CST3504/3604 Lecture Notes

A Practical Approach to Database Development

Using Oracle Server Express Edition (Part 4)

CST3604 Lecture 1A (Part 1) – Stored Procedures & Functions

(Part 1 of 2)

(Lecture Notes 1A)

Prof. Abel Angel Rodriguez

Part 4 – Database Development in Oracle XE 11g Creating Stored Procedures & Functions

4.1 Introduction to Oracle Routines – Functions & Procedures (Stored Procedures)

4.1.1 Introduction Routines (Functions & Procedures) in Oracle

- Introduction:
  - A procedure:
    - A group of SQL statements & DBMS native code placed into a single block of code & compiled.
    - The Procedure is then a schama object that exists in the database.
    - Thereafter, the procedure can be called and executed as often as desired.

Functions & Procedures in typical programming languages (C#, Java, etc.)

- Routines in Oracle, have the same concept as **Functions** & **Procedures** in programing languages:
  - In typical programming languages like C#, JAVA, etc., **Functions** & **Procedures** are defined as follows:
    - **Function** – a block of code that returns one value & can have input parameters
    - **Procedures** – a block of code that DOES NOT return a value & can have input parameters

Implementing Functions & Procedures in Programming Languages

- In programming languages like C#, JAVA, etc., **Functions** & **Procedures** are implemented using two steps:

1. **Functions** & **Procedures** **DEFINITION** – The **Functions** or **Procedures** are declared and created.
2. **Functions** & **Procedures** **CALL/EXECUTION** – The **Functions** or **Procedures** are called or executed

STEP 1 – Function & Procedure Definition

- STEP 1 – Defining **Functions** & **Procedures** usually comes in 4 variations (I will use pseudo-code and not target any language):
  - **Procedure that DOES NOT return a VALUE (void) & WITH NO parameters:**

```
//Procedure with NO parameter & DOES NOT returns a value:
void ProcedureName ()
{
      //Body Code goes here!
}
```

  - **Procedure that DOES NOT return a VALUE (void) & WITH parameters:**

```
//Procedure WITH parameters & DOES NOT returns a value:
void ProcedureName (TYPE variable1, TYPE variable2, TYPE variable(n), … ) {

      //Body Code goes here!
}
```

- **Function that DOES return a VALUE (void) & WITH NO parameters:**

```
//Function with NO parameter & DOES return a value:
TYPE FunctionName (void)
{
      //Body Code goes here!
      Return value;
}
```

```
//Function WITH parameters & DOES return a value:
TYPE FunctionName (TYPE variable1, TYPE variable2, TYPE variable(n), … )
{
      //Body Code goes here!
      Return value;
}
```

STEP 2 – Function & Procedure Call or Execution

- STEP 2 – Calling or executing a **Functions** or **Procedures** also comes in 4 variations since there are 4 types of definitions:

  - For a **FUNCTION** that **RETURNS** a **VALUE** and has **NO ARGUMENTS**:

```
aVariable = FunctionName();
```

  - For a **FUNCTION** that **RETURNS** a **VALUE** and **HAS ARGUMENTS**:

```
aVariable = FunctionName(arg1, arg2, etc.);
```

  - For a **PROCEDURE** that **DON'T RETURN** a **VALUE** and takes **NO ARGUMENTS**:

```
ProcedureName();
```

  - For a **PROCEDURE** that **DON'T RETURN** a **VALUE** and **HAS ARGUMENTS**:

```
ProcedureName(arg1, arg2, etc.);
```

Routines in Oracle – Functions & Procedures

Overview of Routines in Oracle

- Like traditional programming languages, Oracle has **Functions** & **Procedures** but calls them **ROUTINES**.
- In Oracle, a **ROUTINE**:

  - A **ROUTINE** (Function & Procedure) is a group of **PROCEDURAL (processing statements/native DBMS code) & SQL STATEMENTS**, compiled into a single block of code, given a unique name.
  - Once created, the ROUTINE is a SCHEMA OBJECT that exist in the database and can be CALLED by name to ***EXECUTE***.
  - When called to EXECUTE, a **ROUTINE** executes ALL the **PROCEDURAL (processing statements/native DBMS code) & SQL STATEMENTS** will execute as a package.
  - A **ROUTINE** (Function & Procedure) can take parameters or information from the outside world into the **ROUTINE** for processing with the code.

- In ORACLE, a **ROUTINE** or comes in two flavors: **Functions** & **Procedures** which are defined as follows:

  1. **Function** – a block of code that **RETURNS** one value & can have **INPUT** parameters (same as programming languages)
  2. **Procedures** – a block of code that **DOES NOT RETURN** a value but can have **INPUT** parameters, **OUTPUT** parameters & combination of both **INPUT & OUTPUT** parameters.

  - Routines parameter classification description:

    - **INPUT** parameter – data sent into to the function or procedure from the outside world for processing.
    - **OUTPUT** parameter – (Procedures only) data sent out to the outside world from inside the procedure.
    - **INPUT & OUTPUT** parameter – (Procedures only) (has dual role) data sent into to the procedure from the outside world for processing or data can be sent out to the outside world from inside the procedure using the same parameter.

Implementing Routines (Function & Procedure) in Oracle

- Like traditional programming languages, Oracle **Functions** & **Procedures** are implemented using two steps:

  1. **Functions** & **Procedures DEFINITION** – The **Functions** or **Procedures** are declared.
  2. **COMPILE the Function or Procedure** – The **Functions** or **Procedures** is compiled and DATABASE OBJECT is created for the **Functions** or **Procedures** in the database
  3. **Functions** & **Procedures CALL/EXECUTION** – The **Functions** or **Procedures** are called or executed

- In the next sections, we will dive deeply into these two processes.

Usage or why use Routines (Function & Procedure)

- You create procedures & function to do the following:

  - Create a block of code that will **PERFORM AN OPERATION OR PROCESS OR ACTION ON THE DATA**.
  - You want to do something with the data, massage or process the data in some way.
  - The idea is you want to do something. Is not just execute a SELECT * FROM TABLE query. For example, calculate & give a 10% raise to all employees who live in the state of New York, etc. The idea is processing or action that will yield results.
  - **Functions** & **Procedures** contain procedural language components such as IF/ELSE, LOOPS, VARIABLES, ARRAYS, COLLECTIONS etc., for you to perform any required process.

Benefits of Routines

- **ROUTINES** (Functions & Procedures) have the following benefits:

  - **Reduces network traffic** – all statements are transmitted at one time instead of individually
  - **Security** – Security policies can be applied to **ROUTINE** (Function & Procedure). Users not authorized to call it, will get an error msg. Users authorized to call it, but may not have access to certain content will get an error. Therefore, you can apply security policies to control not only access to **ROUTINE**, but also access to the content being used by the **ROUTINE**.
  - **Flexibility** – Can be used in many situations. Many code options (Processing statements & SQL statements)
  - **Efficiency** – Can be designed, tweaked & optimized to run more quickly than general SQL Statements.
  - **Share-ability** – Can be cashed (stored in live memory) on the server and made available to all users.
  - **Applicability** – Can be stored as part of the entire DBMS thus available to all client applications. This feature aligns with Share-ability.
  - **Flexibility/Powerful** – Can be Stored procedures reside on the Database Server & provides a mechanism for SERVER SIDE PROCESSING

Approach to learning Routines in this document (Stored Procedure first, Function second)

- I will first focus on Procedures. Learn how to create, compile and execute them.
- Thereafter, we will focus on Functions.

