# Case study 1

**5.1 Create appropriate Test Cases for the case study**

1. Test case for 5.2

a) Test Case for Inserting NRI Customer

b) Test Case for Inserting Indian Customer (IND)

c) Test Case for Inserting Customer with Invalid Type

2. Test case for 5.3

a) Valid Transfer from Existing Account to Existing Account

b) Valid Transfer from Existing Account to Non-Existing Account

c) Invalid Customer ID (Non-Existing)

d) For amount less than zero

e) Insufficient Balance for Transfer

f) To Account Number is Null

g) Transfer from Null Account to Non-Existing Account (Null To Account)

h) Transfer from Non-Existing Customer

**5.2 Write a procedure to accept customer name, address, and customer type and account type. Insert the details into the respective tables.**

CREATE OR REPLACE PROCEDURE insert\_customer(

name customer\_masters.cust\_name%TYPE,

address customer\_masters.address%TYPE,

c\_type customer\_masters.Customer\_Type%TYPE,

a\_type account\_masters.account\_Type%TYPE

) AS

c\_id number := cust\_seq.nextval;

a\_id number := acc\_seq.nextval;

min\_bal number;

BEGIN

IF a\_type = 'SAV' then

min\_bal := 5000;

ELSIF a\_type = 'SAL' then

min\_bal := 0;

END IF;

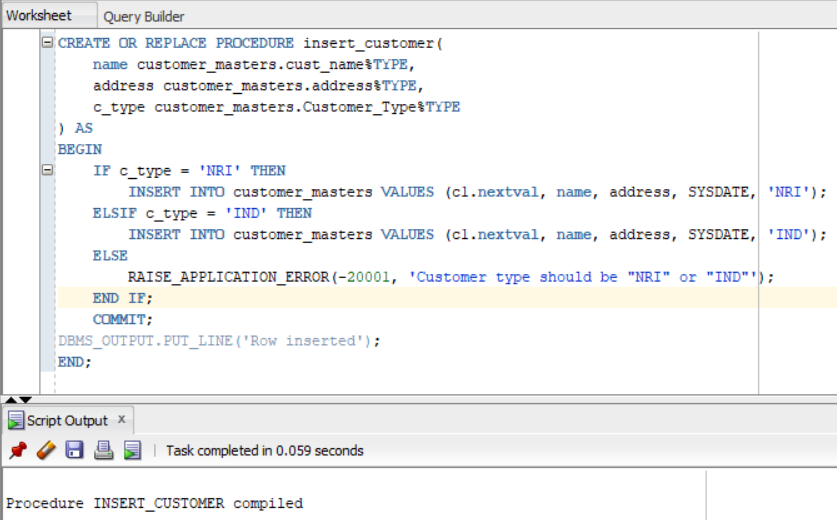
INSERT INTO customer\_masters VALUES (c\_id, name, address, SYSDATE, c\_type);

INSERT INTO account\_masters VALUES (a\_id, c\_id, a\_type, min\_bal);

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Row inserted');

END;



**5.3 Write a procedure to accept customer id, amount and the account number to which the customer requires to transfer money. Following validations need to be done**

* **Customer id should be valid**
* **From account number should belong to that customer**
* **To account number cannot be null but can be an account which need not exist in account masters (some other account)**
* **Adequate balance needs to be available for debit**

CREATE OR REPLACE PROCEDURE transfer\_money (

c\_id IN NUMBER,

amt IN NUMBER,

to\_acc\_no IN NUMBER

) AS

v\_from\_acc\_no NUMBER;

v\_from\_acc\_bal NUMBER;

v\_to\_acc\_bal NUMBER;

v\_c\_exist NUMBER;

BEGIN

-- Validation: Check if the customer id is valid

SELECT COUNT(\*) INTO v\_c\_exist FROM customer\_masters WHERE Cust\_Id = c\_id;

IF v\_c\_exist = 0 THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Invalid customer id.');

ELSIF v\_c\_exist > 1 THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Multiple customers found for the same id.');

END IF;

-- Validation: To account number cannot be null

IF to\_acc\_no IS NULL THEN

RAISE\_APPLICATION\_ERROR(-20004, 'To account number cannot be null.');

END IF;

-- Validation: To account number cannot be null

IF amt <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20005, 'The amount should be grater than 0.');

END IF;

-- Validation: From account number should belong to that customer

SELECT Ledger\_Balance, Account\_Number INTO v\_from\_acc\_bal, v\_from\_acc\_no

FROM Account\_Masters

WHERE Cust\_ID = c\_id;

IF v\_from\_acc\_bal IS NULL THEN

RAISE\_APPLICATION\_ERROR(-20007, 'From account does not belong to the customer.');

END IF;

-- Validation: Adequate balance needs to be available for debit

IF v\_from\_acc\_bal < amt THEN

RAISE\_APPLICATION\_ERROR(-20003, 'Insufficient balance for the requested transfer.');

END IF;

-- Perform the money transfer

UPDATE Account\_Masters

SET Ledger\_Balance = v\_from\_acc\_bal - amt

WHERE Cust\_ID = c\_id;

INSERT INTO transaction\_masters values(transact\_seq.nextval, v\_from\_acc\_no, sysdate, v\_from\_acc\_no, to\_acc\_no, amt, 'CR');

-- Update the target account balance only if it exists

BEGIN

SELECT Ledger\_Balance INTO v\_to\_acc\_bal

FROM Account\_Masters

WHERE Account\_Number = to\_acc\_no;

-- Update the target account balance

UPDATE Account\_Masters

SET Ledger\_Balance = NVL(v\_to\_acc\_bal, 0) + amt

WHERE Account\_Number = to\_acc\_no;

INSERT INTO transaction\_masters values(transact\_seq.nextval, to\_acc\_no, sysdate, v\_from\_acc\_no, to\_acc\_no, amt, 'DB');

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

NULL;

END;

DBMS\_OUTPUT.PUT\_LINE('Money transfer successful.');

COMMIT;

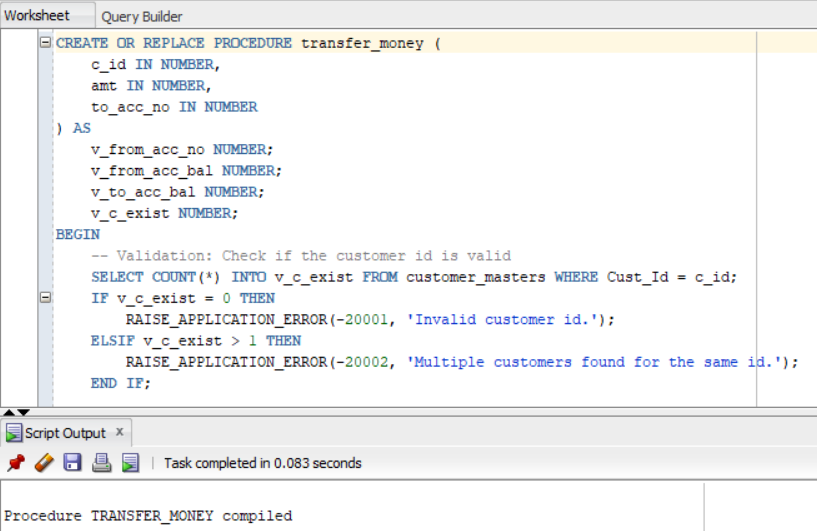
EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLCODE || ' - ' || SQLERRM);

END;



-- Package Specification

CREATE OR REPLACE PACKAGE case\_study\_1 AS

PROCEDURE insert\_customer(

name customer\_masters.cust\_name%TYPE,

address customer\_masters.address%TYPE,

c\_type customer\_masters.Customer\_Type%TYPE,

a\_type account\_masters.account\_Type%TYPE

);

PROCEDURE transfer\_money (

c\_id IN NUMBER,

amt IN NUMBER,

to\_acc\_no IN NUMBER

);

