659.80          -
60   Quigley       38   Sales         -  66,808.30     650.25
70   Rothman       15   Sales        7   66,502.83   1,152.00
80   James         20   Clerk         -  63,504.60     128.20
```

```
CALL UPDATE_SAL(70,1)
```

```
ID   NAME        DEPT   JOB      YEARS      SALARY       COMM
10   Sanders       20   Mgr          7   68,357.50          -
20   Pernal        20   Sales        8   68,171.25     612.45
30   Marenghi      38   Mgr          5   67,506.75          -
40   O'Brien       38   Sales        6   68,006.00     846.55
50   Hanes         15   Mgr         10   70,659.80          -
60   Quigley       38   Sales         -  66,808.30     650.25
70   Rothman       15   Sales        7   73,153.11   2,152.00
80   James         20   Clerk         -  63,504.60     128.20
```

PL/SQL (Oracle) Exception

- Exception part handles the errors occurred during the execution of a PL/SQL block.
- DECLARE
- value_1 NUMBER := 20;
- value_2 NUMBER := 0;
- division NUMBER
- BEGIN
-   division := value_1 / value_2
- DBMS_OUTPUT.PUT_LINE ('division: ' || division);
- The execution of the above code stops with an error. See the following output:
- Error report:
- ORA-01476: divisor is equal to zero ORA-06512: at line 8
- 01476. 00000 - "divisor is equal to zero"

Exception Handling

- To handle the exception errors, we use EXCEPTION section. WE add this section to handle the error in the code from previous slide:

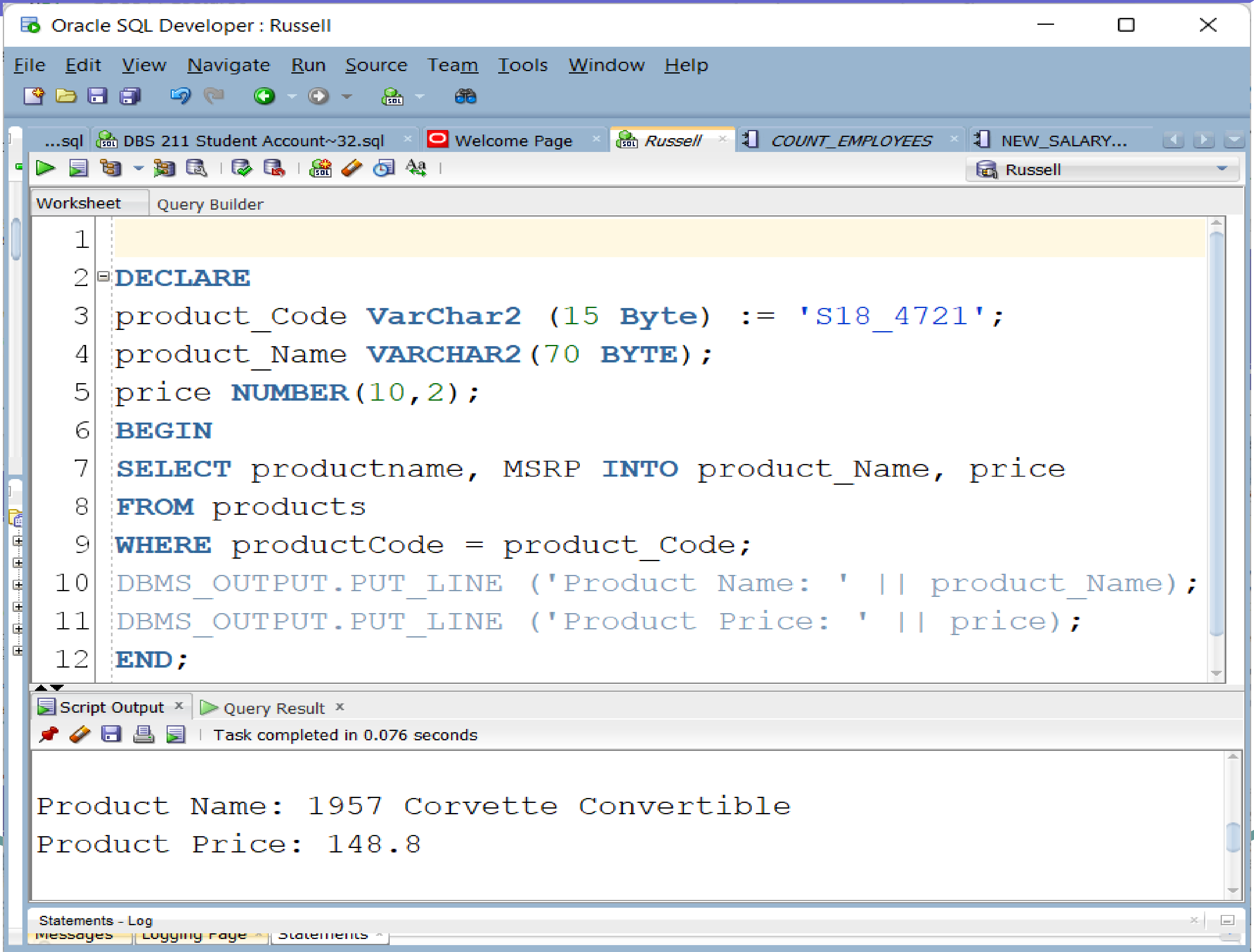
- EXCEPTION
- WHEN OTHERS
- THEN
- DBMS_OUTPUT.PUT_LINE ('Error!');

SELECT INTO

- IN PL/SQL, you can use SELECT INTO statement to store data from a single row fetch by a SELECT statement.
- SELECT column_list
- INTO   variable_list
- FROM   table_name
- WHERE condition(s);

- TOO_MANY_ROWS exception
- An exception will occur if the SELECT statement returns more than one row.
- NO_DATA_FOUND exception
- An exception will occur if the SELECT statement returns no data.

SELECT INTO (Example)



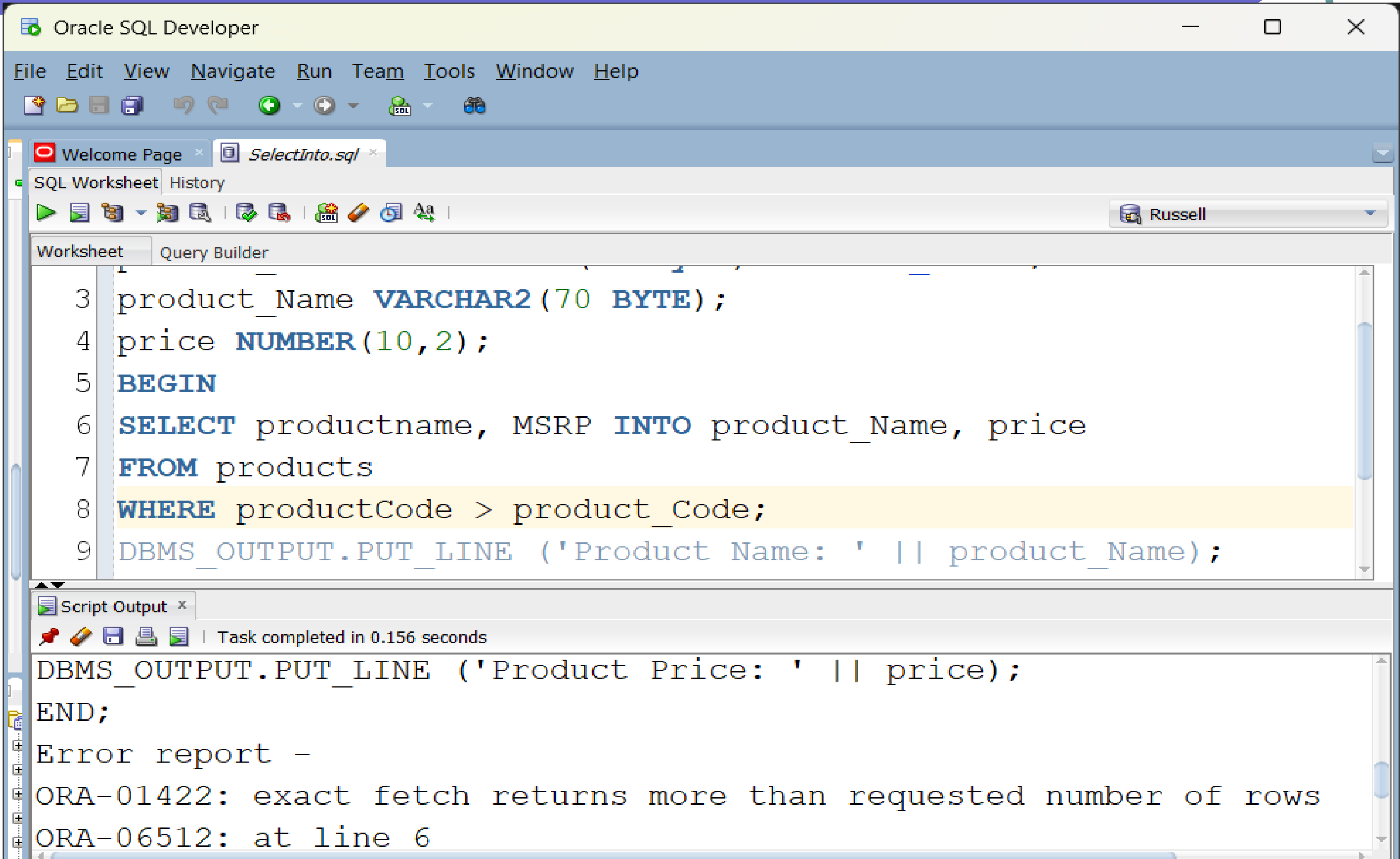
```