4.1.2 Stored Procedures Definition and Call/Execution Syntax

- In this section, we cover the two steps to create and execute procedure routines:

Create Procedure Definition (Declare & Compile the Procedure)

Procedure Definition Syntax

- A simplified version of the syntax to create a procedure is shown below:

```
CREATE [OR REPLACE] PROCEDURE procedure_name [(Param1, Param2, Param3. Param(n))]

IS | AS

     <Variable_Declaration_section>;

BEGIN
     <Oracle PL/SQL statements/executable_section >;

END [procedure name];
```

- **CREATE** – keyword to create the procedure.
- **OR REPLACE** – [OPTIONAL] optional keyword used in addition to the keyword CREATE. Used to replace or re-create the procedure if it already exists.
- **PROCEDURE** – keyword indicating this is a procedure
- **Procedure_name** – name you give to the procedure or block of call. This will be the name used to call the procedure from the outside world.
- **`Parameter List(Param1, Param2, Param3. Param(n))`** – Optional list of parameters (within parentheses) passed into the procedure. This is optional since you can pass parameters or not. In addition, your parameter declarations:
    - **Parameter** declaration syntax:

```
ParemeterName IN|OUT|IN OUT OracleDataType
```

- **ParameterName** – name of parameter
- **Parameter Types** – There are 3 types of parameter types:
    - **IN**– **READ-ONLY** parameter where data of parameter sent into to the function or procedure from the outside world for processing can only be read by the procedures.
    - **OUT**– (***Usually Procedures only***) **WRITE-ONLY** parameter passed into the procedure whose objective is to be written with data from inside the procedure. The parameter populated within the procedure, can be used by the outside world. Usually these parameters are empty coming into the procedure and expected to be populated by the procedure for consumption in the outside world.
    - **IN OUT**– (***Usually Procedures only***) **READ-WRITE** (has dual role). Parameter values can be READ or WRITTEN by within the procedure.

- **OracleDataType** – Oracle Datatype such as VARCHAR, INT, DATE, etc:

  - **IMPORTANT CAUTION!** – when declaring the **OracleDataType** DON'T include the size limitations etc.
  - For example, if we declare the following parameter:

    CustID IN NUMBER(4)

  - This syntax **WILL NOT COMPILE**
  - The correct syntax:

    CustID IN NUMBER

- **IS | AS** – either of these keywords can be used to start the declaration section of the procedure. In this declaration section, you can declare variables and other declarations.
- **Variable_Declaration_section** – Optional section where variables and other declarations are listed:

  - **Variable** declaration syntax:

```
VariableName OracleDataType;
```

    - **VariableName** – name of variable
    - **OracleDataType** – Oracle Datatype such as VARCHAR, INT, DATE, etc.

    - **IMPORTANT!** – In the declaration section, you CAN INCLUDE the **OracleDataType** size limitations etc.
    - For example, you can declare the following variable with size information:

      CustID IN NUMBER(4)

- **BEGIN** – starts the executable section.
- **Oracle PL/SQL statements/executable_section** – in the BEGIN section, is where executable SQL & native code is declared. In this section, you can add the PL/SQL language code such as variables, control statements (if statements, loops, etc.) & SQL statements (SELECT, UPDATE, INSERT & DELETE). Below is the syntax for assigning data to the variables declared in the **IS/AS** section:

  - **Variable** assignment syntax:

```
VariableName := value;
```

    - **VariableName** – name of variable
    - **:=** – Assignment symbol to assign values to variables,
    - **value** – value being assigned to variable.
    - Example:

      CustID := 3333;

- **END** – ends the procedure
- **Procedure_name** – [OPTIONAL] name you give to the procedure or block of call repeated again for ease of identifying the procedures in its end when you have many procedures in a script.

- Note that this is a very simple syntax. The full syntax contains many more keywords for security and other parameters for more sophisticated operations.

- We can simply this further to align it to our comparison to traditional programming languages as follows:
  - **Stored Procedure (DOES NOT return a VALUE) WITH NO parameters:**

```
//Procedure with NO parameter & DOES NOT returns a value:
CREATE [OR REPLACE] PROCEDURE procedure_name
IS | AS

     <Variable_Declaration_section>;

BEGIN
     <Oracle PL/SQL statements/executable_section >;

END [procedure_name];
```

  - **Procedure (DOES NOT return a VALUE) WITH parameters:**

```
//Procedure WITH parameters & DOES NOT returns a value:
CREATE [OR REPLACE] PROCEDURE procedure_name [(Parameters_list)]

IS | AS

     <Variable_Declaration_section>;

BEGIN
     <Oracle PL/SQL statements/executable_section >;

END [procedure_name];
```

Create Procedure Call Syntax

Calling Stored Procedures Syntax

- The syntax to call a stored procedure depends on where you are calling the stored procedure from.
- Here are some common places stored procedures can be called from

1. **From Oracle SQL Developer or SQL Plus script** (not within a code block) – simply call the stored procedure from a script in area that is NOT defined as a block of code.
2. **From Oracle SQL Developer or SQL Plus script CODE BLOCK** – In a script, you can create/define a BLOCK OF CODE section & from within this BLOCK you can call a stored procedure
3. **From within CLIENT applications** – eventually this will be the #1 place where stored procedures will be called. Makes sense since applications is the front-end of a database application.
4. **From within another Stored Procedure** – You can also call stored procedures from within another stored procedure.
5. Other

- Below are two examples of crating procedures using the two syntax above

Scenario #1 – Calling Stored Procedure from Oracle SQL Developer/SQL+ NOT within a Code Block

- Now we look at how you call these stored procedures from an application or the SQL+ command console or Oracle SQL Developer in the editor window but NOT within a block of code.
- What we mean, NOT within a block of code is that you simply call the stored procedure outside a defined block of code

- For a **PROCEDURE** that **DON'T RETURN** a **VALUE** and takes **NO ARGUMENTS**:

```
EXECUTE ProcedureName();
```

```
EXEC ProcedureName();
```

- For a **PROCEDURE** that **DON'T RETURN** a **VALUE** and **HAS ARGUMENTS**:

```
EXECUTE ProcedureName(arg1, arg2, etc.);
```

```
EXEC ProcedureName(arg1, arg2, etc.);
```

Turning SERVER OUTPUT ON to view the output of a Stored Procedure in Oracle SQL Developer & SQL+ Console Windows

- Using stored procedures is different than running normal queries from Oracle SQL Developer & SQL+ Console Windows:

  - You simply cannot CALL the stored procedure and values appear in the message windows, like when you executed normal queries from your scripts.
  - If you wish to display the output of a stored procedure you need to TURN SERVER OUTPUT ON!
  - This is done using the following SYNTAX:

```
SET SERVEROUTPUT ON;
```

  - Don't think you need to do this every time you test, but it can be turned on once during the session.
  - Keep it in mind in case you don't see any results from your stored procedure execution, you may want to execute the statement above

Example 1 – Create a Stored Procedure that inserts a record into a hire_employee table

Procedure Definition/Declaration

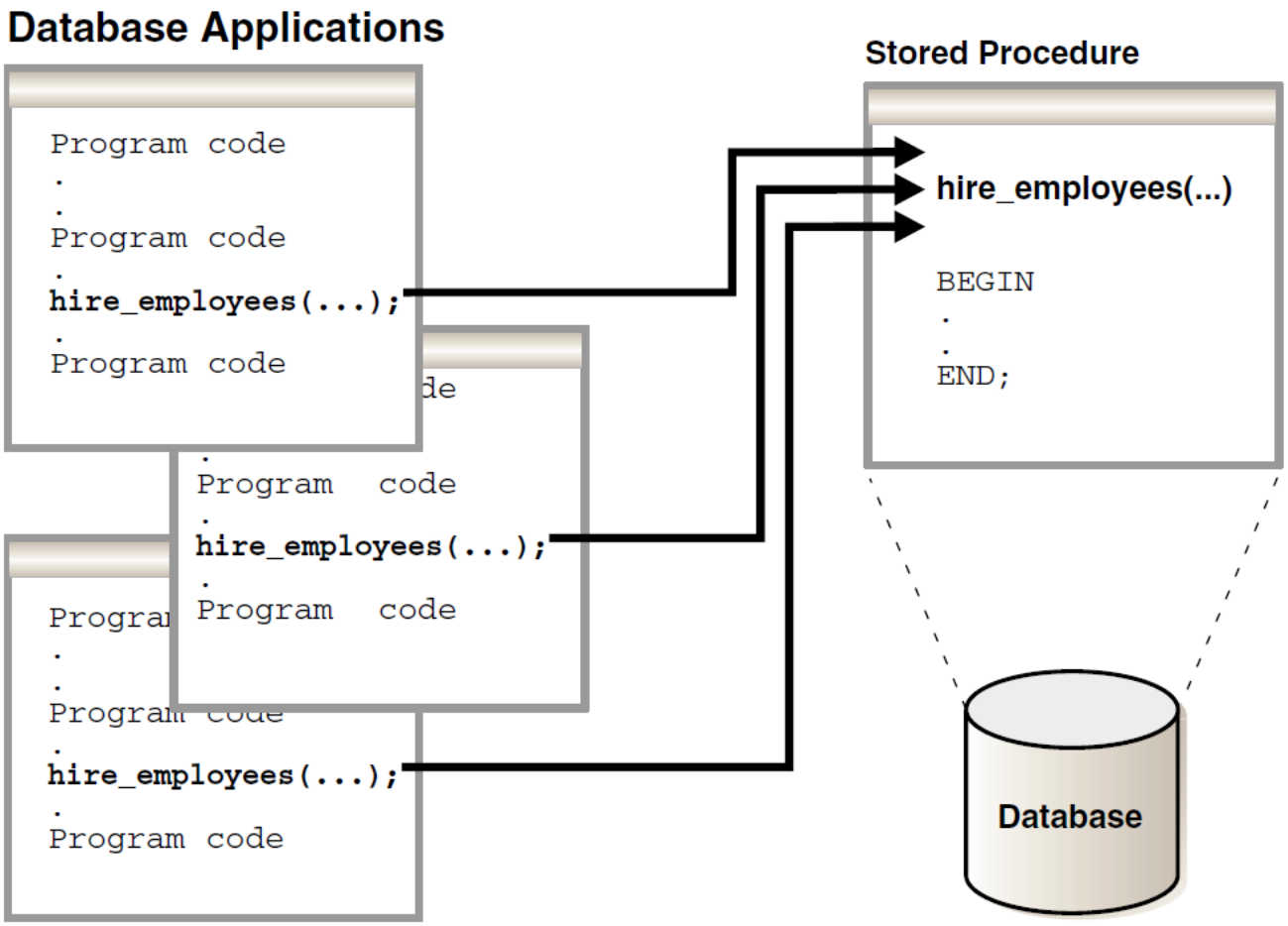
```
CREATE PROCEDURE hire_employees
  (p_last_name VARCHAR2, p_job_id VARCHAR2, p_manager_id NUMBER, p_hire_date DATE,
   p_salary NUMBER, p_commission_pct NUMBER, p_department_id NUMBER)
IS
BEGIN
.
.
.
  INSERT INTO employees (employee_id, last_name, job_id, manager_id, hire_date,
    salary, commission_pct, department_id)
  VALUES (emp_sequence.NEXTVAL, p_last_name, p_job_id, p_manager_id, p_hire_date,
    p_salary, p_commission_pct, p_department_id);
.
.
.
END;
```

Procedure Call (Assumed from SQL+ console or Oracle SQL Developer)

```
EXECUTE hire_employees ('TSMITH', 'CLERK', 1037, SYSDATE, 500, NULL, 20);
```

Procedure Call (Assumed from SQL+ console or Oracle SQL Developer)

- Calling procedure from the client application (database application) requires that the call is done from within the CLIENT CODE.
- No EXECUTE or EXEC keyword required.



4.1.3 Stored Procedures Examples using Oracle SQL Developer

- In this section, we show example of creating the stored procedure in Oracle SQL Developer console:

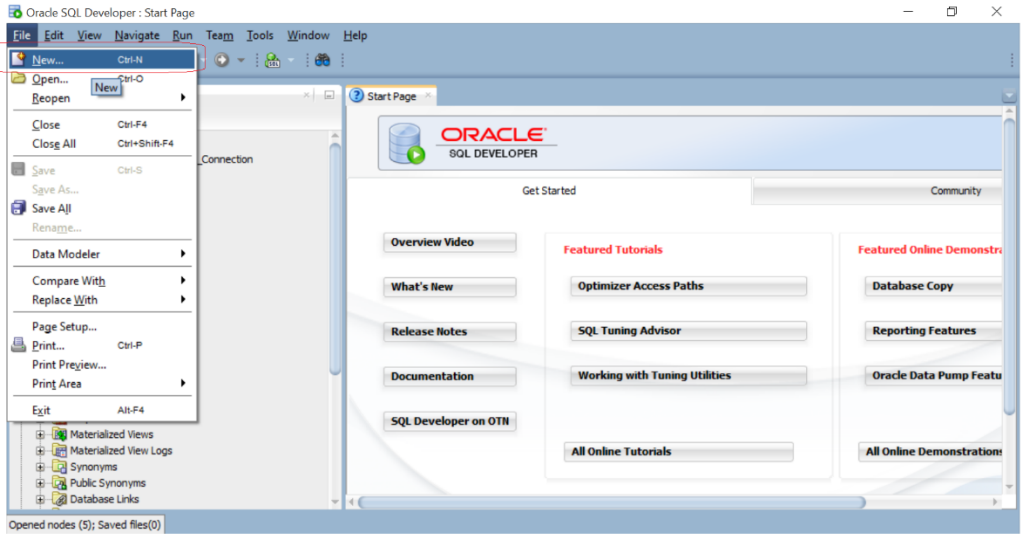
Step 1 - Prepare the Environment (For this course)