END;

-- Package Body

CREATE OR REPLACE PACKAGE BODY case\_study\_1 AS

PROCEDURE insert\_customer(

name customer\_masters.cust\_name%TYPE,

address customer\_masters.address%TYPE,

c\_type customer\_masters.Customer\_Type%TYPE,

a\_type account\_masters.account\_Type%TYPE

) AS

c\_id number := cust\_seq.nextval;

a\_id number := acc\_seq.nextval;

min\_bal number;

BEGIN

IF a\_type = 'SAV' then

min\_bal := 5000;

ELSIF a\_type = 'SAL' then

min\_bal := 0;

END IF;

INSERT INTO customer\_masters VALUES (c\_id, name, address, SYSDATE, c\_type);

INSERT INTO account\_masters VALUES (a\_id, c\_id, a\_type, min\_bal);

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Row inserted');

END;

PROCEDURE transfer\_money (

c\_id IN NUMBER,

amt IN NUMBER,

to\_acc\_no IN NUMBER

) AS

v\_from\_acc\_no NUMBER;

v\_from\_acc\_bal NUMBER;

v\_to\_acc\_bal NUMBER;

v\_c\_exist NUMBER;

BEGIN

-- Validation: Check if the customer id is valid

SELECT COUNT(\*) INTO v\_c\_exist FROM customer\_masters WHERE Cust\_Id = c\_id;

IF v\_c\_exist = 0 THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Invalid customer id.');

ELSIF v\_c\_exist > 1 THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Multiple customers found for the same id.');

END IF;

-- Validation: To account number cannot be null

IF to\_acc\_no IS NULL THEN

RAISE\_APPLICATION\_ERROR(-20004, 'To account number cannot be null.');

END IF;

-- Validation: To account number cannot be null

IF amt <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20005, 'The amount should be grater than 0.');

END IF;

-- Validation: From account number should belong to that customer

SELECT Ledger\_Balance, Account\_Number INTO v\_from\_acc\_bal, v\_from\_acc\_no

FROM Account\_Masters

WHERE Cust\_ID = c\_id;

IF v\_from\_acc\_bal IS NULL THEN

RAISE\_APPLICATION\_ERROR(-20007, 'From account does not belong to the customer.');

END IF;

-- Validation: Adequate balance needs to be available for debit

IF v\_from\_acc\_bal < amt THEN

RAISE\_APPLICATION\_ERROR(-20003, 'Insufficient balance for the requested transfer.');

END IF;

-- Perform the money transfer

UPDATE Account\_Masters

SET Ledger\_Balance = v\_from\_acc\_bal - amt

WHERE Cust\_ID = c\_id;

INSERT INTO transaction\_masters values(transact\_seq.nextval, v\_from\_acc\_no, sysdate, v\_from\_acc\_no, to\_acc\_no, amt, 'CR');

-- Update the target account balance only if it exists

BEGIN

SELECT Ledger\_Balance INTO v\_to\_acc\_bal

FROM Account\_Masters

WHERE Account\_Number = to\_acc\_no;

-- Update the target account balance

UPDATE Account\_Masters

SET Ledger\_Balance = NVL(v\_to\_acc\_bal, 0) + amt

WHERE Account\_Number = to\_acc\_no;

INSERT INTO transaction\_masters values(transact\_seq.nextval, to\_acc\_no, sysdate, v\_from\_acc\_no, to\_acc\_no, amt, 'DB');

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

NULL;

END;

DBMS\_OUTPUT.PUT\_LINE('Money transfer successful.');

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLCODE || ' - ' || SQLERRM);

END;

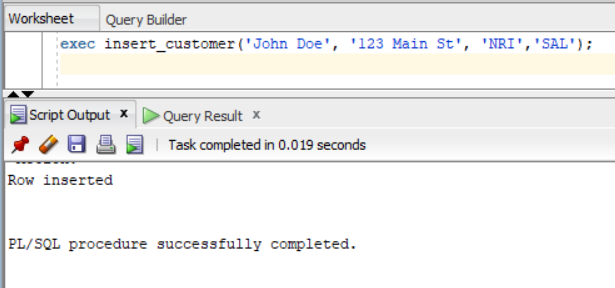
END;

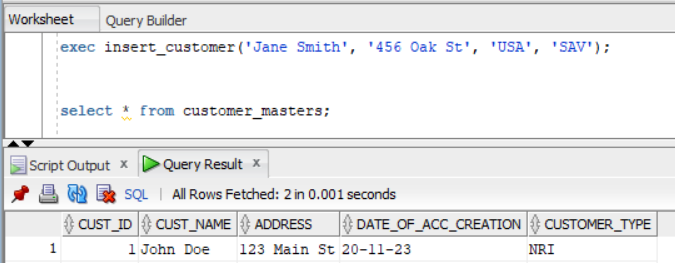
**5.4 Ensure all the Test cases defined are executed**

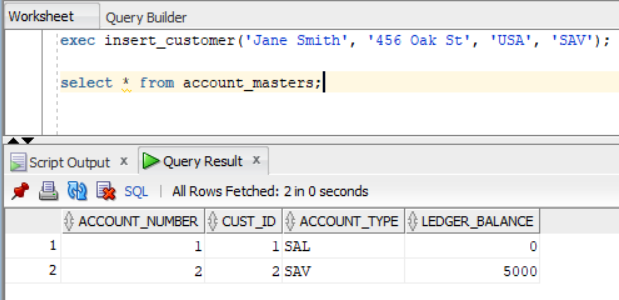
1. Test case for 5.2

a) Test Case for Inserting NRI Customer:

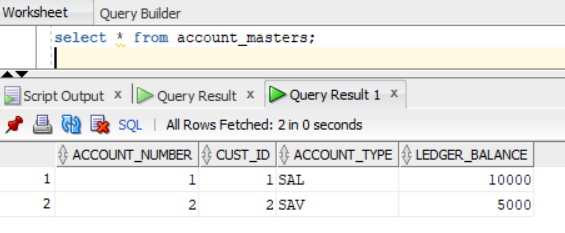
exec insert\_customer('John Doe', '123 Main St', 'NRI',’SAL’);





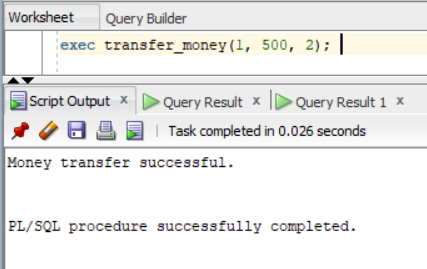


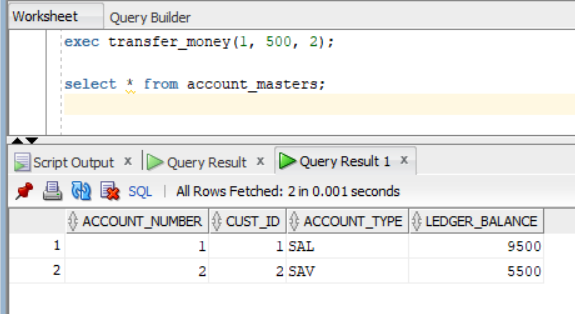
2. Test case for 5.3



a) Valid Transfer from Existing Account to Existing Account:

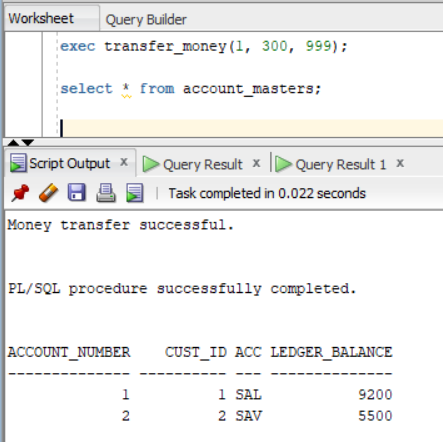
exec transfer\_money(1, 500, 2);





