**1. The Raise Procedure**

Create a copy of the EMPLOYEES table by running the following code:

CREATE TABLE MyEmployees AS SELECT \* FROM HR.Employees;

Now, create a new table that you will use for logging.

CREATE TABLE HW1Log

(

Message VARCHAR2(200),

TStamp TIMESTAMP DEFAULT SYSTIMESTAMP

);

Notice that the TStamp column was created with a default value. You can add a log message to the table like this:

INSERT INTO HW1Log (Message) VALUES ('Some message');

If you look at the table now, you’ll see that the timestamp column has been populated automatically. Later on, you’ll be asked to use code like this to write messages to your log table.

Now, create a stored procedure called RaiseSalary that accepts two parameters – an employee id and an integer raise percentage. The procedure will raise the salary of the specified employee by the specified percentage. It will do this by updating the Salary column in MyEmployees. Calling the procedure with an argument of 101 for the employee id and 20 for the raise percentage would raise that employee’s salary by 20 percent.

Your procedure should include exception handling. It should have a generic error handler which prints out both the SQLERRM and the phrase “*Encountered while trying to give employee <parameter value> a raise*.” Of course, instead of <parameter value>, your error message should include the value of the employee id parameter that was passed into the procedure.

Every time your procedure raises the salary of an employee, you should write a message to your HW1Log table in the following form:

*Employee <employee id> was raised by <raise percentage> percent.*

Make sure that you use %TYPE where applicable. Also, do a “COMMIT;” inside your procedure (after all the other code in the BEGIN section but before EXCEPTION) to make your changes public and permanent. It’s not always a good idea to have a COMMIT inside your procedures, but for the sake of this exercise, please put one in there.

**PROCEDURE**

SET SERVEROUTPUT ON

CREATE OR REPLACE PROCEDURE raiseEmpSalary (

p\_emp\_id MyEmployees.employee\_id%TYPE,

p\_sal\_raise NUMBER

) AS

v\_emp\_update\_count INTEGER:=0;

BEGIN

UPDATE MyEmployees set salary = (salary\*p\_sal\_raise/100)+salary where employee\_id = p\_emp\_id;

v\_emp\_update\_count := SQL%ROWCOUNT;

IF v\_emp\_update\_count = 0 THEN

RAISE NO\_DATA\_FOUND;

ELSE

INSERT INTO HW1Log (message) VALUES ('Employee ' || p\_emp\_id || ' was raised by ' || p\_sal\_raise || ' percent.');

END IF;

COMMIT;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE(SQLERRM);

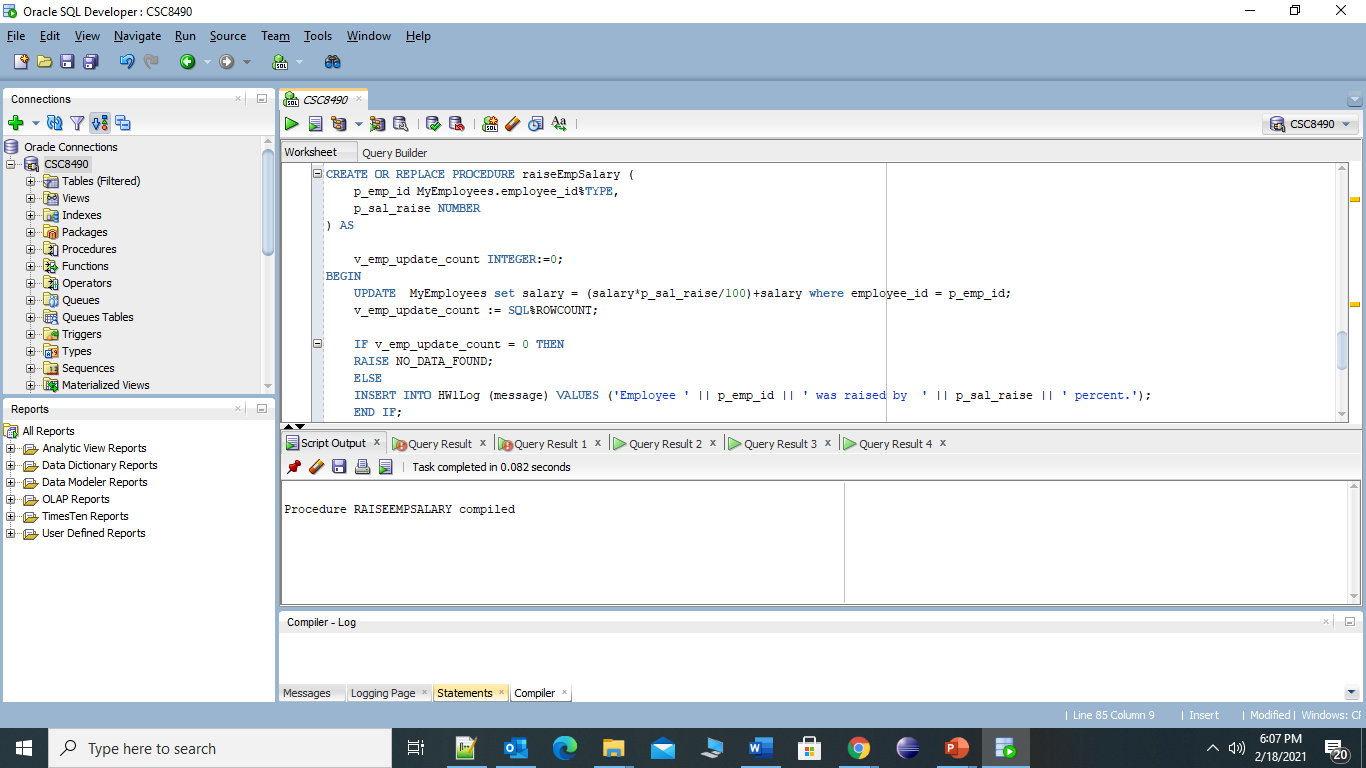
DBMS\_OUTPUT.PUT\_LINE('Encountered while trying to give employee '|| p\_emp\_id ||' a raise.');

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Some other error.');

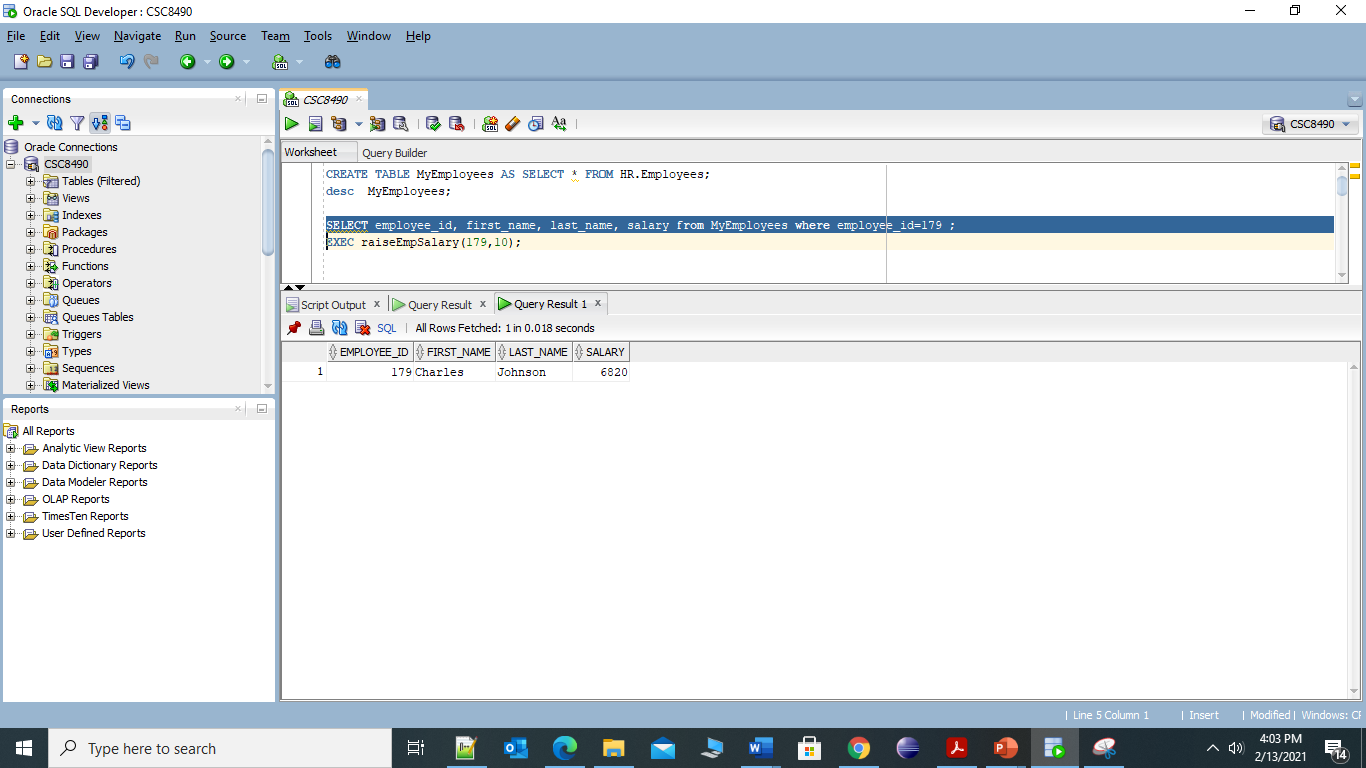
END raiseEmpSalary;