Create a separate Script File to Store all DML STORED PROCEDURES examples

- Let's create a separate script file to store all the STORED PROCEDURES used in this section. We CREATE PROCEDURES SCRIPT #2

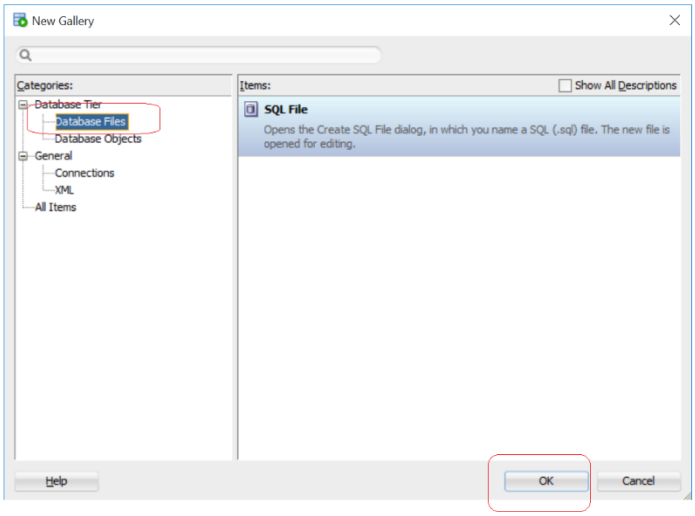
Step 1: Invoke the New Gallery Dialog Box

- Select:
  - Select **File|New** to invoke New Gallery Dialog Box



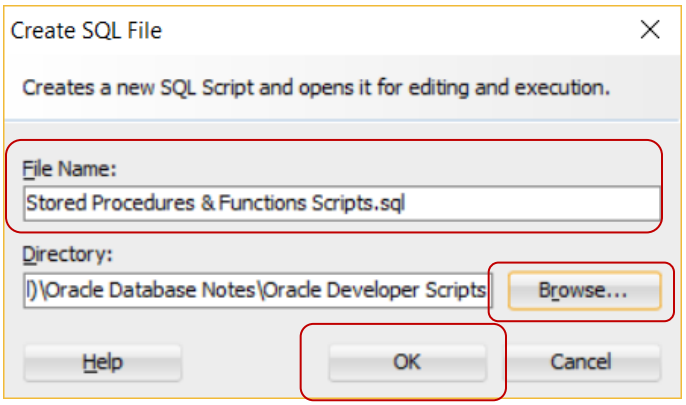
Step 2: In the New Gallery Dialog Box, select Database file and click OK

- Steps:



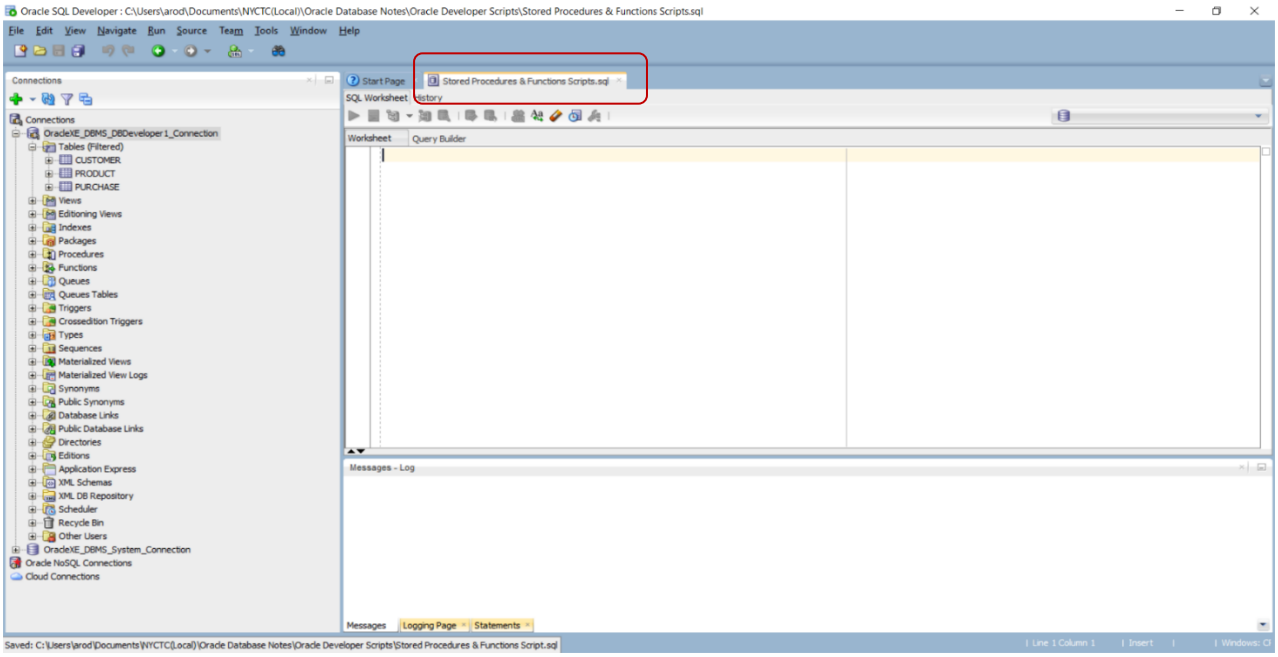
Step 3: In the Create SQL File Dialog Box, Enter File Name, Browse to folder location and click OK

- Steps:



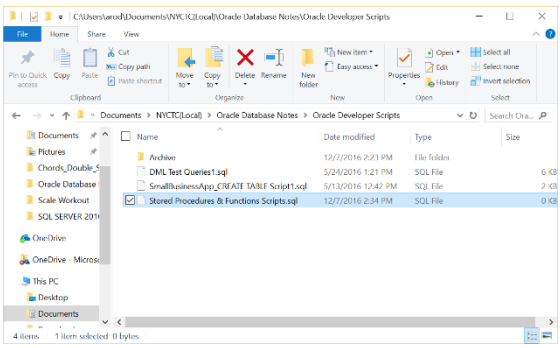
Step 4: The new Script is available in the SQL Worksheet for you to enter your Queries

- Steps:



Step 5: The file is also available in your computer file system

- Steps:



Examples Set # 1 – Simple examples to understand the basics Mechanism of Stored Procedures

Stored Procedure Example 1 – Stored Procedure that displays content of two variables Hardcoded inside the Stored Procedure