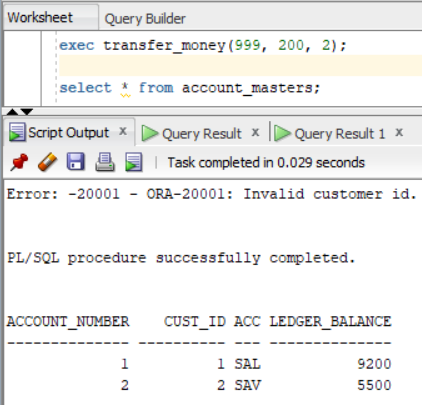
b) Valid Transfer from Existing Account to Non-Existing Account:

exec transfer\_money(1, 300, 999);



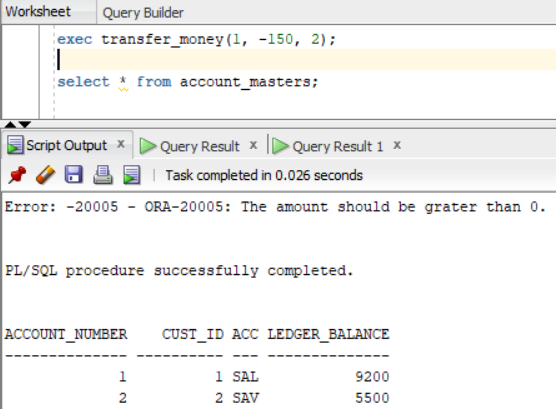
c) Invalid Customer ID (Non-Existing):

exec transfer\_money(999, 200, 2);



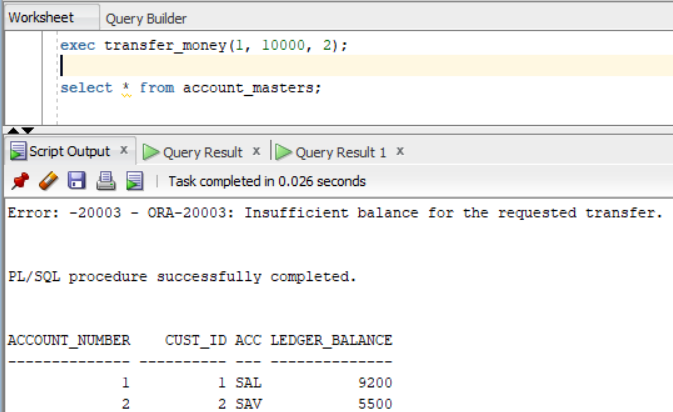
d) For amount less than zero:

exec transfer\_money(1, -150, 2);



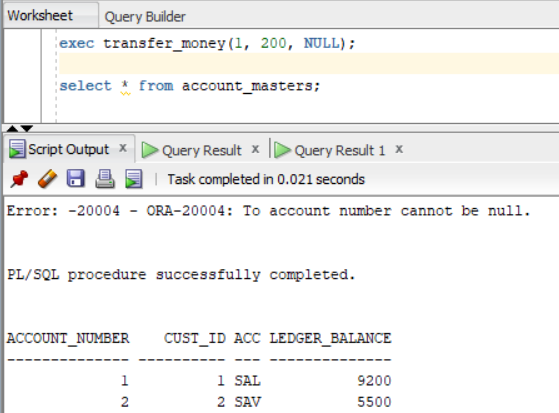
e) Insufficient Balance for Transfer:

exec transfer\_money(1, 10000, 2);



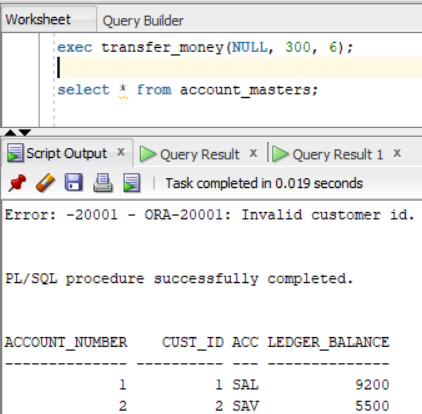
f) To Account Number is Null:

exec transfer\_money(1, 200, NULL);



g) Transfer from Null Account to Non-Existing Account (Null To Account):

exec transfer\_money(NULL, 300, 6);



# Case study 2

**6.1 Create appropriate Test Cases for the case study followed up by Self/Peer to Peer**

1. Test case for 6.2
   1. Valid case for add\_emp
   2. Invalid case for add\_emp
   3. Valid case for raise\_sal
   4. Invalid case for raise\_sal
   5. Valid case for remove\_emp
   6. Invalid case for remove\_emp
2. Test case for 6.3
   1. Valid case for add\_emp
   2. Invalid case for add\_emp
   3. Valid case for raise\_sal
   4. Invalid case for raise\_sal
   5. Valid case for remove\_emp
   6. Invalid case for remove\_emp
3. Test case for 6.4
   1. Test Case for Inserting a New Employee
   2. Test Case for Updating Employee Salary (within the limit)
   3. Test Case for Updating Employee Salary (exceeding the limit)
   4. Test Case for Deleting an Employee
   5. Test Case for Deleting Non-Existent Employee (no\_data\_found scenario)

**6.2 Recreate the procedure (run\_task) which is more efficient in performing the same.**

CREATE OR REPLACE PROCEDURE run\_task (

task\_number\_in IN INTEGER,

emp\_number\_in IN NUMBER DEFAULT NULL,

ename\_in IN VARCHAR2 DEFAULT NULL,

city\_in IN VARCHAR2 DEFAULT NULL,

designation\_in IN VARCHAR2 DEFAULT NULL,

salary\_in IN NUMBER DEFAULT NULL

)

IS

BEGIN

CASE task\_number\_in

WHEN 1 THEN

add\_emp(ename\_in, city\_in, designation\_in, salary\_in);

WHEN 2 THEN

raise\_sal(emp\_number\_in, salary\_in);

WHEN 3 THEN

remove\_emp(emp\_number\_in);

ELSE

RAISE\_APPLICATION\_ERROR(-20002, 'Invalid task number.');

END CASE;

EXCEPTION

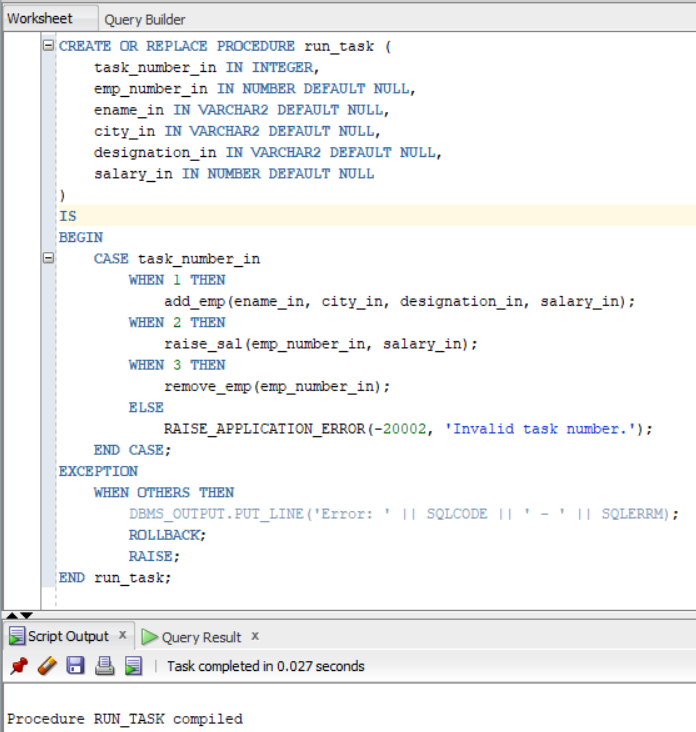
WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLCODE || ' - ' || SQLERRM);