Reference: <https://docs.oracle.com/cd/B19306_01/appdev.102/b14261/sqloperations.htm>



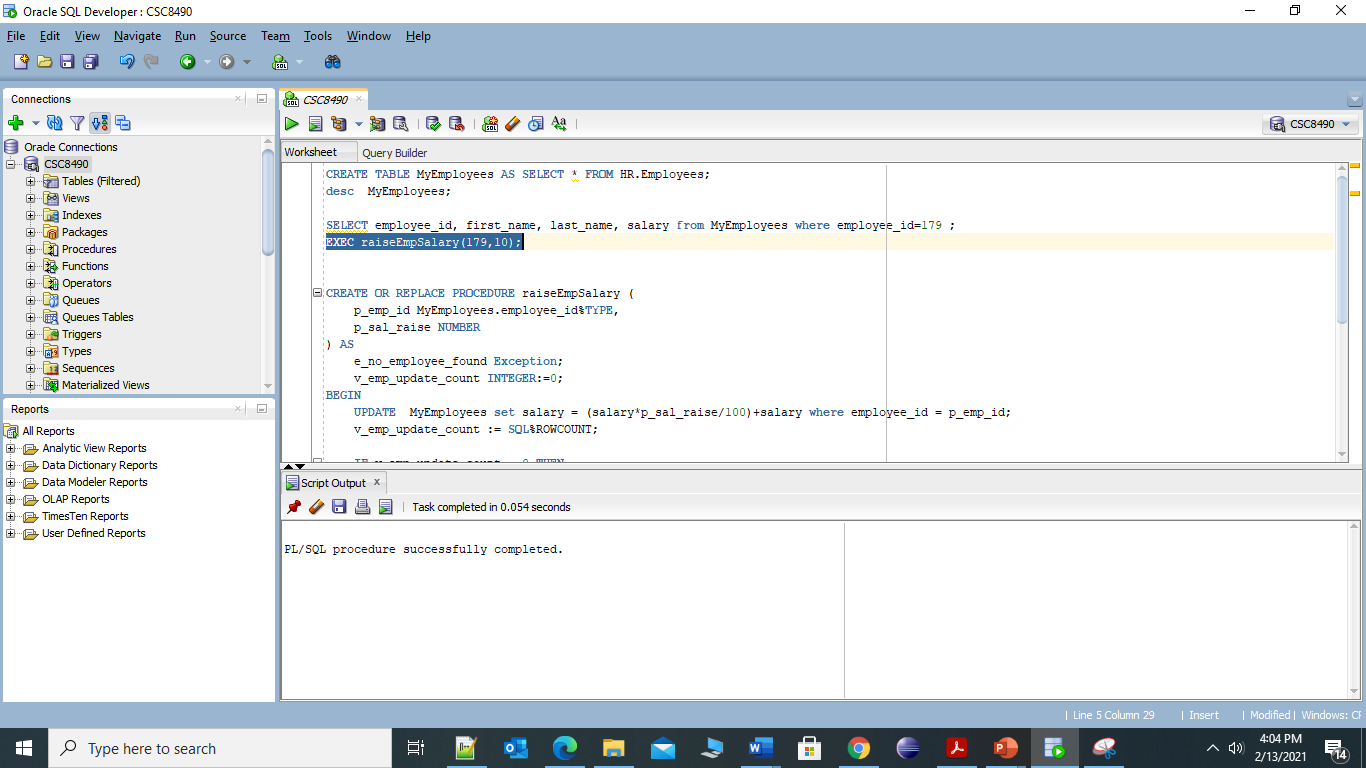
* Show the initial salary of employee 179 (can do this with a standard SQL query)

SELECT employee\_id, first\_name, last\_name, salary from MyEmployees where employee\_id=179 ;



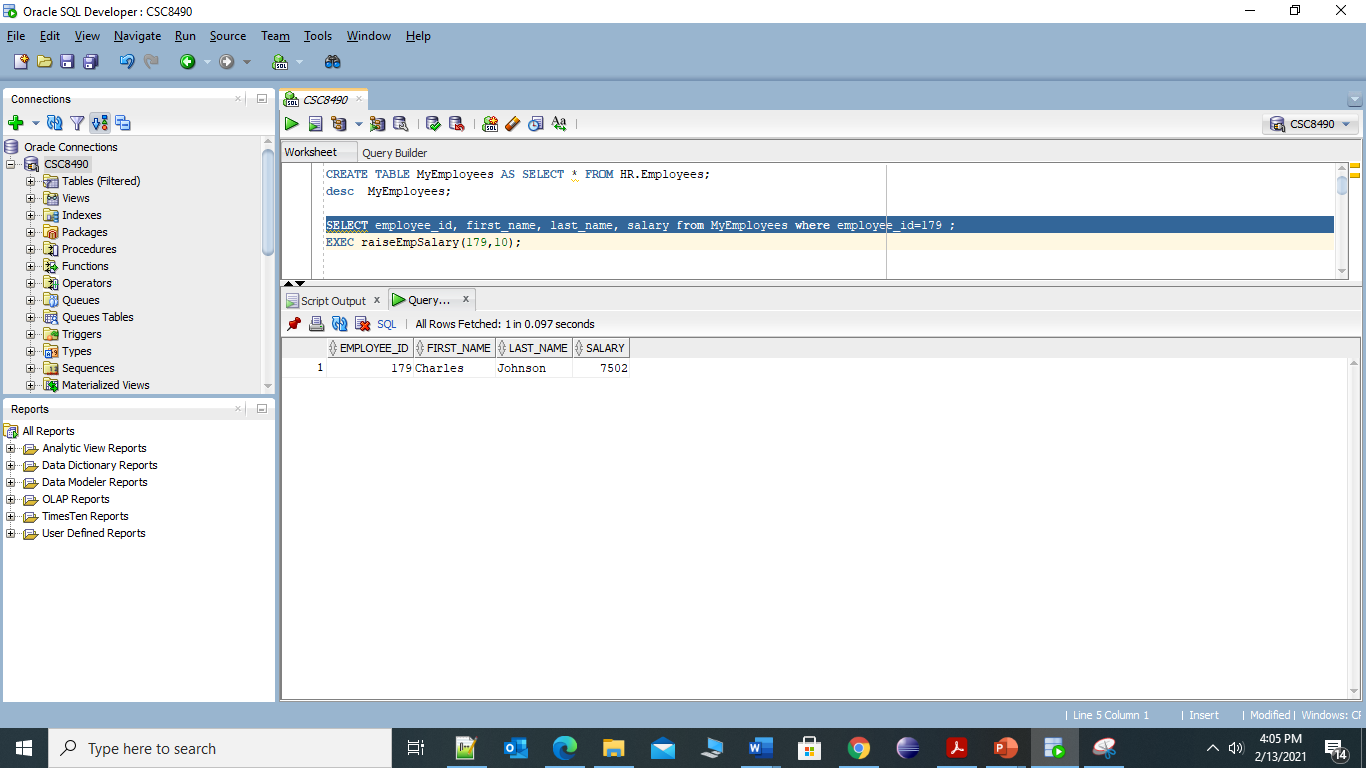
* Call your procedure with an employee id of 179 and a percentage of 10

EXEC raiseEmpSalary(179,10);