DECLARE
product_Code VarChar2 (15 Byte) := 'S18_4721';
product_Name VARCHAR2(70 BYTE);
price NUMBER(10,2);
BEGIN
SELECT productname, MSRP INTO product_Name, price
FROM products
WHERE productCode = product_Code;
DBMS_OUTPUT.PUT_LINE ('Product Name: ' || product_Name);
DBMS_OUTPUT.PUT_LINE ('Product Price: ' || price);
END;
```

```
Product Name: 1957 Corvette Convertible
Product Price: 148.8
```

TOO_MANY_ROWS Exception

- In SELECT INTO statement, an exception occurs if the result of the fetched data includes more than one row.
- We change the condition to
- WHERE productCode > product_Code;
- Since, we have many rows greater than the inputted product code, the SELECT INTO statement fetches more than one row and raised an exception.

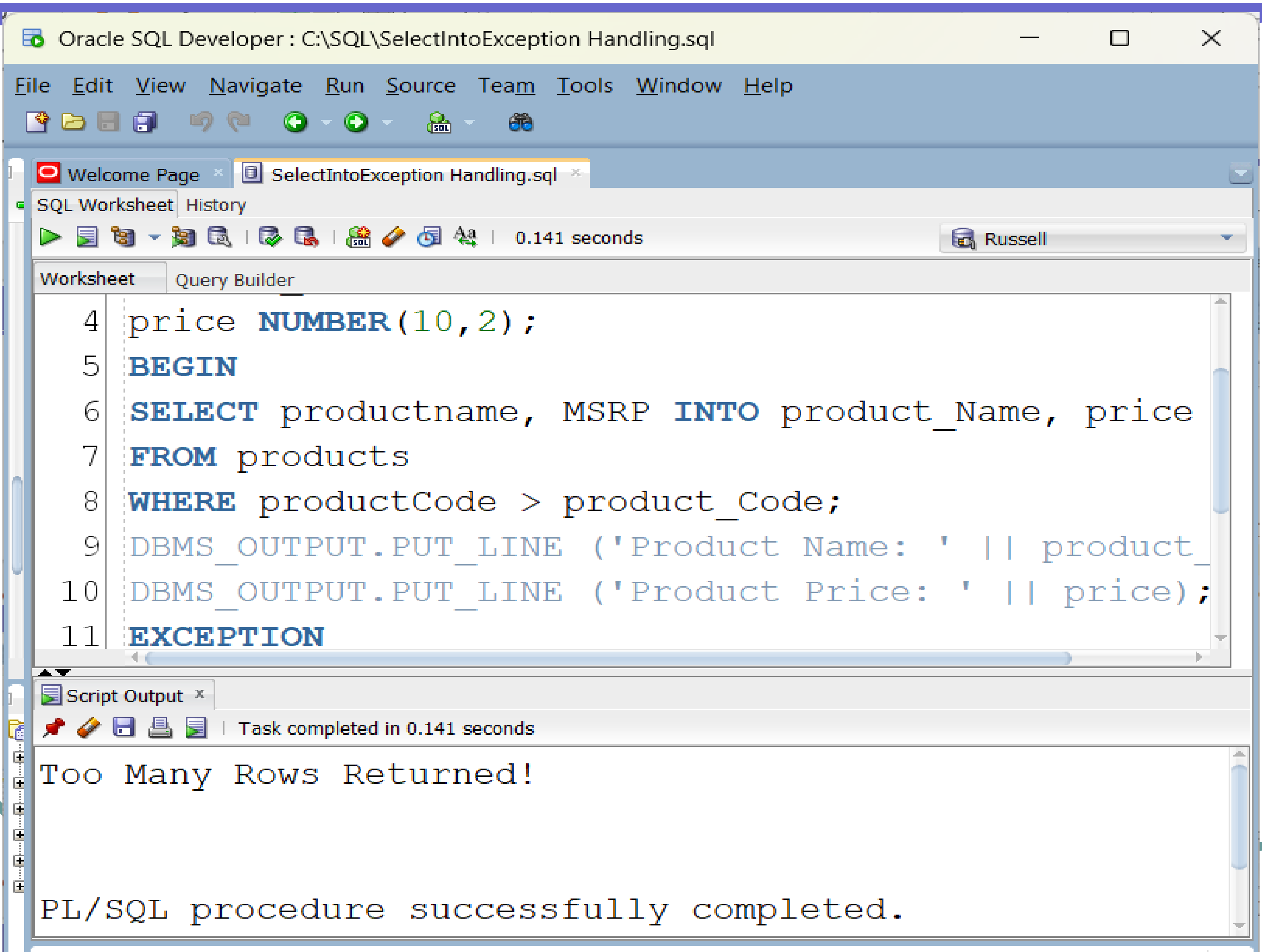
TOO_MANY_ROWS Exception