ROLLBACK;

RAISE;

END run\_task;



**6.3 Also, create relevant procedures (add\_emp , raise\_sal ,remove\_emp) with relevant logic (read comments)to verify the same.**

CREATE OR REPLACE PROCEDURE add\_emp(

v\_ename myEmp.ename%type,

v\_city myEmp.city%type DEFAULT 'Mumbai',

v\_des myEmp.Designation%type,

v\_sal myEmp.Salary%type

) AS

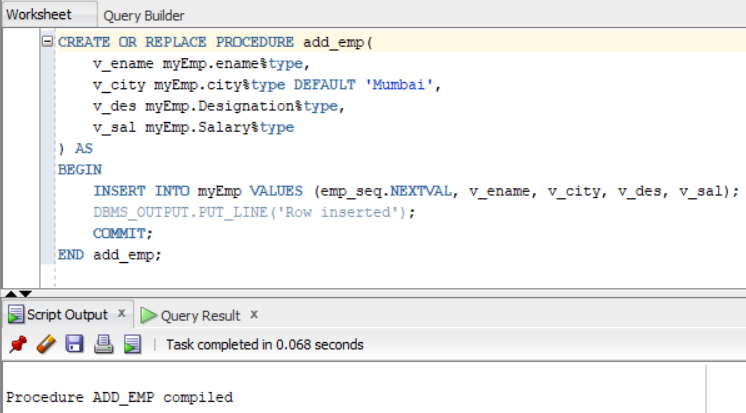
BEGIN

INSERT INTO myEmp VALUES (emp\_seq.NEXTVAL, v\_ename, v\_city, v\_des, v\_sal);

DBMS\_OUTPUT.PUT\_LINE('Row inserted');

COMMIT;

END add\_emp;



CREATE OR REPLACE PROCEDURE raise\_sal (

p\_empno IN myEmp.EmpNo%type,

p\_sal IN myEmp.Salary%type

) AS

v\_current\_salary NUMBER;

v\_upper\_limit NUMBER := 1.30;

BEGIN

SELECT Salary INTO v\_current\_salary FROM myEmp WHERE EmpNo = p\_empno;

IF v\_current\_salary IS NULL THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Employee not found.');

END IF;

IF p\_sal > v\_current\_salary \* v\_upper\_limit THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Salary raise exceeds the 30% limit.');

END IF;

UPDATE myEmp SET Salary = p\_sal WHERE EmpNo = p\_empno;

DBMS\_OUTPUT.PUT\_LINE('Row updated');

COMMIT;

EXCEPTION

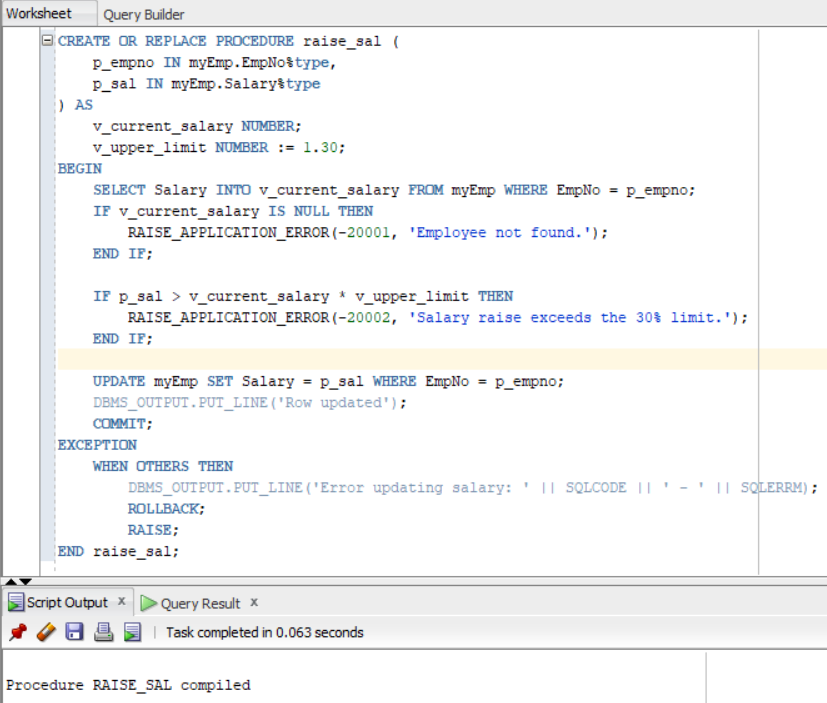
WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error updating salary: ' || SQLCODE || ' - ' || SQLERRM);

ROLLBACK;

RAISE;

END raise\_sal;



CREATE OR REPLACE PROCEDURE remove\_emp(v\_empno myEmp.EmpNo%type) AS

v\_name VARCHAR2(50);

BEGIN

SELECT ename INTO v\_name FROM myEmp WHERE EmpNo = v\_empno;

DELETE FROM myEmp WHERE EmpNo = v\_empno;

DBMS\_OUTPUT.PUT\_LINE('Row deleted');

COMMIT;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('No such employee id');

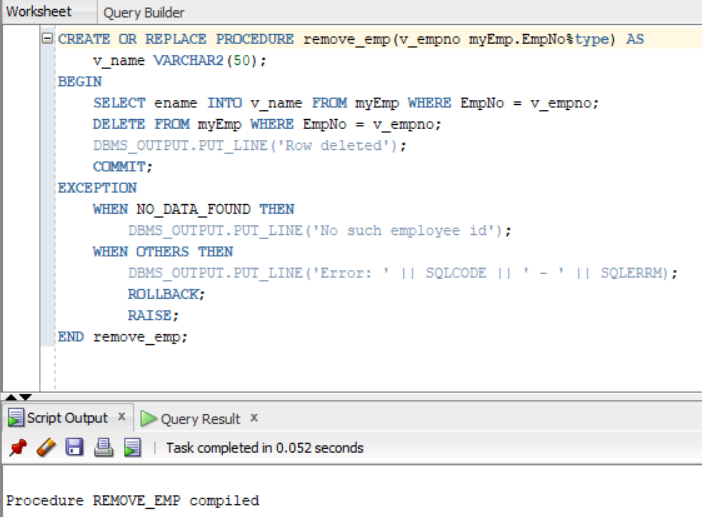
WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLCODE || ' - ' || SQLERRM);

ROLLBACK;

RAISE;

END remove\_emp;



**6.4 Extend the above implementation using Packages**

-- Package Specification

CREATE OR REPLACE PACKAGE case\_study\_2 AS

PROCEDURE run\_task(

ename\_in IN VARCHAR2 DEFAULT NULL,

salary\_in IN NUMBER DEFAULT NULL,

city\_in IN VARCHAR2 DEFAULT 'Mumbai',

designation\_in IN VARCHAR2 DEFAULT NULL

);

PROCEDURE run\_task(emp\_number\_in IN NUMBER, salary\_in IN NUMBER);

PROCEDURE run\_task(emp\_number\_in IN NUMBER);

END;

-- Package Body

CREATE OR REPLACE PACKAGE BODY case\_study\_2 AS

PROCEDURE run\_task(

ename\_in IN VARCHAR2 DEFAULT NULL,

salary\_in IN NUMBER DEFAULT NULL,

city\_in IN VARCHAR2 DEFAULT 'Mumbai',

designation\_in IN VARCHAR2 DEFAULT NULL

) AS

BEGIN

INSERT INTO myEmp VALUES (emp\_seq.nextval, ename\_in, city\_in, designation\_in, salary\_in);

DBMS\_OUTPUT.PUT\_LINE('Row inserted');

END;