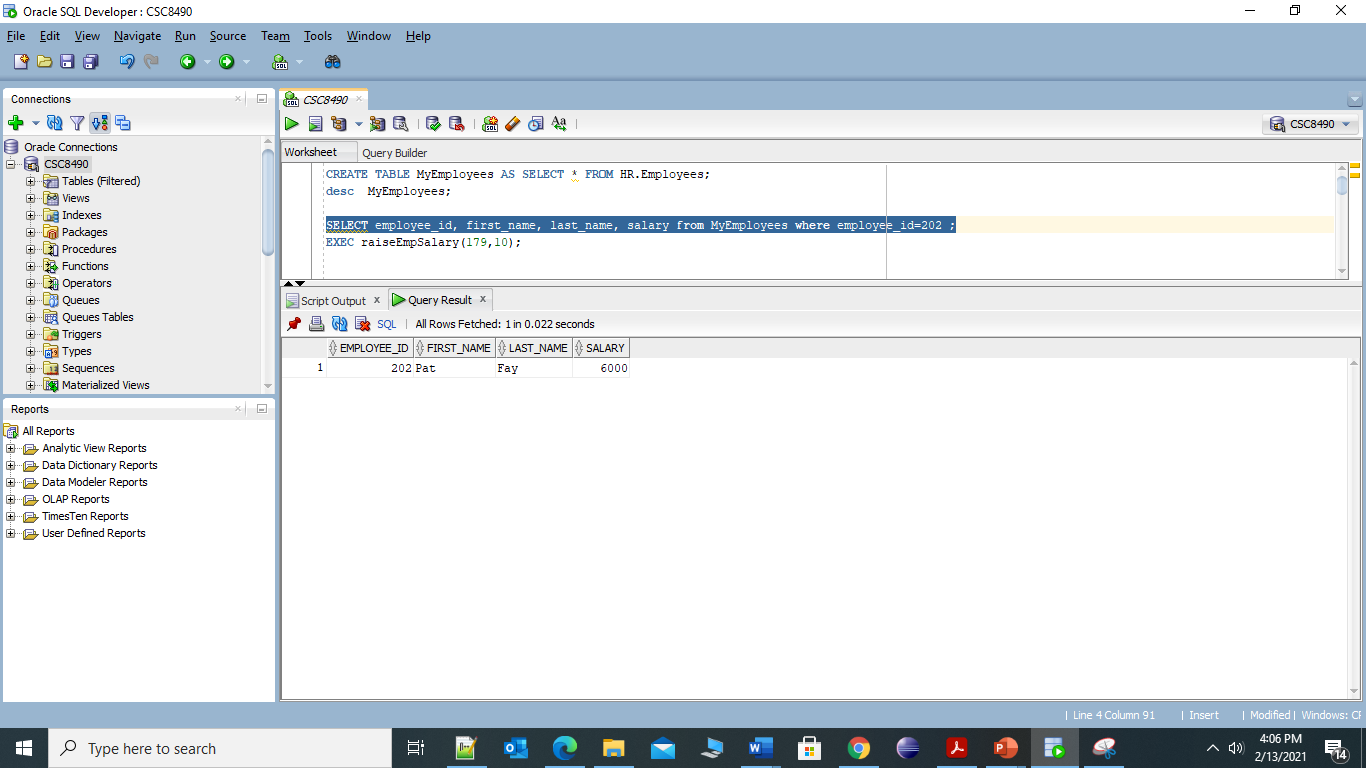
* Show the salary of employee 179 after the procedure call (can do this with a standard SQL query)

SELECT employee\_id, first\_name, last\_name, salary from MyEmployees where employee\_id=179 ;



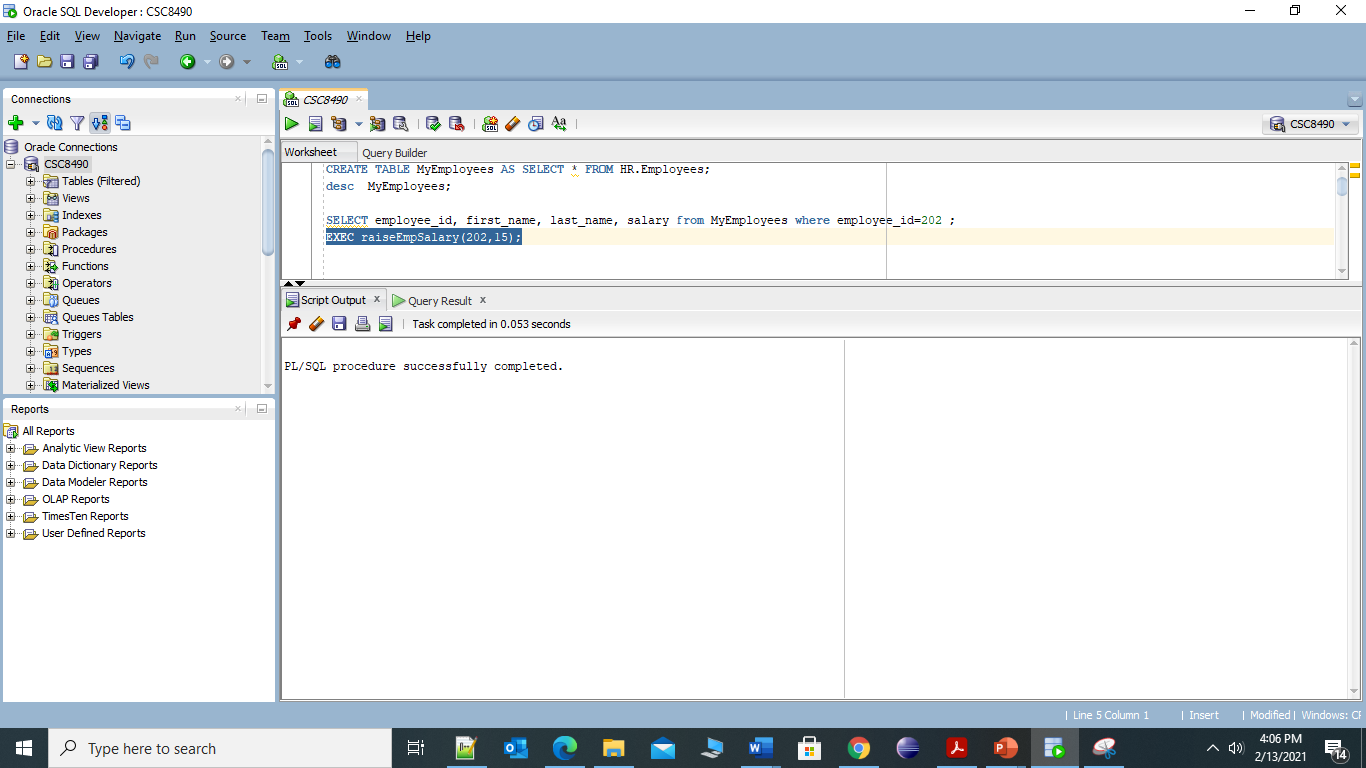
* Show the initial salary of employee 202 (can do this with a standard SQL query)

SELECT employee\_id, first\_name, last\_name, salary from MyEmployees where employee\_id=202 ;



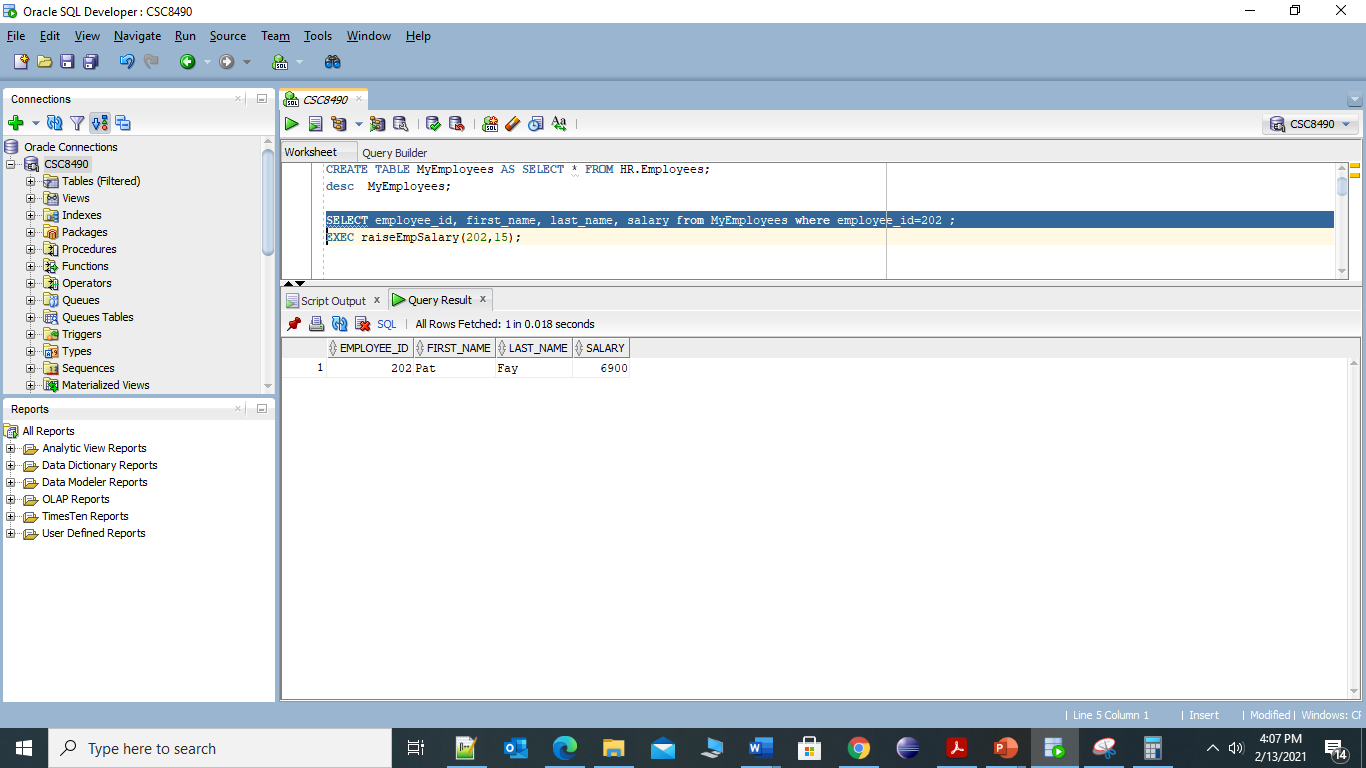
* Call your procedure with an employee id of 202 and a percentage of 15