Target Code to Implement using Stored Procedure

- We start by simply creating a stored procedure that display content of two variables that are created and populated inside the stored procedure. The objective is to start getting used to variable creation, assignment and displaying their content.
- Target OUTPUT we wish stored procedure to provide:

```
--Stored Procedure to display the following messages
  - Variable1 = Hello World! And Variable2 = I love
    Summers!
```

Specification Implementation Steps

Step 1: Design the Stored Procedure Definition

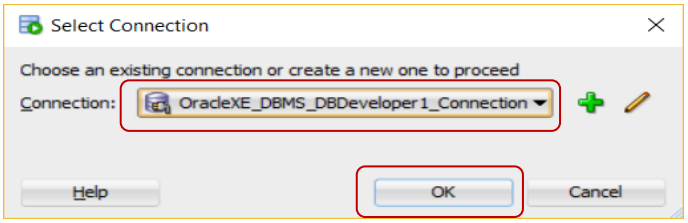
- Design:
  - Stored Procedure design is as follows:

```
-- Create Stored Procedure to display the content of 2 variables inside the
-- Stored Procedure.
-- Objecives - show creation of variables inside stored procedure & Displaying
-- content from a stored procedure
CREATE OR REPLACE PROCEDURE sp_DisplayVariableContent
IS  --Code declaration section

  v_Variable1  VARCHAR2(20);
  v_Variable2  VARCHAR2(20);

BEGIN  --Code execution section
  -- Assign values to variables
  v_Variable1:= 'Hello World!';
  v_Variable2:= 'I love Summers!';

-- Display to Oracle SQL Developer Message Window the content of variables
  DBMS_OUTPUT.PUT_LINE('Content of variable1 = '||v_Variable1|| ' and variable2 =
'||v_Variable2);

END sp_DisplayVariableContent;
```

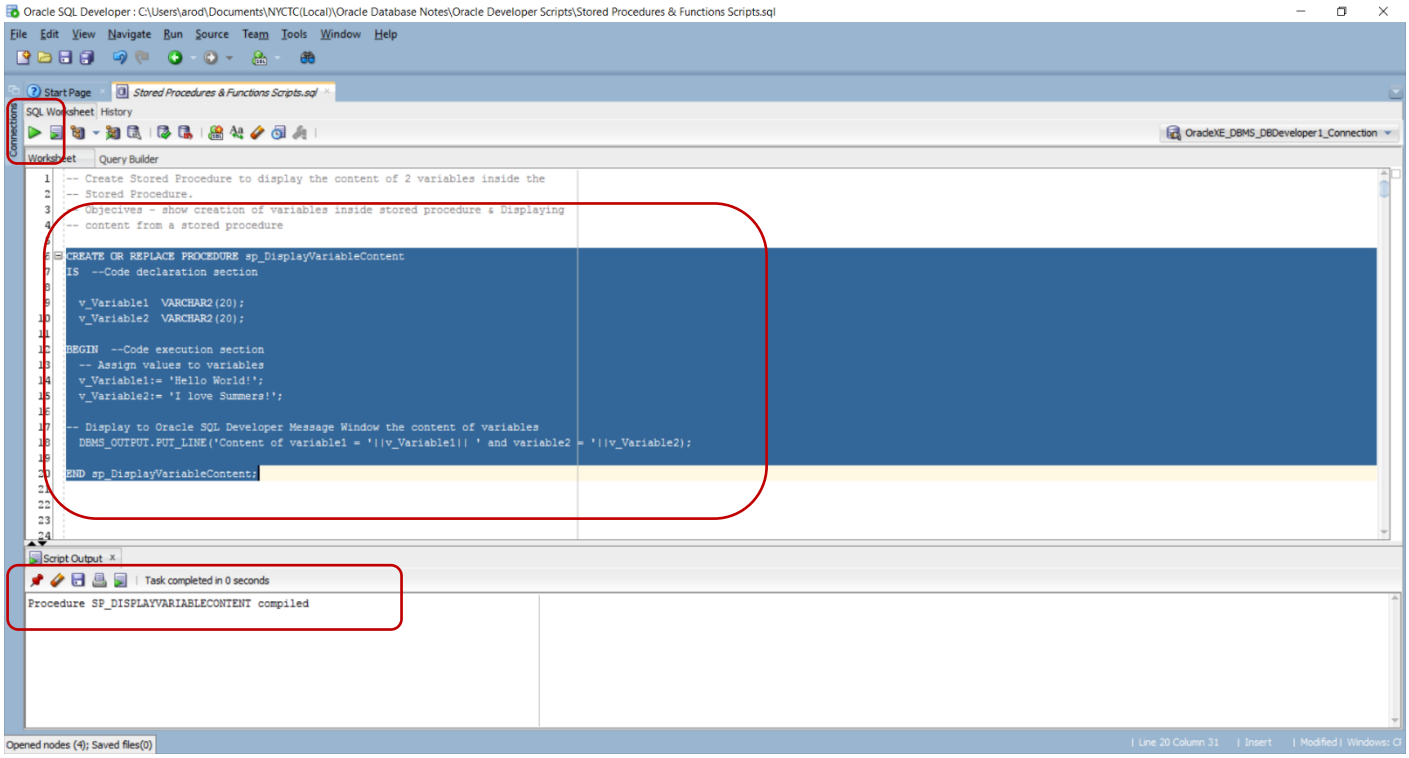
Step 2: In the Oracle SQL Developer Script Windows Enter the Code & Compile Stored Procedure

- In SQL Developer:
  - Select **File|Open** to navigate to the script file you want to enter code, or Select **File|New** if is a new script file & do the following:

    1) Enter code in script
    2) Execute/Compile the Script
    3) You may be prompted to select connection:



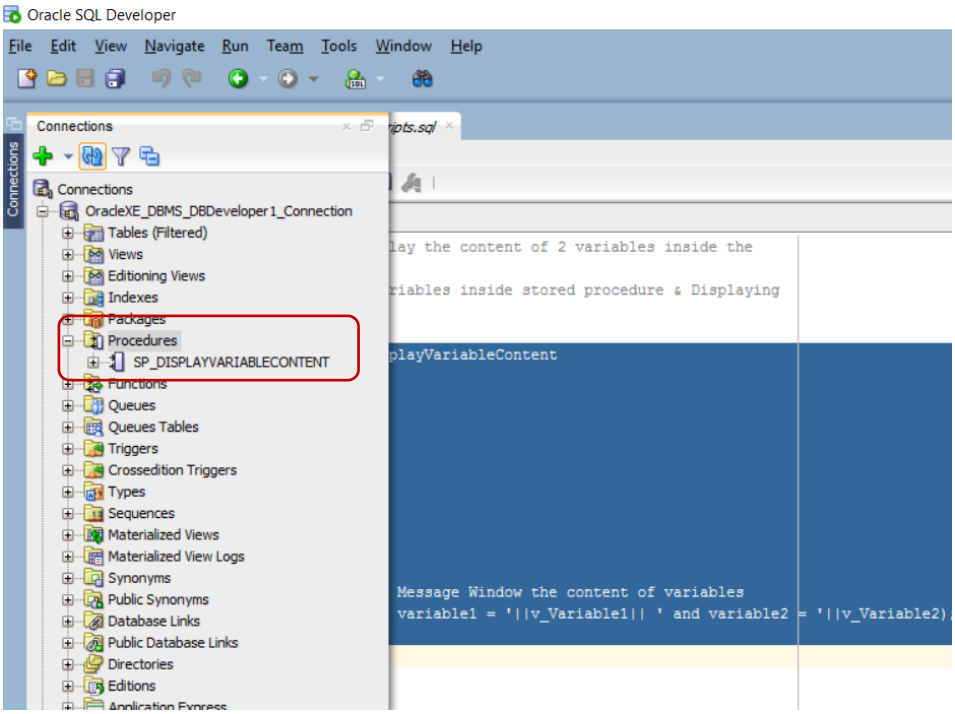
    4) Verify Procedure was successfully compiled