PROCEDURE run\_task(emp\_number\_in IN NUMBER, salary\_in IN NUMBER) AS

v\_current\_salary NUMBER;

v\_upper\_limit NUMBER := 1.30;

BEGIN

SELECT Salary INTO v\_current\_salary FROM myEmp WHERE EmpNo = emp\_number\_in;

IF v\_current\_salary IS NULL THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Employee not found.');

END IF;

DECLARE

v\_proposed\_salary NUMBER := v\_current\_salary \* v\_upper\_limit;

BEGIN

IF salary\_in > v\_proposed\_salary THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Salary raise exceeds the 30% limit.');

END IF;

UPDATE myEmp SET Salary = salary\_in WHERE EmpNo = emp\_number\_in;

DBMS\_OUTPUT.PUT\_LINE('Row updated');

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

RAISE\_APPLICATION\_ERROR(-20003, 'Error updating salary: ' || SQLCODE || ' - ' || SQLERRM);

END;

END;

PROCEDURE run\_task(emp\_number\_in IN NUMBER) AS

v\_name VARCHAR(50);

BEGIN

SELECT ename INTO v\_name FROM myEmp WHERE EmpNo = emp\_number\_in;

DELETE FROM myEmp WHERE EmpNo = emp\_number\_in;

DBMS\_OUTPUT.PUT\_LINE('Row deleted');

COMMIT;

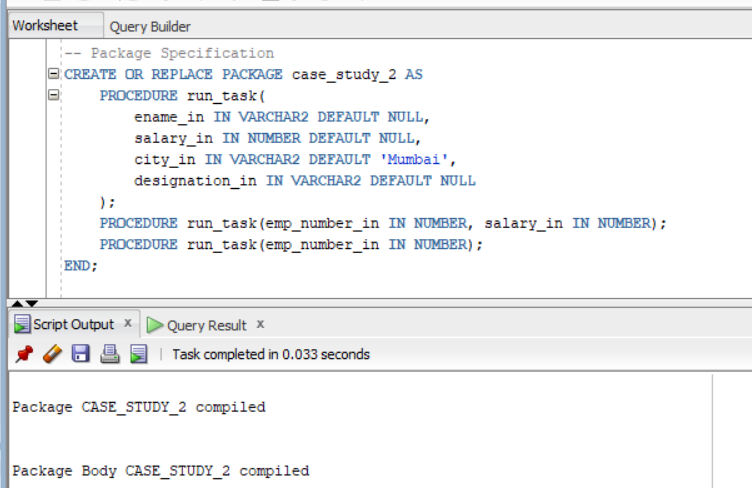
EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('No such employee ID');

END;

END;

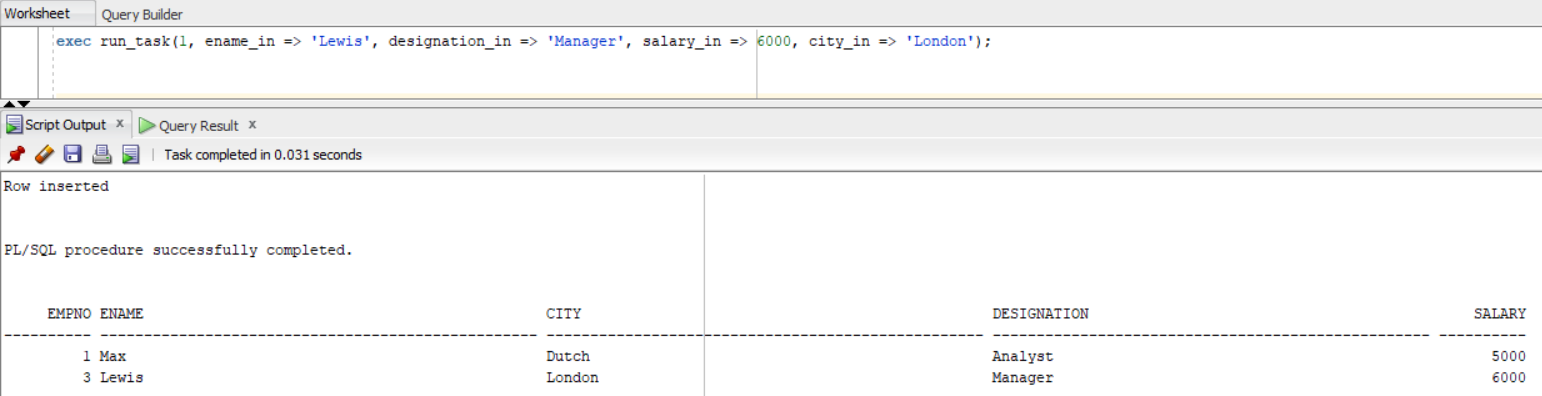


**6.5 Ensure all the Test cases defined are executed.**

1. Test case for 6.2

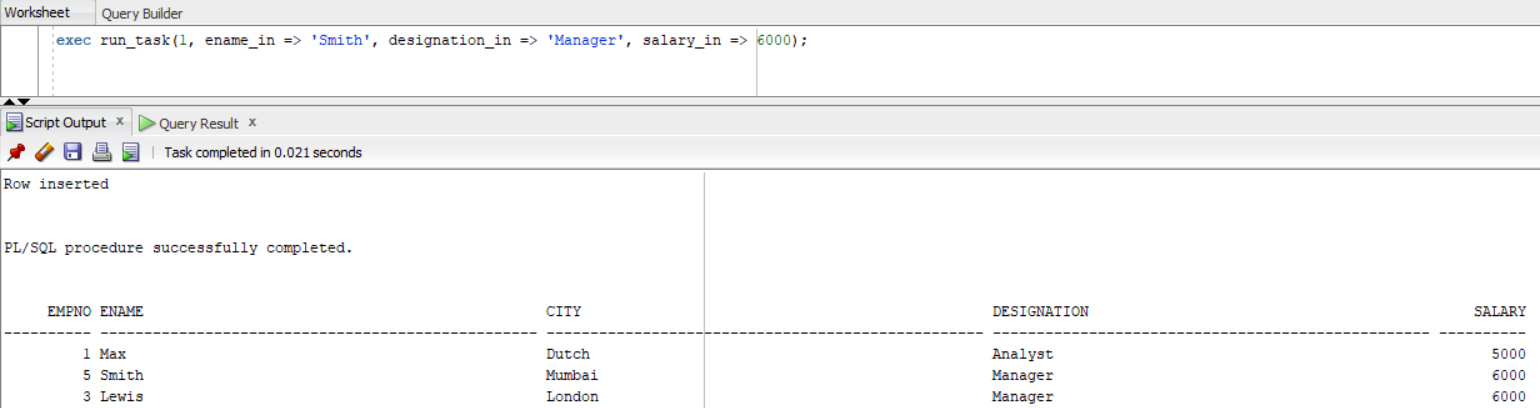
a) Valid case for add\_emp

exec run\_task(1, ename\_in => 'Lewis', designation\_in => 'Manager', salary\_in => 6000, city\_in => 'London');