EXEC raiseEmpSalary(202,15);



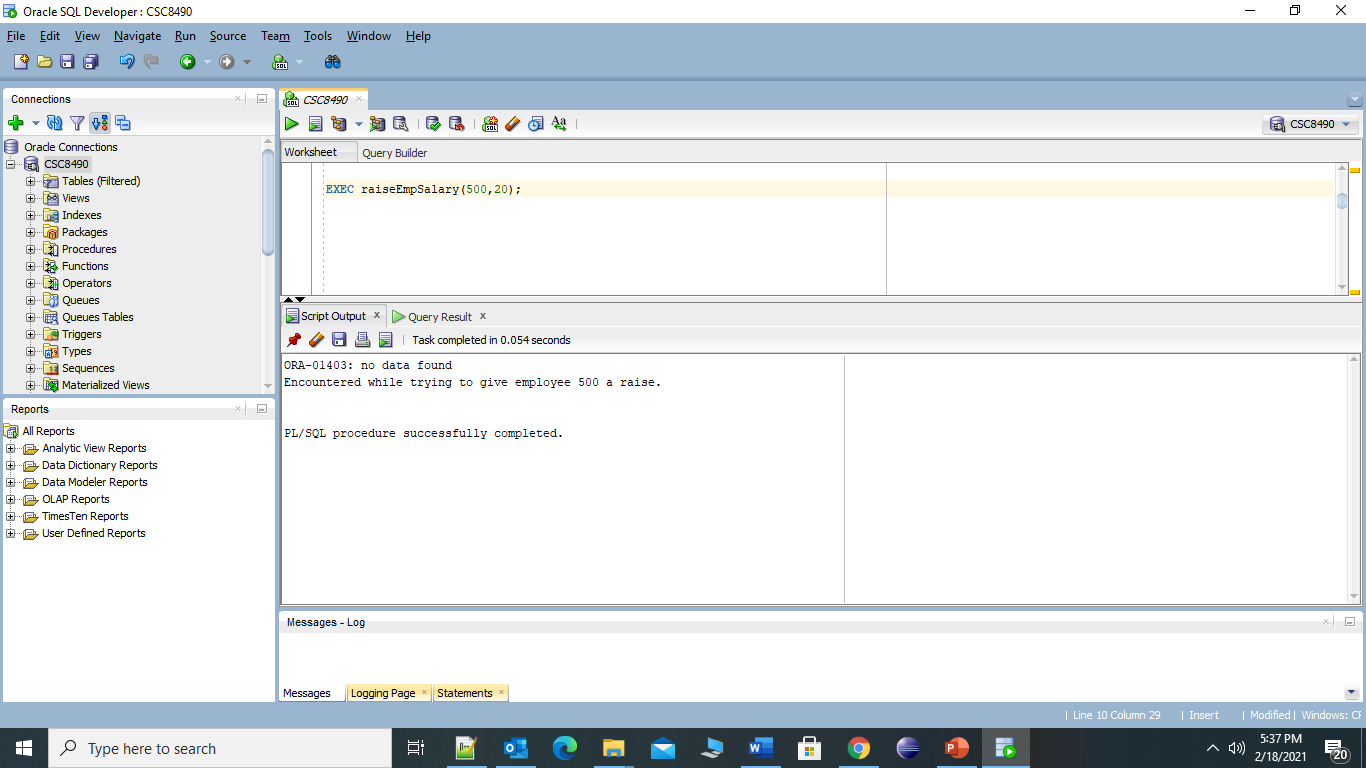
* Show the salary of employee 202 after the procedure call (can do this with a standard SQL query)

SELECT employee\_id, first\_name, last\_name, salary from MyEmployees where employee\_id=202 ;



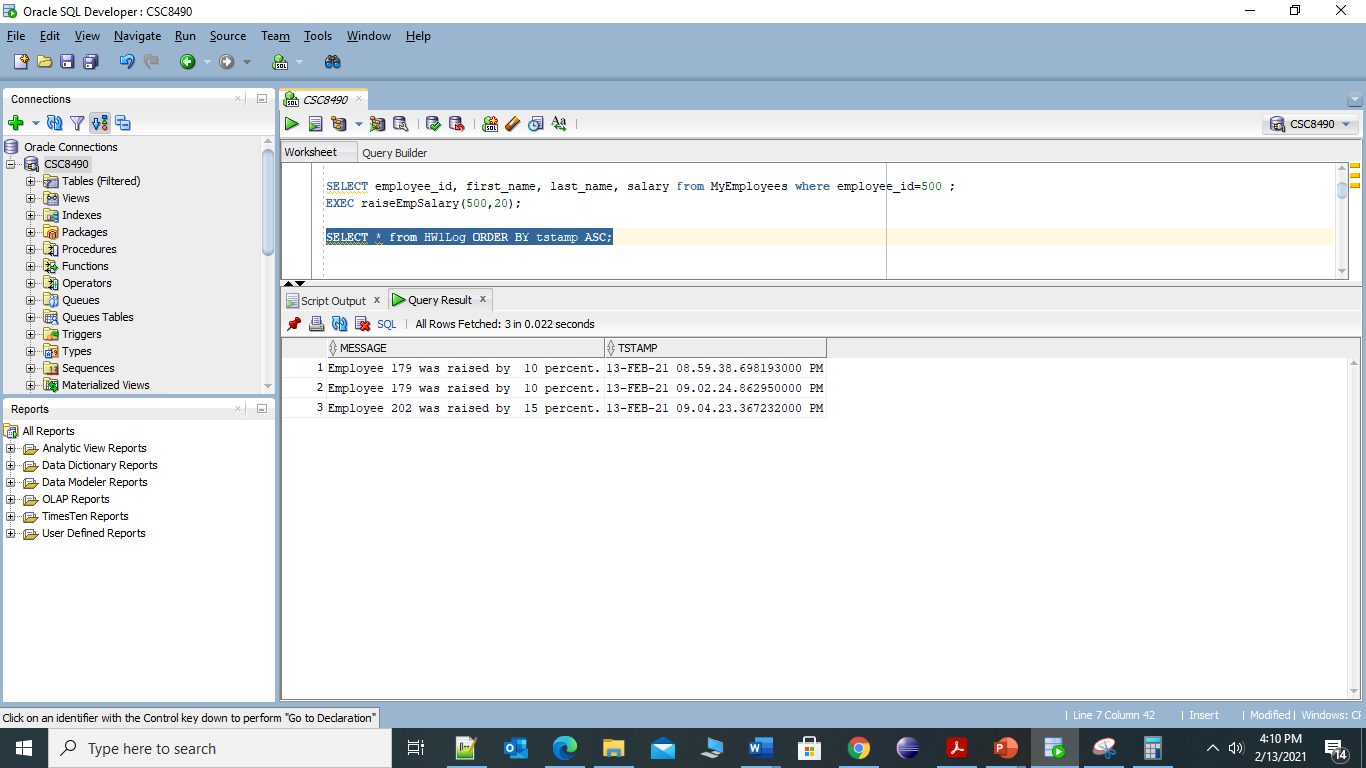
* Call your procedure with an employee id of 500 and a percentage of 20

EXEC raiseEmpSalary(500,20);



* Show all of the data from your HW1Log table (can do this with a standard SQL query) – order your results by ascending TStamp value.

SELECT \* from HW1Log ORDER BY tstamp ASC;



**2. But Money Doesn’t Grow on Trees**

The company had a good year, and so we want to give out some raises. We generally think that a good raise is 5% of a person’s salary. However, we don’t have an unlimited amount of money. There are only $25,000 dollars available in our raise budget, and that isn’t enough to give everybody a 5% raise, so we need to make some choices.

Management has decided to give people 5% raises until the money runs out. Since there isn’t enough money available for everybody, we need a way to prioritize who gets the raises. Management determines that raises should be prioritized in salary order – the employees with the lowest salaries should get raises first. If two people have the same salary, they should be prioritized by their hire date, with employees who were hired first having higher priority.

Write a procedure called AssignRaises that determines who will get raises. Your procedure should take the raise percentage (5% in our example above) and raise budget ($25,000 in our example above) as parameters, so that we can vary them when we need to.