Step 3: In the Oracle SQL Developer Navigator Window, Verify Schema Object was created for Stored Procedure

- In SQL Developer:

1) In the Navigator Window Verify Schema Object exists



Step 4: In the Oracle SQL Developer Script Windows TEST the Stored Procedure by EXECUTING it

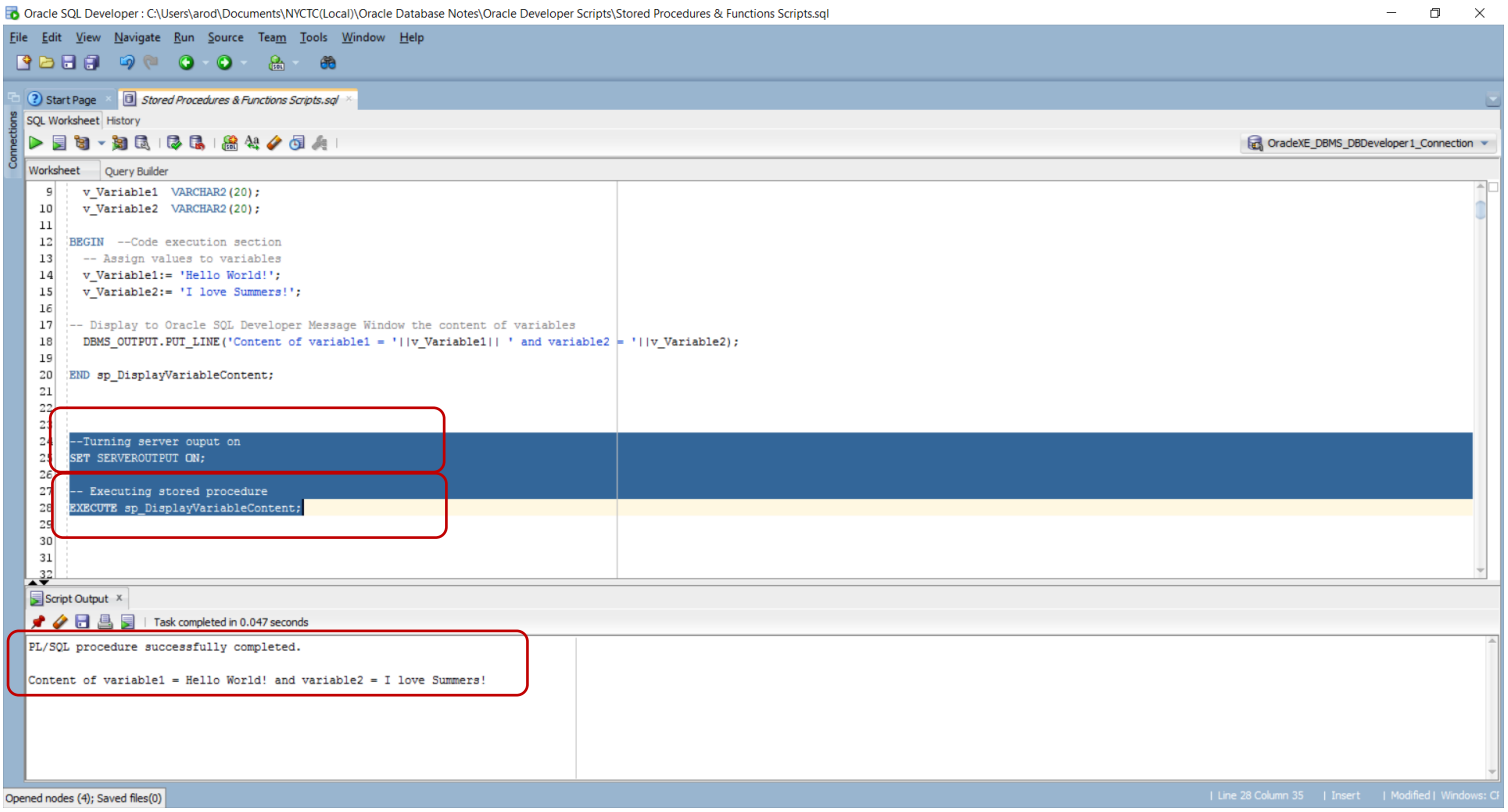
- In SQL Developer:
  - Execute as follows:

1) Since we are displaying data to the message window we need to **TURN SERVER OUTPUT ON**. Enter code to turn the server output on

2) Execute/Compile the Script using the following syntax

```
EXECUTE ProcedureName();
```

3) Message Window should indicate if execution was successful & show output of results



Stored Procedure Example 2 – Stored Procedure that displays content of two variables Passed as Parameters to Stored Procedure (Version 1)

Target Code to Implement using Stored Procedure

- Same as previous example, but this time will demonstrate how information can be passed as PARAMETERS to the stored procedure and used within the stored procedure.
- Target OUTPUT we wish stored procedure to provide as results:

```
--Stored Procedure to display the following messages
    - Variable1 = Hello World! And Variable2 = I love
      Summers!
```