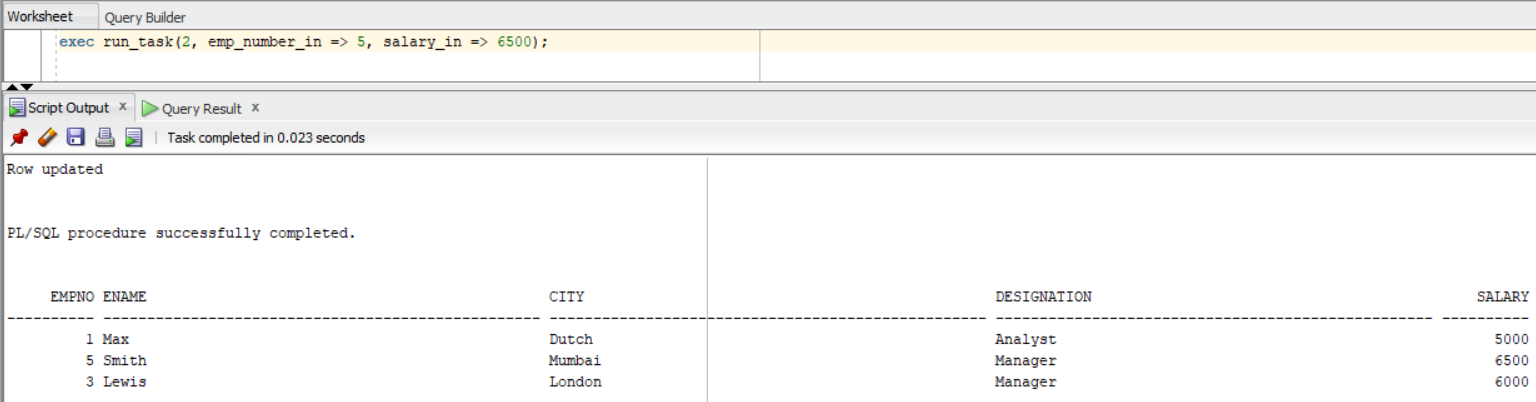
b) Case default location for add\_emp

exec run\_task(1, ename\_in => 'Smith', designation\_in => 'Manager', salary\_in => 6000);



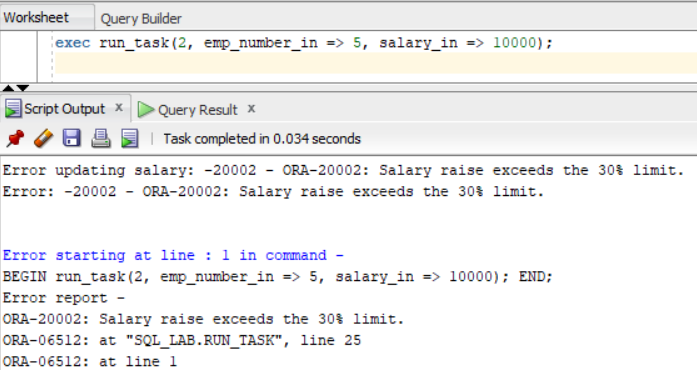
c) Valid case for raise\_sal

exec run\_task(2, emp\_number\_in => 5, salary\_in => 6500);



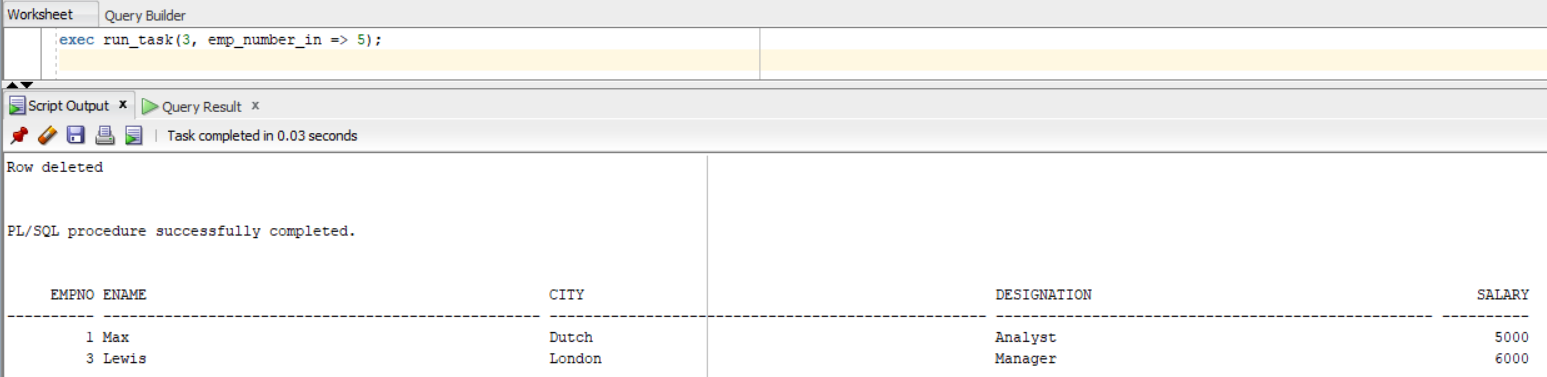
d) Invalid case for raise\_sal

exec run\_task(2, emp\_number\_in => 5, salary\_in => 10000);



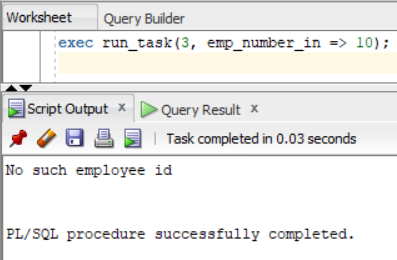
e) Valid case for remove\_emp

exec run\_task(3, emp\_number\_in => 5);



f) Invalid case for remove\_emp

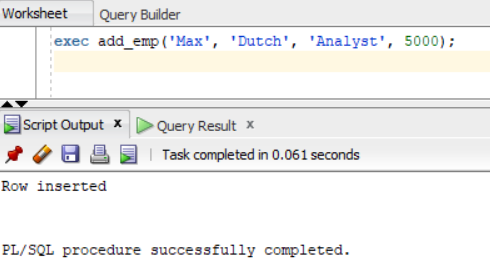
exec run\_task(3, emp\_number\_in => 10);



2.Test case for 6.3

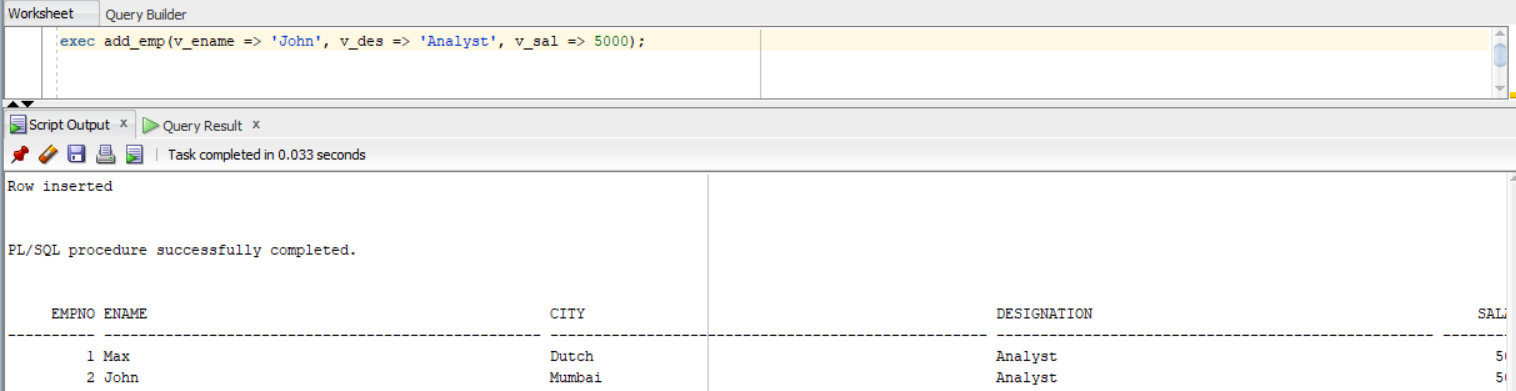
a) Valid case for add\_emp

exec add\_emp('Max', 'Dutch', 'Analyst', 5000);