The procedure should use a cursor to list the employees in the priority order described above. (Remember that you can use the ORDER BY clause in SQL to do this.) It will loop through the cursor and give raises to the employees using the RaiseSalary procedure you created in part 1. However, it needs to keep track of the declining budget, and not give more than $25,000 in raises. At a certain point, the remaining budget will have declined to the point that there is not enough money left to give the next person in the priority order a full 5% raise. When that happens, the procedure should stop giving raises at all, even if there is some money left in the raise budget.

For each person who does not get a raise, the procedure should write a message to the HW1Log table in the following form:

*Not enough money left to give a raise to employee <employee id>.*

Your procedure should finish by printing out the following report.

*Number of employees who received raises: <put value here>*

*Number of employees who did not receive raises: <put value here>*

*Amount of money left unused in the raise budget: <put value here>*

Again, your procedure should COMMIT its changes at the end, just for the sake of this exercise.

**PROCEDURE:**

CREATE OR REPLACE PROCEDURE assignRaises(

p\_emp\_raise\_percent NUMBER ,

p\_emp\_raise\_budget MyEmployees.salary%TYPE )

AS

p\_emp\_count NUMBER:=0;--keep track of total employee

p\_total\_emp\_budget NUMBER:= p\_emp\_raise\_budget;

p\_raise\_received\_count NUMBER:=0;--keep track of employee who get raise

v\_emp\_raise NUMBER:=0;--calculate raise

CURSOR emp\_priority IS

SELECT \* FROM MyEmployees e ORDER BY e.salary ASC ,e.hire\_date ASC;

BEGIN

SELECT COUNT(e.employee\_id) INTO p\_emp\_count FROM MyEmployees e;--get total unique emp count

FOR v\_emp in emp\_priority

LOOP

v\_emp\_raise:=v\_emp.salary\*p\_emp\_raise\_percent/100;

IF p\_total\_emp\_budget>=0 AND p\_total\_emp\_budget>=v\_emp\_raise THEN

raiseEmpSalary(v\_emp.employee\_id,p\_emp\_raise\_percent);

p\_raise\_received\_count := p\_raise\_received\_count +1;

p\_total\_emp\_budget := p\_total\_emp\_budget - v\_emp\_raise;

ELSE

INSERT INTO HW1Log (Message) VALUES ('Not enough money left to give a raise to employee ' || v\_emp.employee\_id || '.');

END IF;

END LOOP;

p\_emp\_count := p\_emp\_count - p\_raise\_received\_count;--update employee who didn’t receive raise

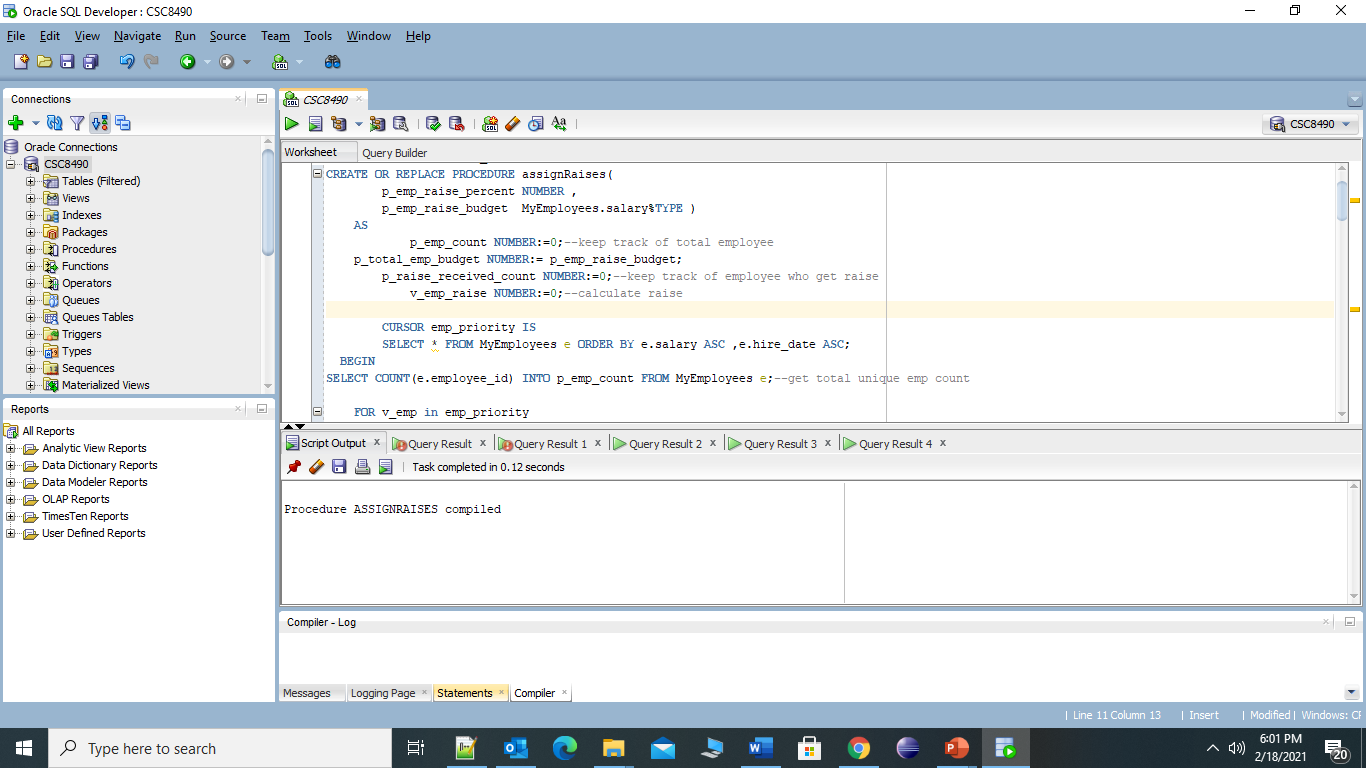
DBMS\_OUTPUT.put\_line('Number of employees who received raises: '|| p\_raise\_received\_count);

dbms\_output.put\_line('Number of employees who did not receive raises: '|| p\_emp\_count);

DBMS\_OUTPUT.PUT\_LINE('Amount of money left unused in the raise budget: '|| p\_total\_emp\_budget);

COMMIT;

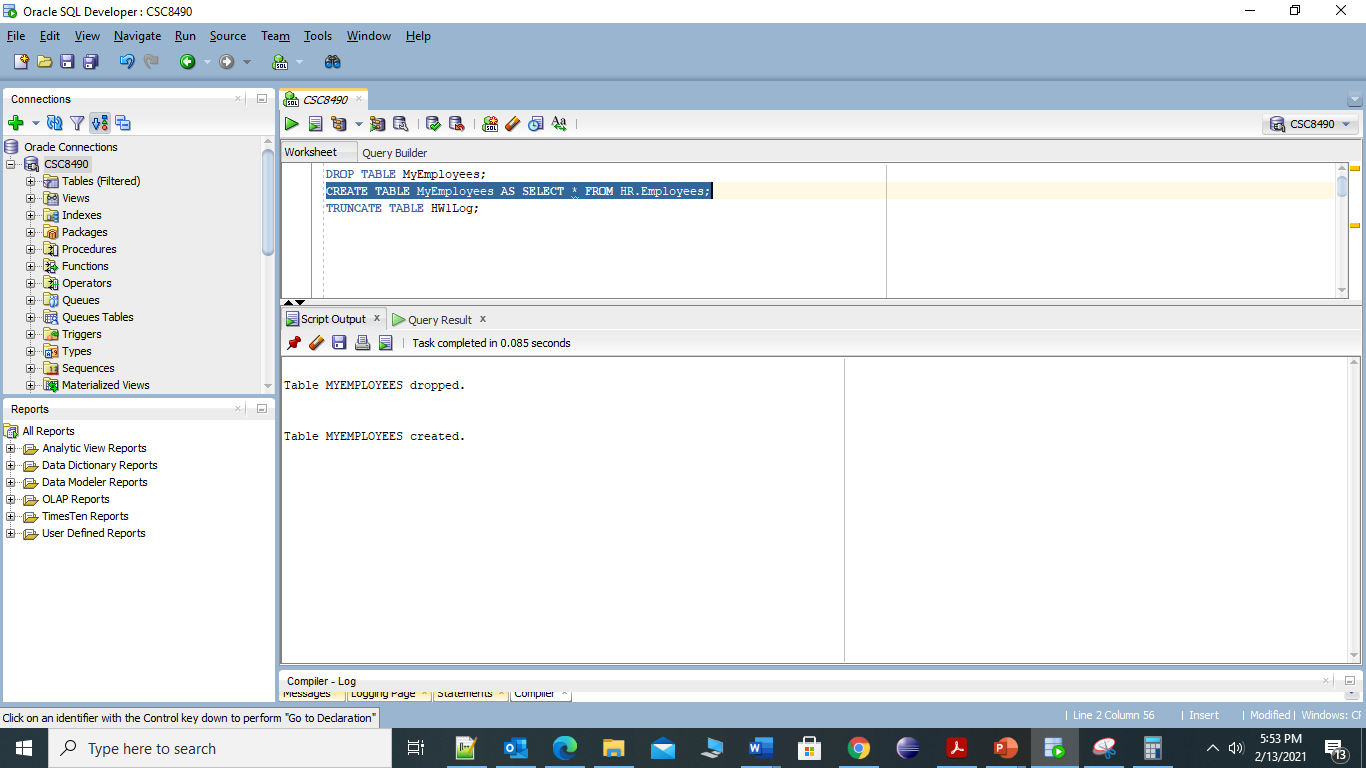
END assignRaises;