Specification Implementation Steps

Step 1: Design the Stored Procedure Definition

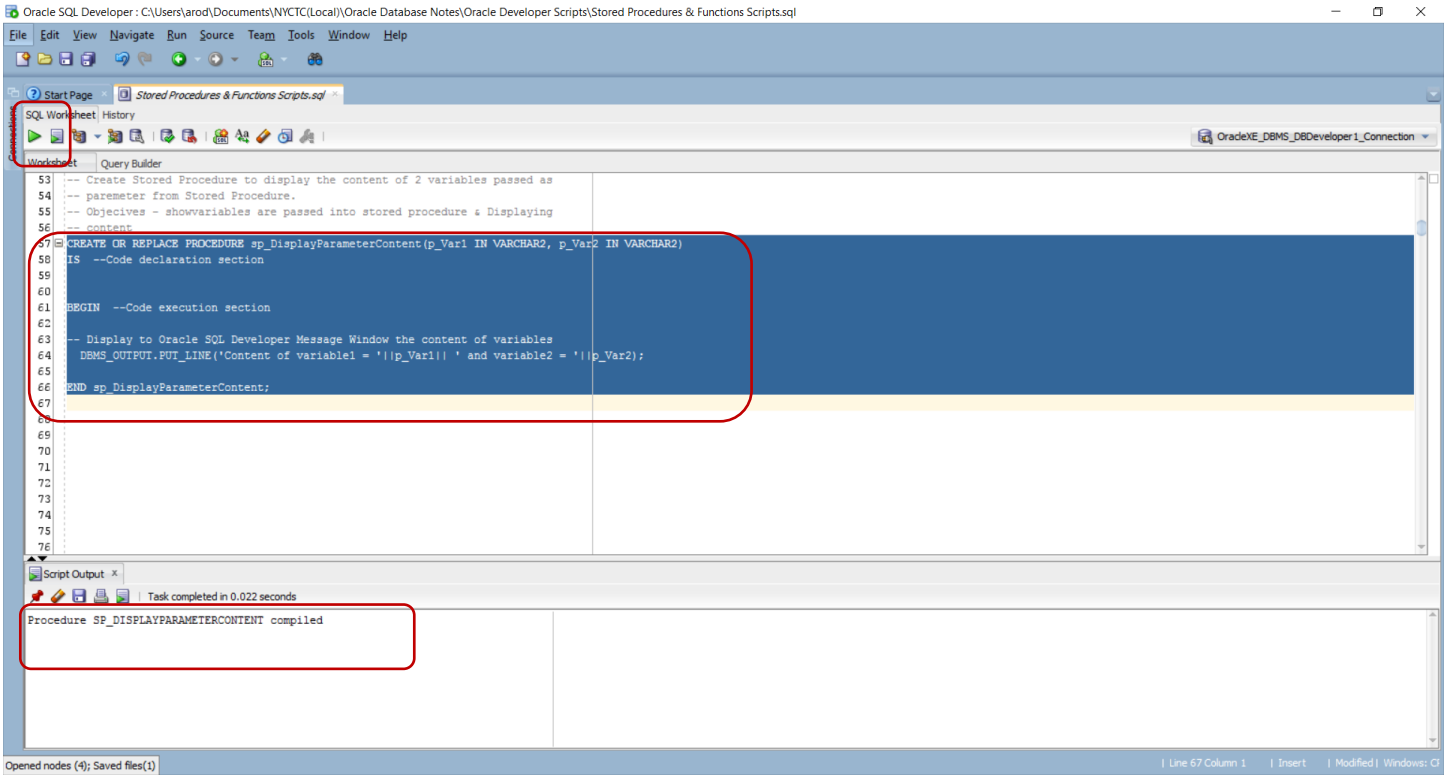
- Design:
  - Stored Procedure design is as follows:

```
-- Create Stored Procedure to display the content of 2 variables passed as
-- paremeter from Stored Procedure.
-- Objecives – show variables are passed into stored procedure & Displaying
-- content
CREATE OR REPLACE PROCEDURE sp_DisplayParameterContent(p_Var1 IN VARCHAR2, p_Var2 IN
VARCHAR2)
IS  --Code declaration section

BEGIN  --Code execution section

-- Display to Oracle SQL Developer Message Window the content of variables
  DBMS_OUTPUT.PUT_LINE('Content of variable1 = '||p_Var1|| ' and variable2 = '||p_Var2);

END sp_DisplayParameterContent;
```

Step 2: In the Oracle SQL Developer Script Windows Enter the Code & Compile Stored Procedure

- In SQL Developer:

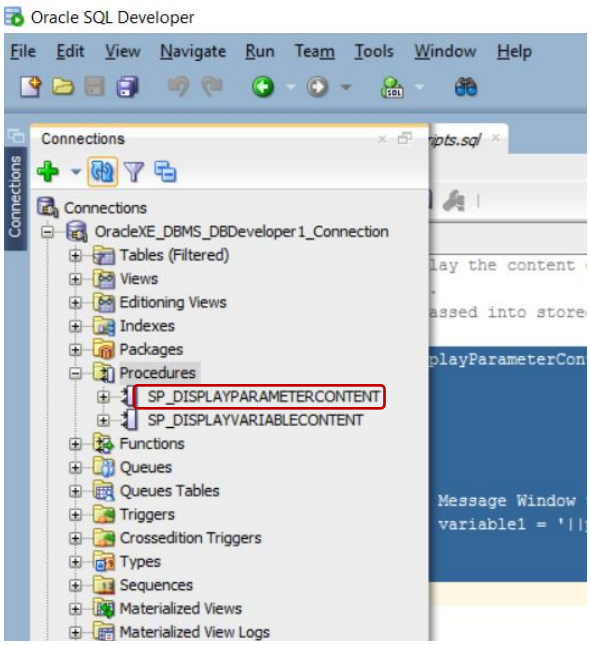
1) Enter code in script
2) Execute/Compile the Script
3) Verify Procedure was successfully compiled



Step 3: In the Oracle SQL Developer Navigator Window, Verify Schema Object was created for Stored Procedure

- In SQL Developer:

1) In the Navigator Window Verify Schema Object exists



Step 3: In the Oracle SQL Developer Script Windows TEST the Stored Procedure by EXECUTING it