```
product_Name VARCHAR2(70 BYTE);
price NUMBER(10,2);
BEGIN
SELECT productname, MSRP INTO product_Name, price
FROM products
WHERE productCode > product_Code;
DBMS_OUTPUT.PUT_LINE ('Product Name: ' || product_Name);
```

```
DBMS_OUTPUT.PUT_LINE ('Product Price: ' || price);
END;
Error report -
ORA-01422: exact fetch returns more than requested number of rows
ORA-06512: at line 6
```

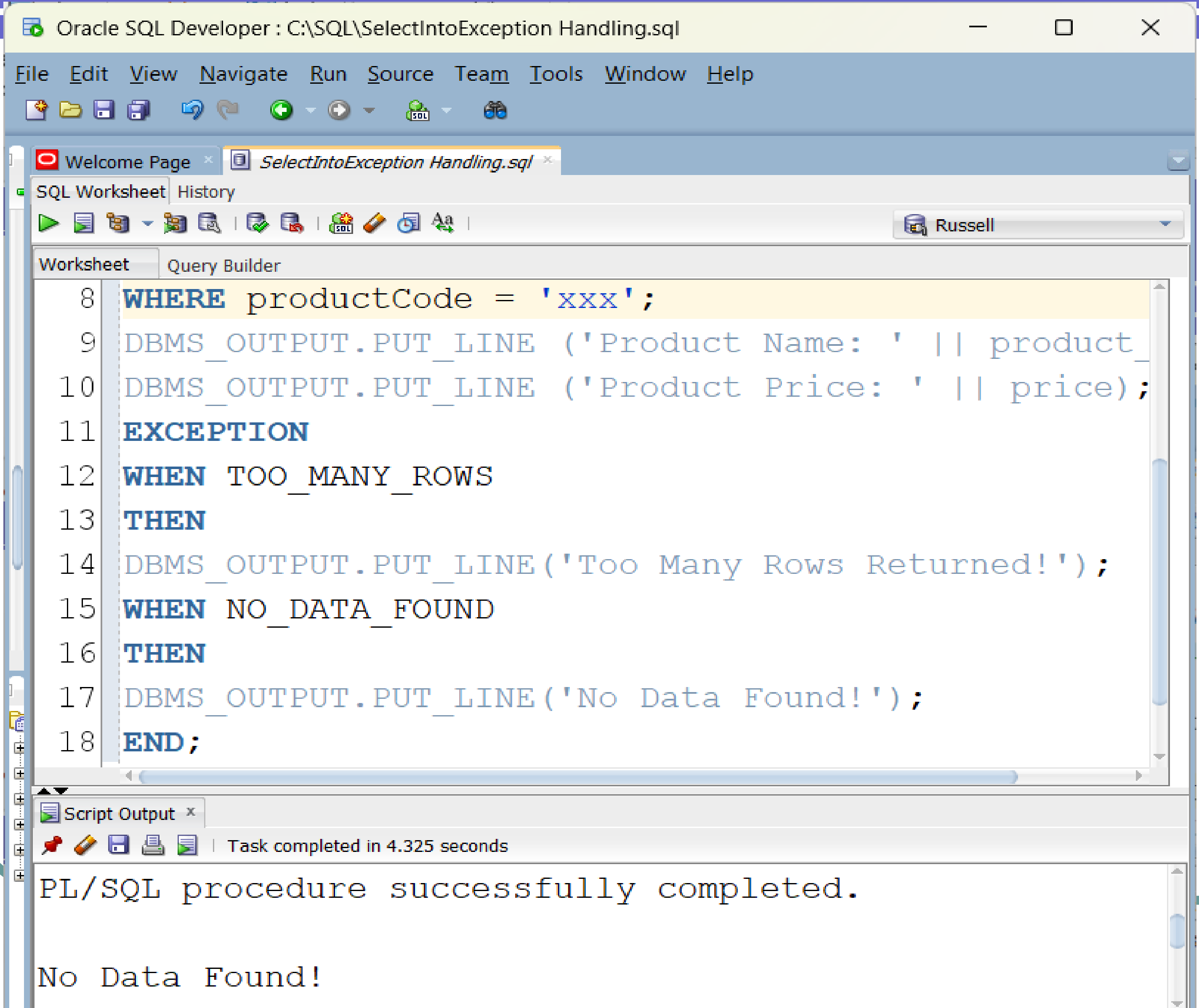
TOO_MANY_ROWS Exception


Oracle SQL Developer : C:\SQL\SelectIntoException Handling.sql
File Edit View Navigate Run Source Team Tools Window Help
Welcome Page
SelectIntoException Handling.sql
SQL Worksheet
History
0.141 seconds
Russell
Worksheet
Query Builder
Script Output
Task completed in 0.141 seconds

```
price NUMBER(10,2);
BEGIN
SELECT productname, MSRP INTO product_Name, price
FROM products
WHERE productCode > product_Code;
DBMS_OUTPUT.PUT_LINE ('Product Name: ' || product_
DBMS_OUTPUT.PUT_LINE ('Product Price: ' || price);
EXCEPTION
```

```
Too Many Rows Returned!


PL/SQL procedure successfully completed.
```

NO_DATA_FOUND Exception