* Drop the MyEmployees table (DROP TABLE MyEmployees) and recreate it using the code from part 1 (CREATE TABLE MyEmployees AS SELECT \* FROM Employees)

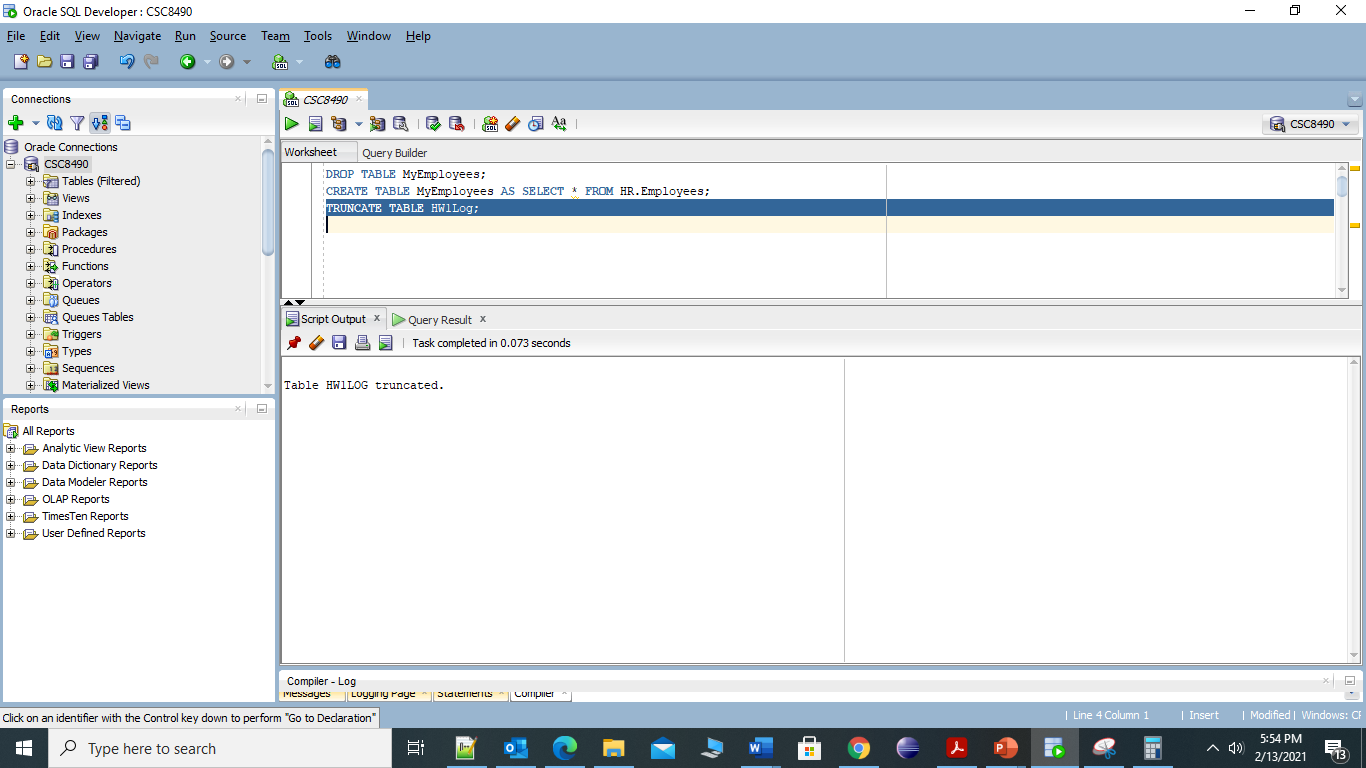
DROP TABLE MyEmployees;

CREATE TABLE MyEmployees AS SELECT \* FROM HR.Employees;



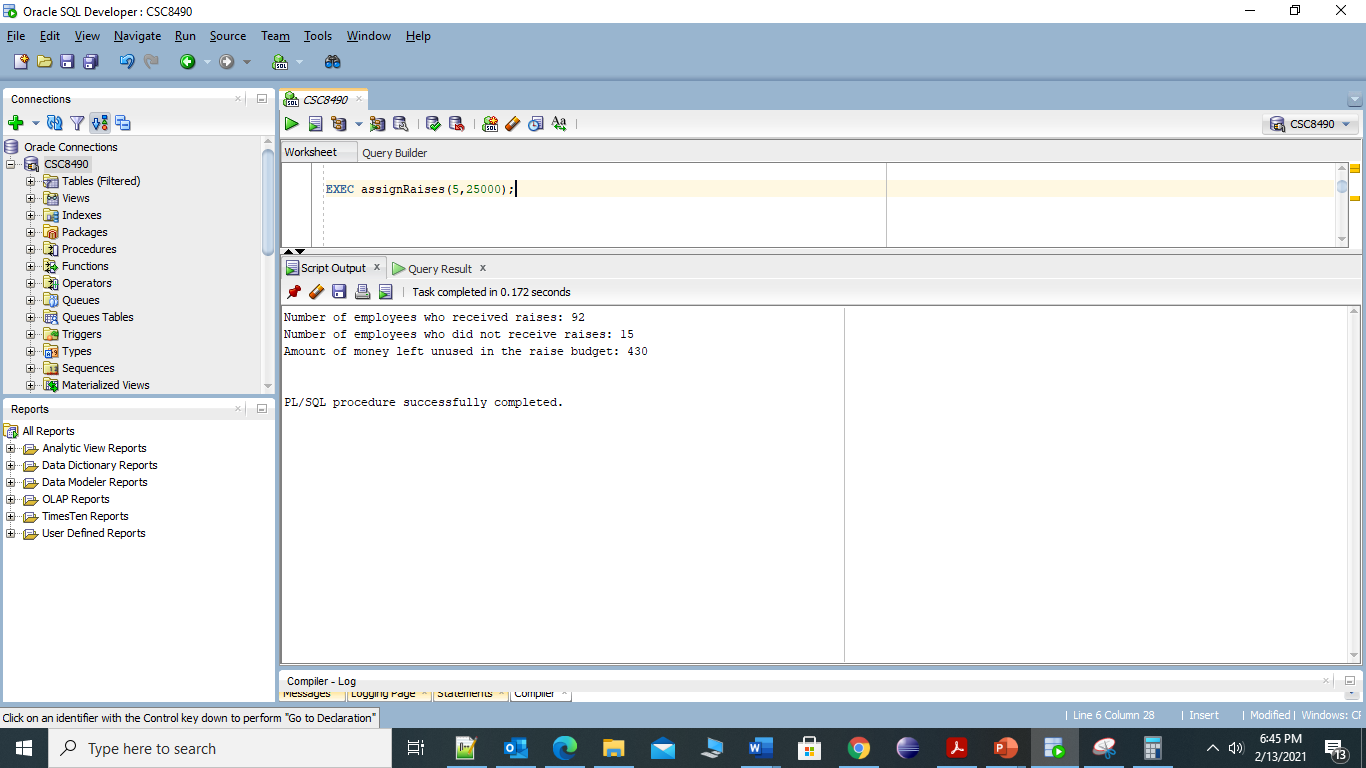
* Truncate the HW1Log table (TRUNCATE TABLE HW1Log) to remove prior records.