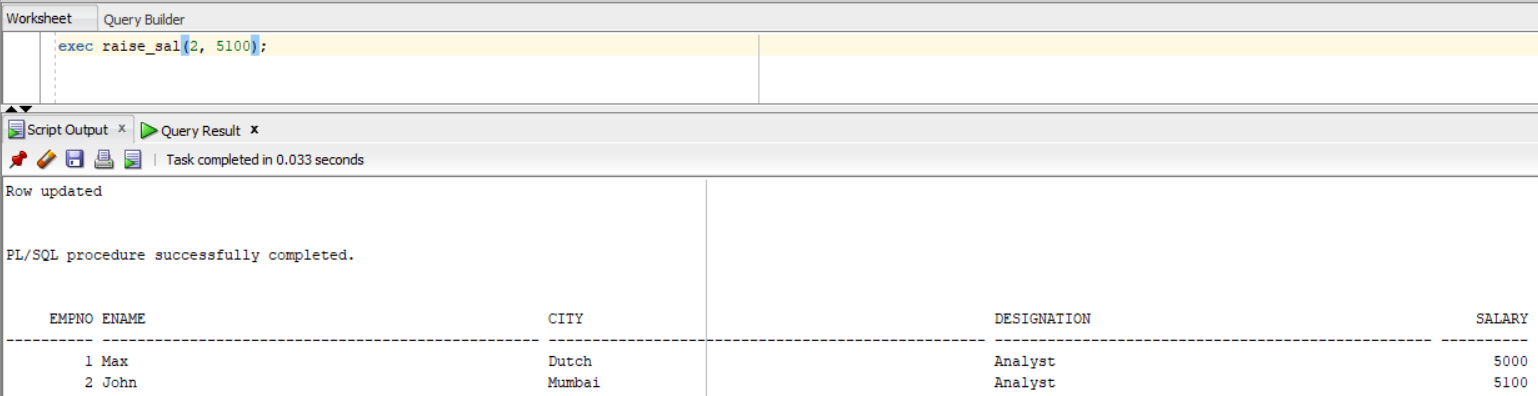
b) Case default location for add\_emp

exec add\_emp(v\_ename => 'John', v\_des => 'Analyst', v\_sal => 5000);



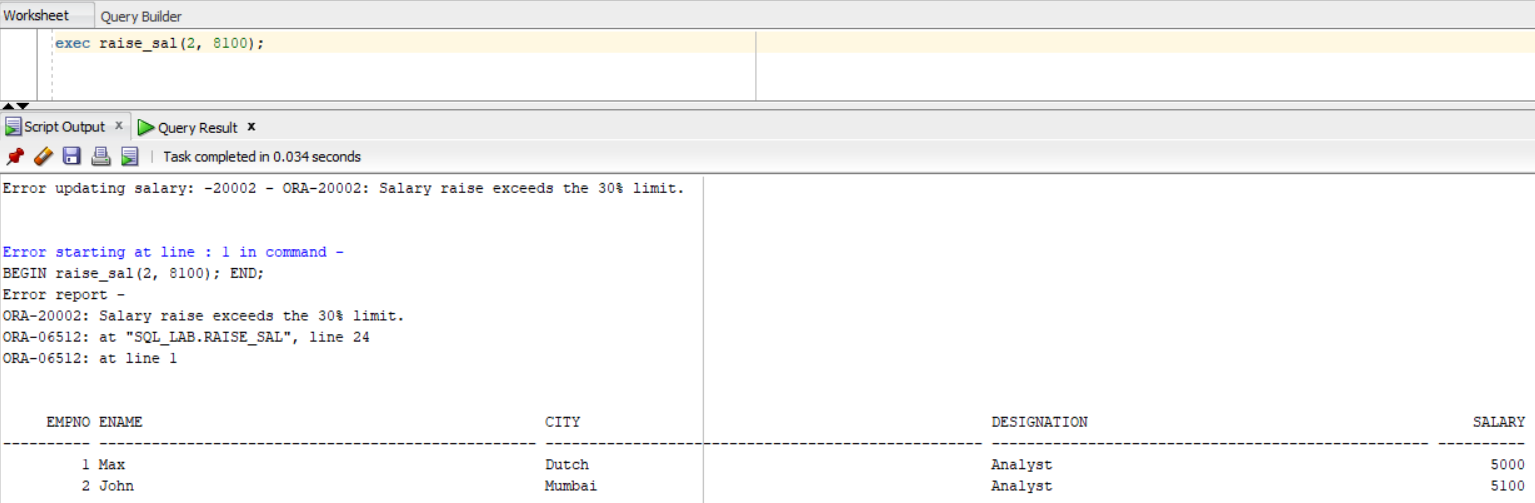
c) Valid case for raise\_sal

exec raise\_sal(2, 5100);



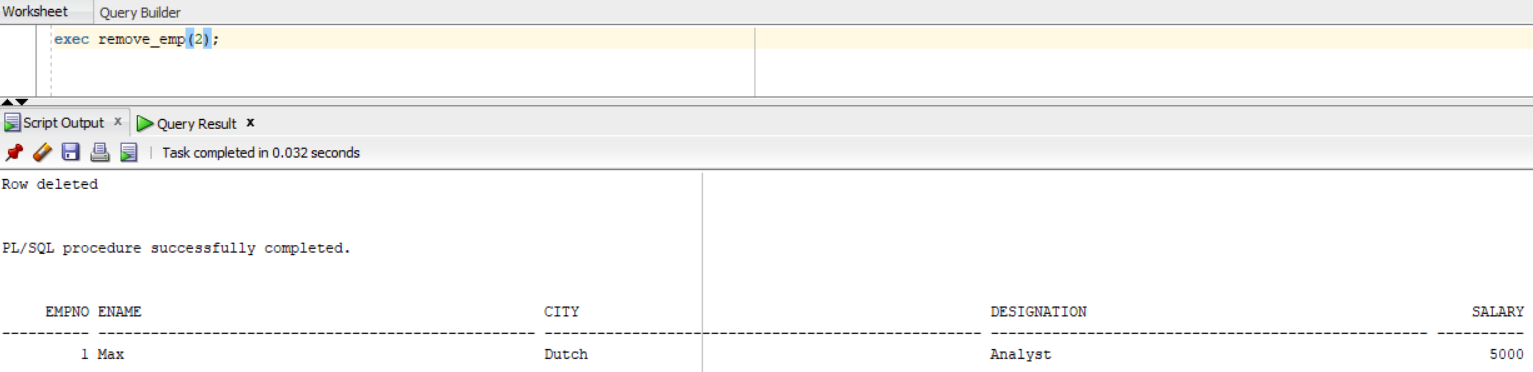
d) Invalid case for raise\_sal

exec raise\_sal(2, 8100);



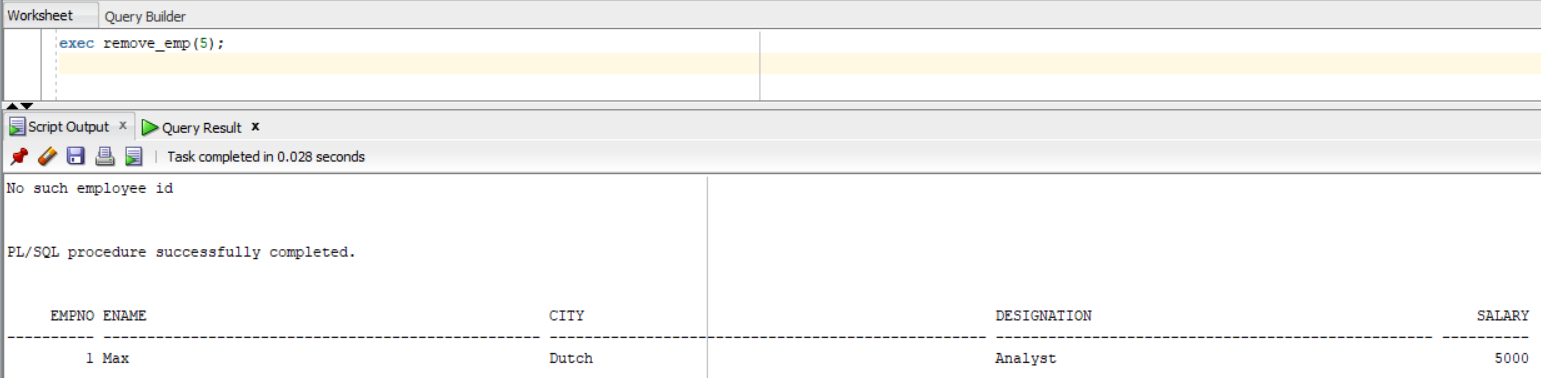
e) Valid case for remove\_emp

exec remove\_emp(2);