Oracle SQL Developer : C:\SQL\SelectIntoException Handling.sql
File Edit View Navigate Run Source Team Tools Window Help
Welcome Page
SelectIntoException Handling.sql
SQL Worksheet History
Russell
Worksheet Query Builder
Script Output
Task completed in 4.325 seconds

```
WHERE productCode = 'xxx';
DBMS_OUTPUT.PUT_LINE ('Product Name: ' || product_
DBMS_OUTPUT.PUT_LINE ('Product Price: ' || price);
EXCEPTION
WHEN TOO_MANY_ROWS
THEN
DBMS_OUTPUT.PUT_LINE('Too Many Rows Returned!');
WHEN NO_DATA_FOUND
THEN
DBMS_OUTPUT.PUT_LINE('No Data Found!');
END;
```

```
PL/SQL procedure successfully completed.

No Data Found!
```

Lab 5 & Assignment 2

- Lab 5 will be an Oracle lab on Procedures
- Lab 5 Due after break week in the Lab period
- Assignment 2 released this week
- We are one week ahead of schedule
- Two work periods to work on first two A2 questions

Date	Lecture Period	Lab Period
10/6	Stored Procedures (Lab 5)	Midterm Test
10/13	Iteration and Triggers (Lab 6)	Work Period – no class
Break Week		
10/27	Work Period – no class	Lab 5 Demonstration
11/3	Cursors	Lab 6 Demonstration
11/10	On Schedule – MongoDB	Lab 7 Demonstration