TRUNCATE TABLE HW1Log;



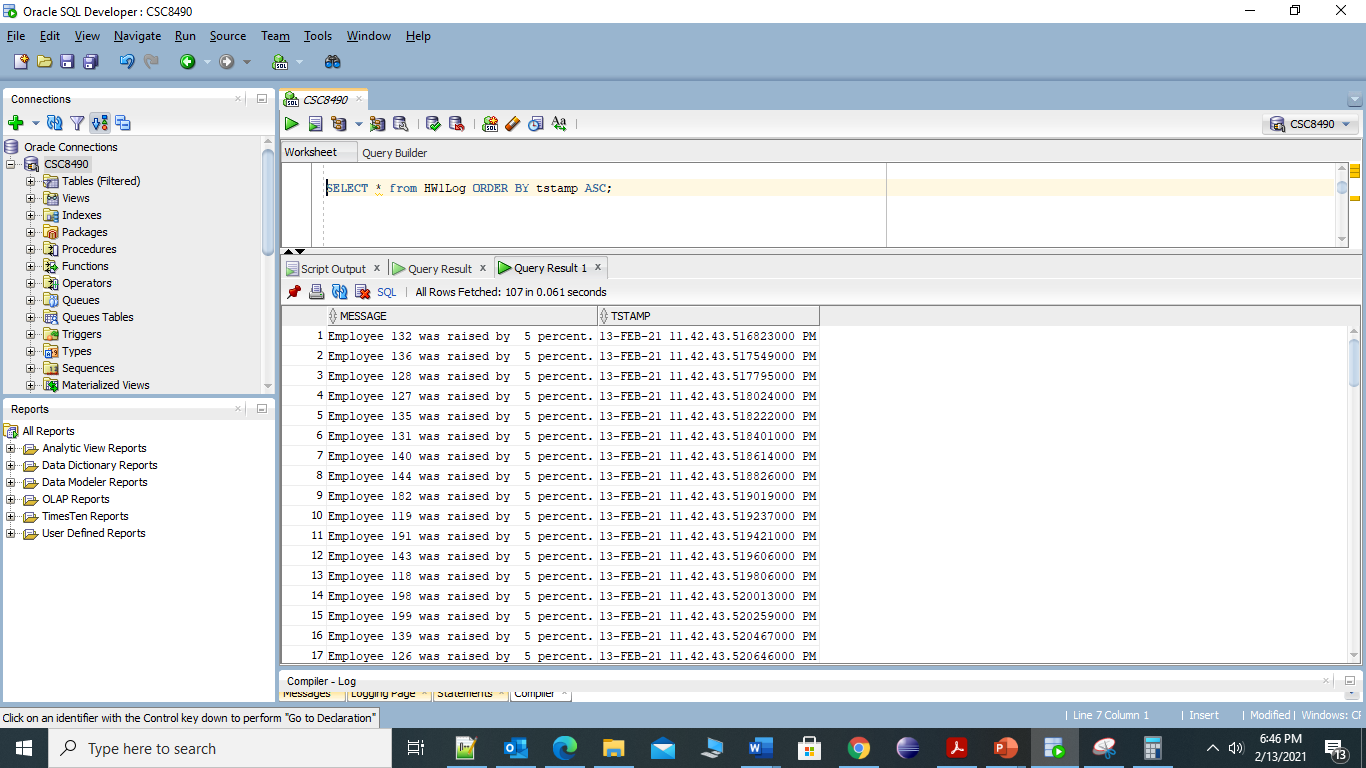
* Run the AssignRaises procedure with a raise percentage of **5** and a raise budget of **25000**.

EXEC assignRaises(5,25000);



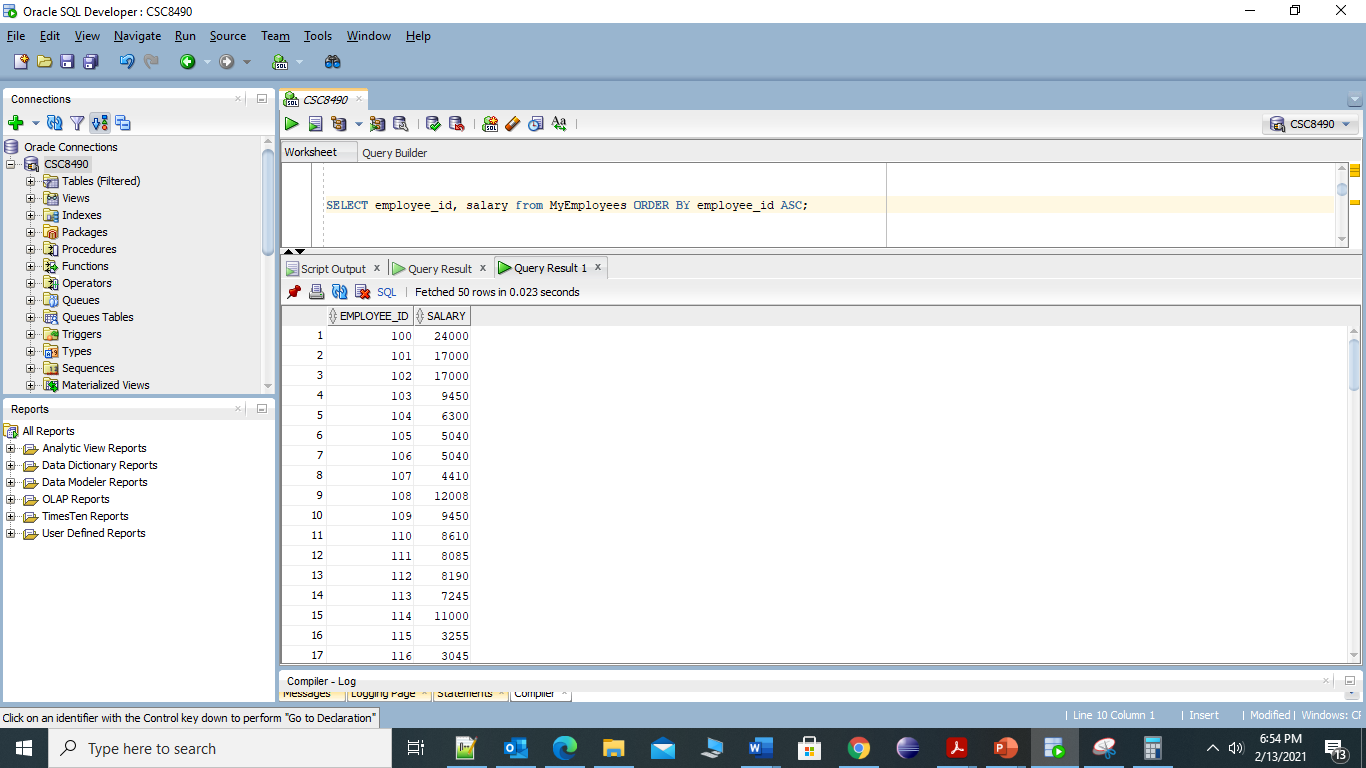
* Show all of the data from your HW1Log table (can do this with a standard SQL query) – order your results by ascending TStamp value.

SELECT \* from HW1Log ORDER BY tstamp ASC;



* Show the employee id and salary of all rows from the MyEmployees table (can do this with a standard SQL query) – order your results by employee id.

SELECT employee\_id, salary from MyEmployees ORDER BY employee\_id ASC;



* Drop the MyEmployees table (DROP TABLE MyEmployees) and recreate it using the code from part 1 (CREATE TABLE MyEmployees AS SELECT \* FROM Employees)

DROP TABLE MyEmployees;

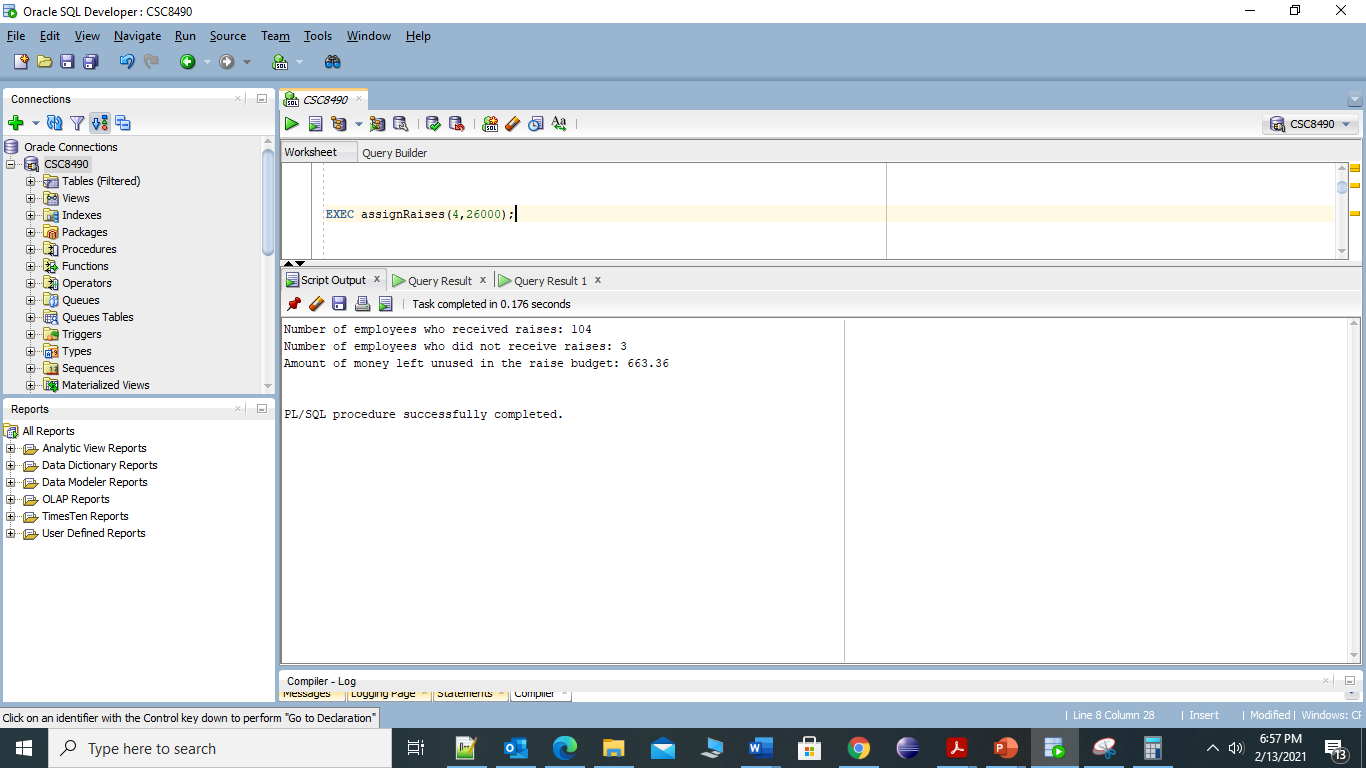
CREATE TABLE MyEmployees AS SELECT \* FROM HR.Employees;