f) Invalid case for remove\_emp

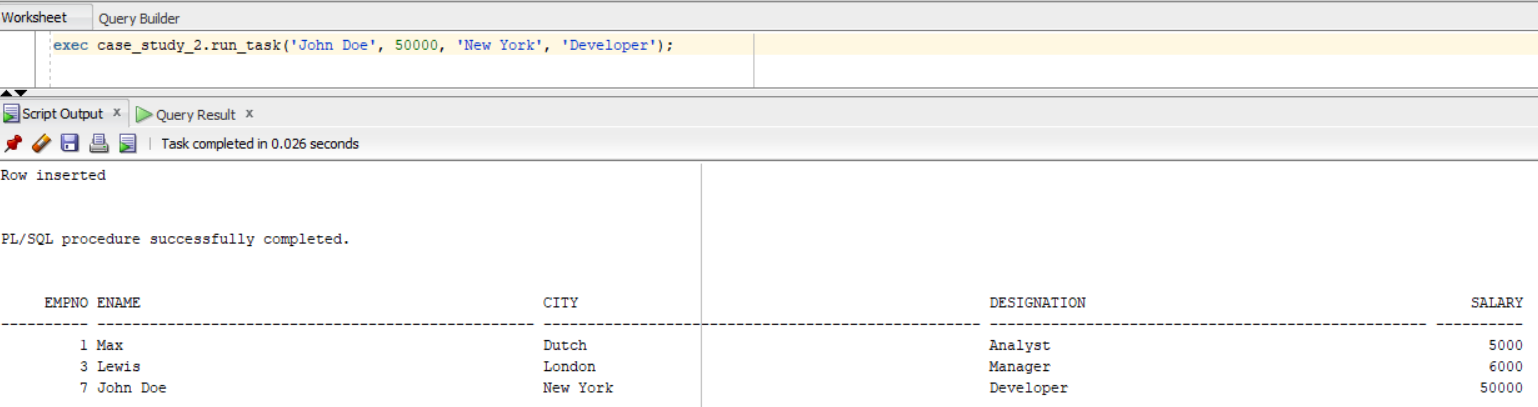
exec remove\_emp(5);



3. Test case for 6.4

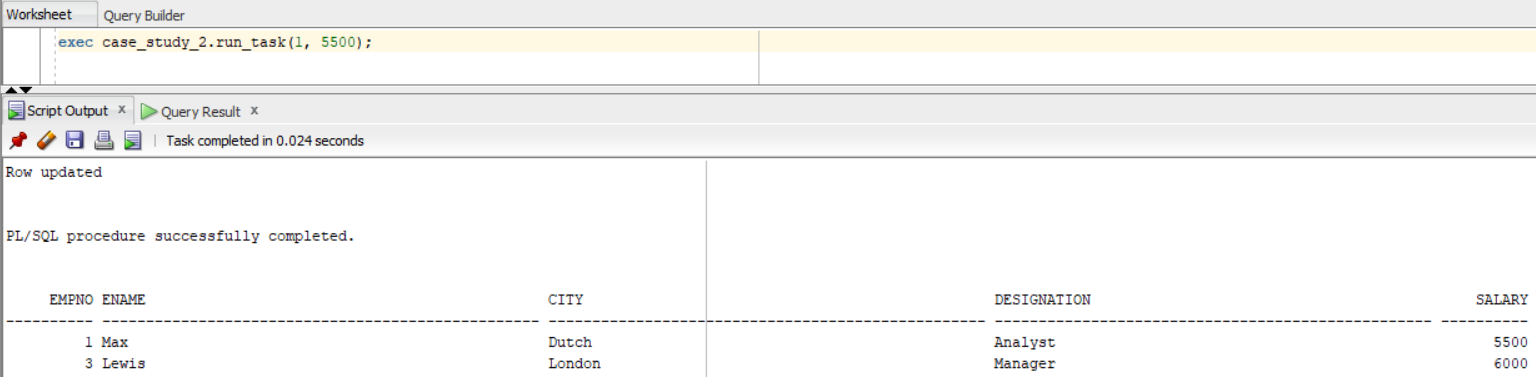
a) Test Case for Inserting a New Employee:

exec case\_study\_2.run\_task('John Doe', 50000, 'New York', 'Developer');



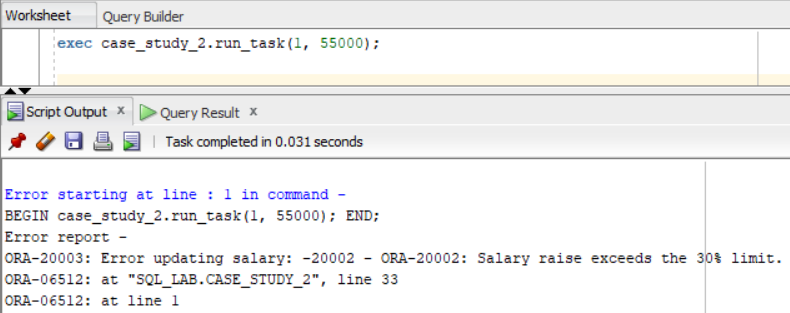
b) Test Case for Updating Employee Salary (within the limit):

exec case\_study\_2.run\_task(1, 55000);



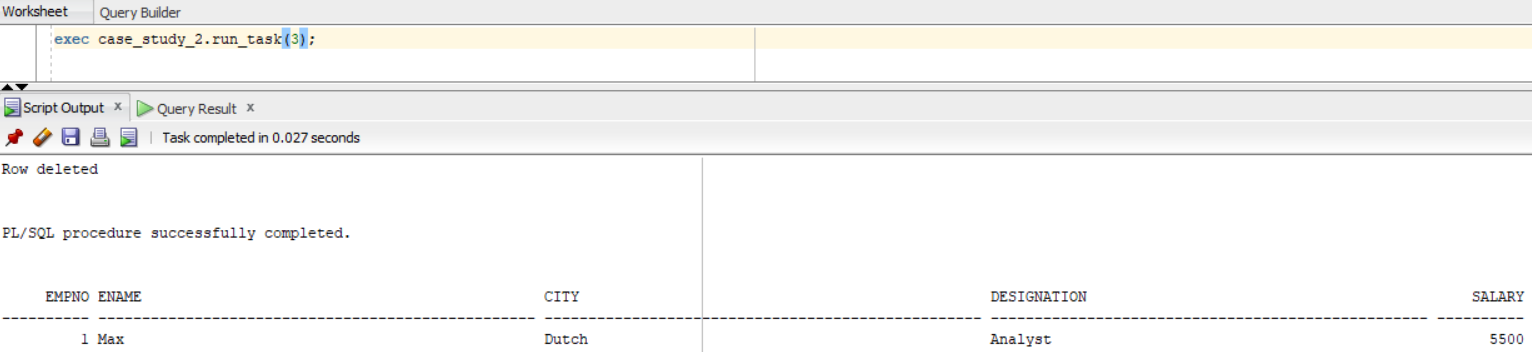
c) Test Case for Updating Employee Salary (exceeding the limit):

Exec case\_study\_2.run\_task(1, 70000);



d) Test Case for Deleting an Employee:

exec case\_study\_2.run\_task(3);



e) Test Case for Deleting Non-Existent Employee (no\_data\_found scenario):

exec case\_study\_2.run\_task(999);