* Truncate the HW1Log table (TRUNCATE TABLE HW1Log) to remove prior records.

TRUNCATE TABLE HW1Log;

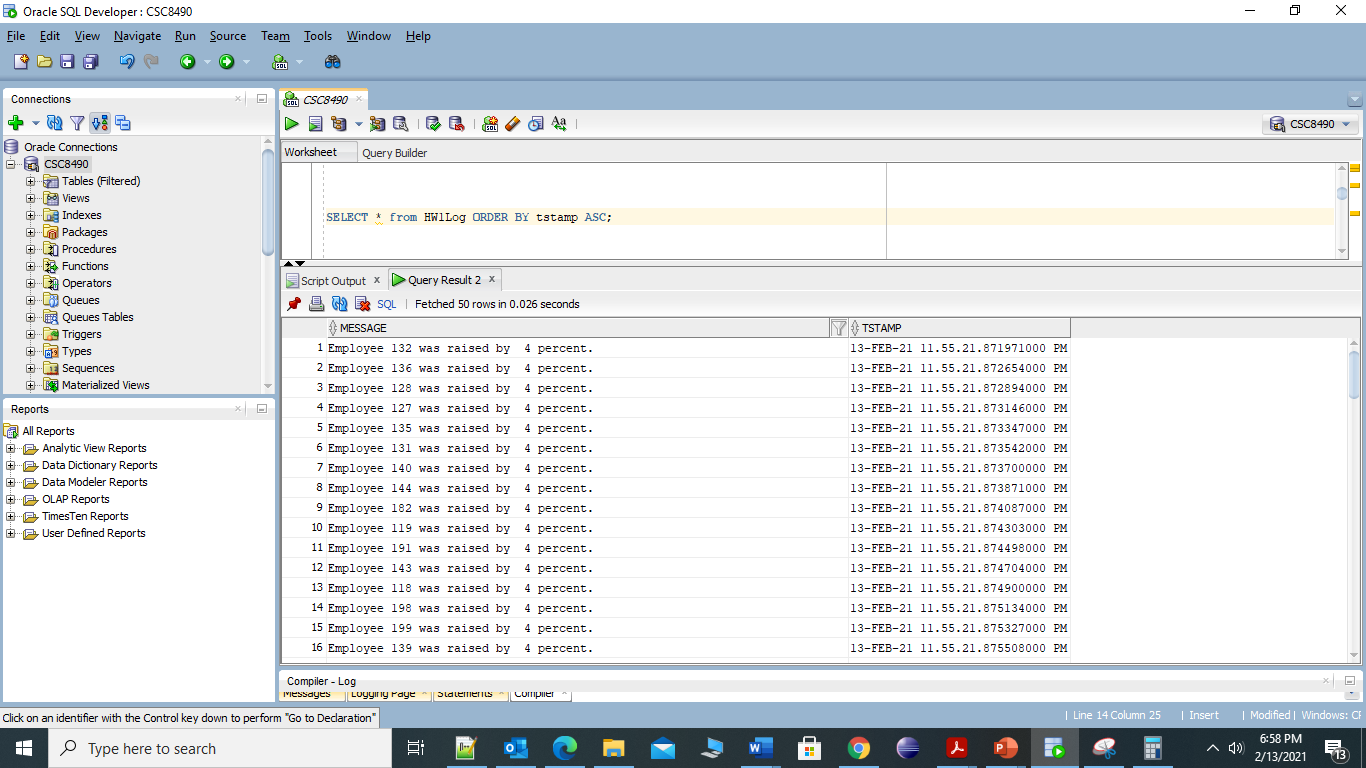
* Run the AssignRaises procedure with a raise percentage of **4** and a raise budget of **26000**.

EXEC assignRaises(4,26000);



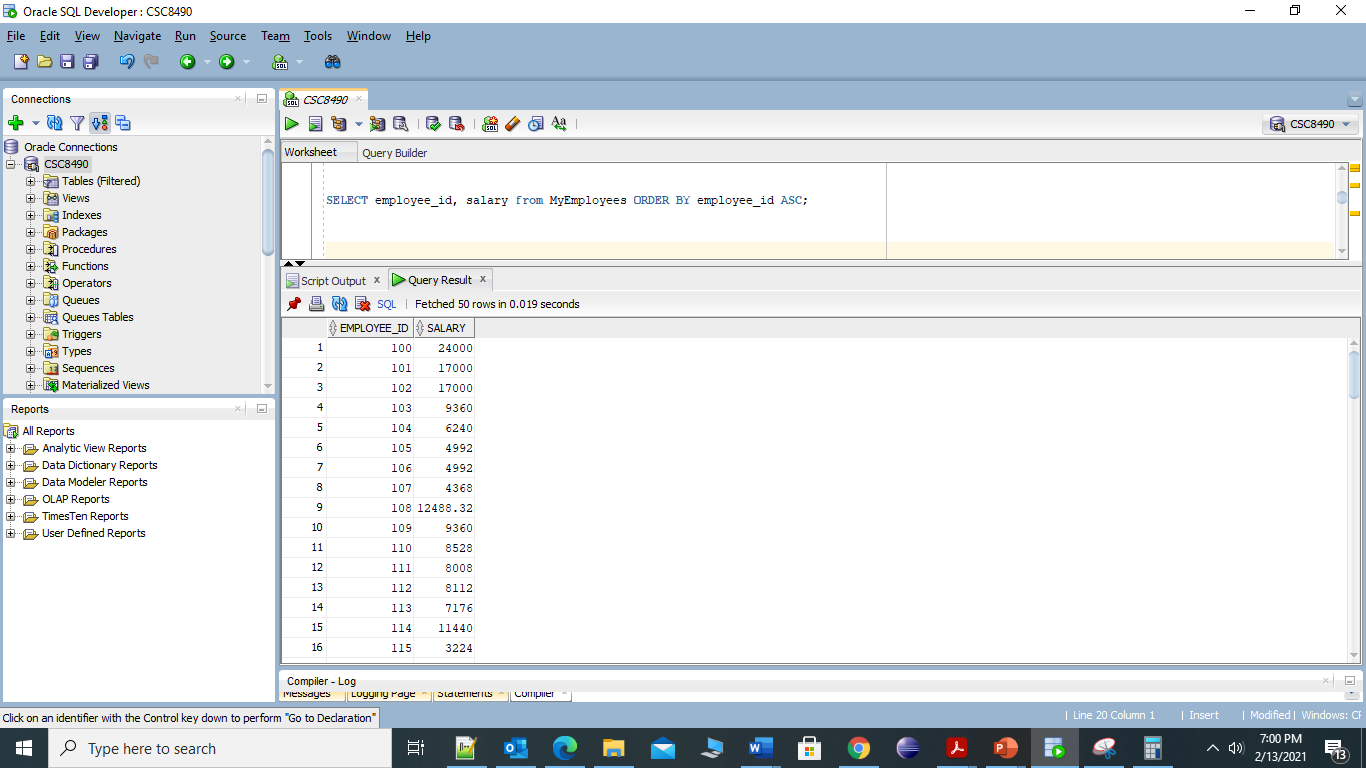
* Show all of the data from your HW1Log table (can do this with a standard SQL query) – order your results by ascending TStamp value.

SELECT \* from HW1Log ORDER BY tstamp ASC;



* Show the employee id and salary of all rows from the MyEmployees table (can do this with a standard SQL query) – order your results by employee id.

SELECT employee\_id, salary from MyEmployees ORDER BY employee\_id ASC;



By the way, this is an example of the kind of work that could be done in straight SQL, but it would require some complex statements. The PL/SQL version would be easier to understand for people who don’t do much complex SQL work.