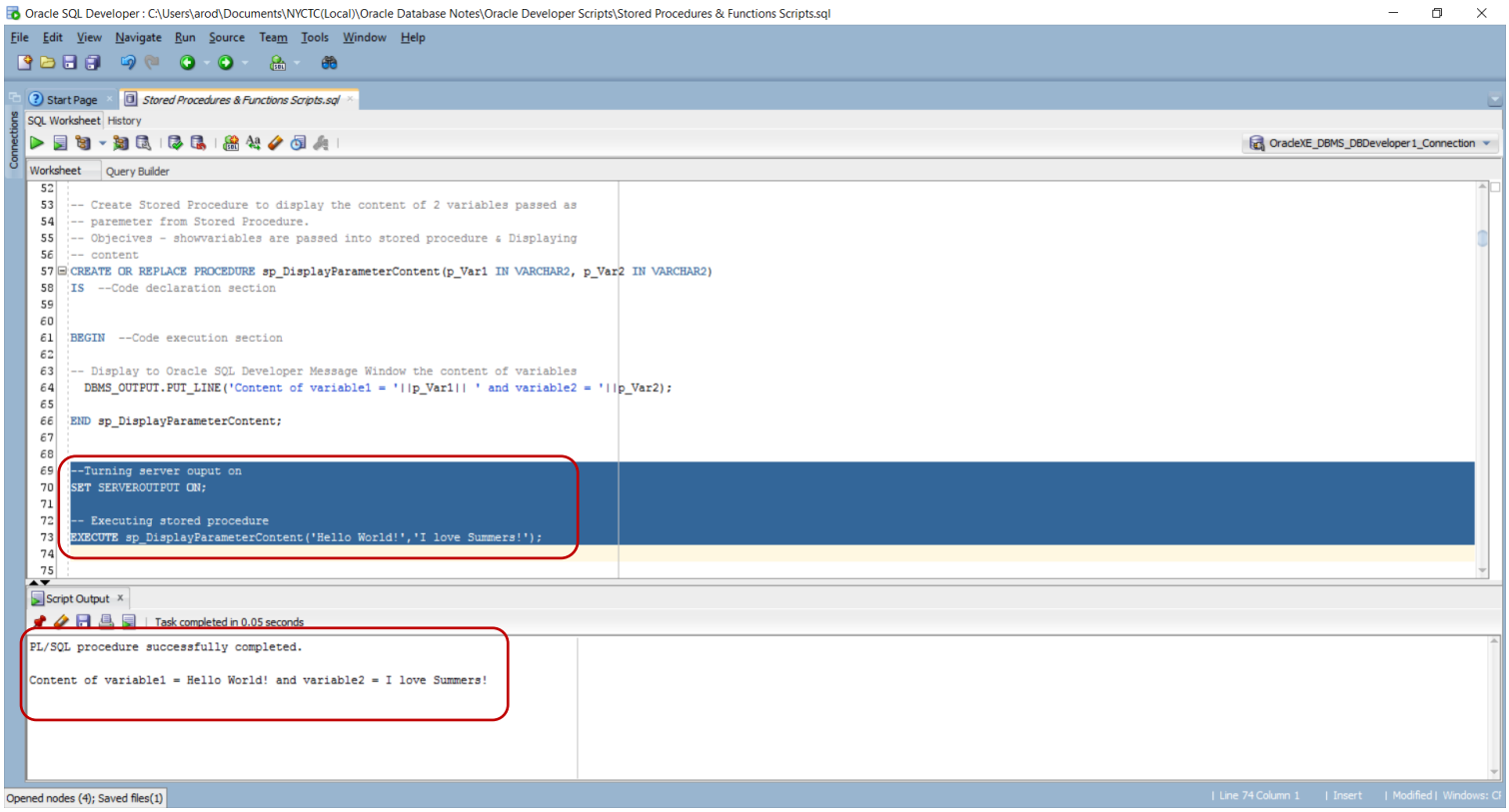
- In SQL Developer:
  - Execute as follows:

1) Since we are displaying data to the message window we need to **TURN SERVER OUTPUT ON**. Enter code to turn the server output on

2) Execute/Compile the Script using the following syntax

```
EXECUTE ProcedureName(arg1, arg2, etc.);
```

3) Message Window should indicate if execution was successful & show output of results



Stored Procedure Example 3 – Stored Procedure that displays content of two variables Passed as Parameters to Stored Procedure (Version 2)

Target Code to Implement using Stored Procedure

- Same as previous example 2, but this version, we will assign the parameters to variables created inside the stored procedure and then process/display the internal variables. The idea here is to show how parameters can be used in many ways.
- Target OUTPUT we wish stored procedure to provide as results:

```
--Stored Procedure to display the following messages
  - Variable1 = Hello World! And Variable2 = I love
    Summers!
```

Specification Implementation Steps

Step 1: Design the Stored Procedure Definition

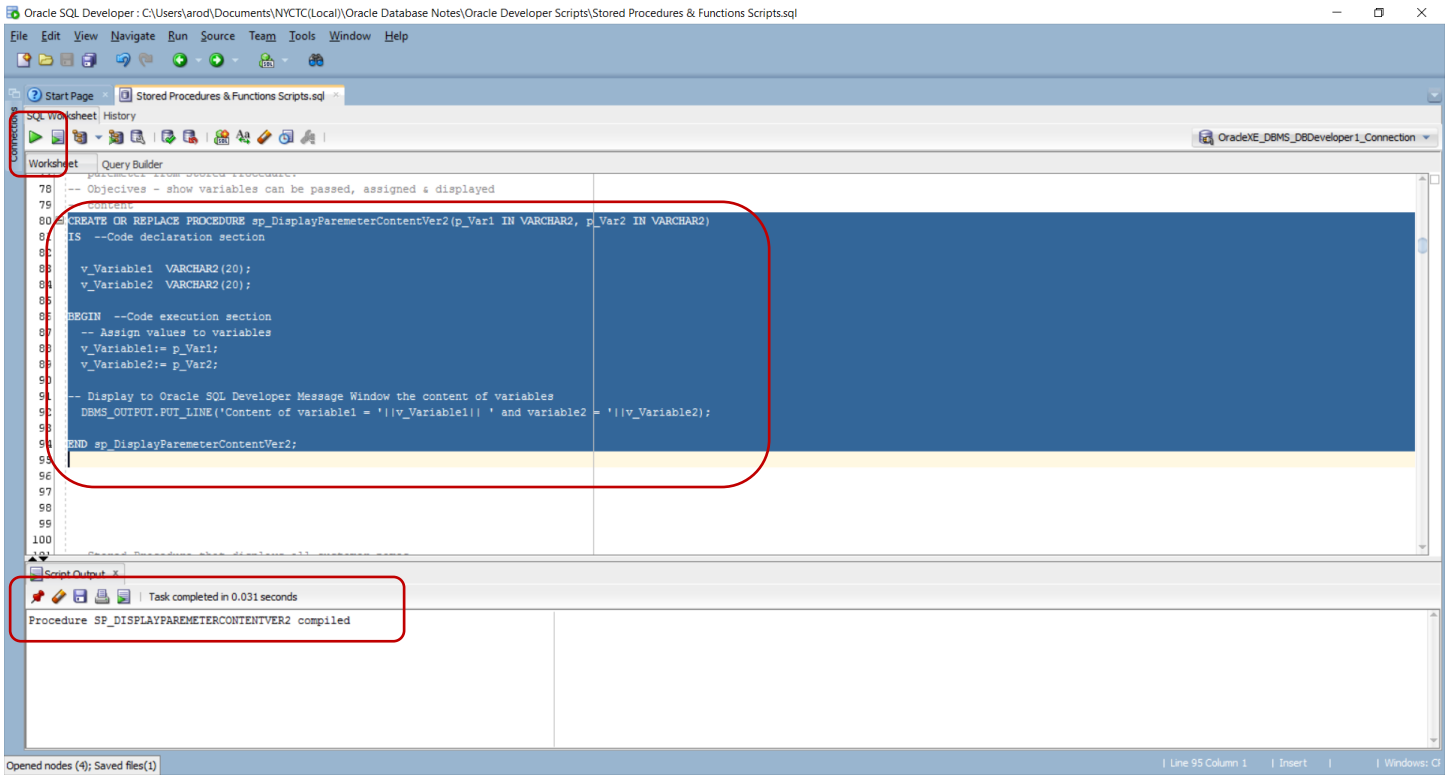
- Design:
  - Stored Procedure design is as follows:

```
-- Create Stored Procedure to display the content of 2 variables passed as
-- paremeter from Stored Procedure.
-- Objecives – show variables are passed into stored procedure & Displaying
-- content
CREATE OR REPLACE PROCEDURE sp_DisplayParemeterContentVer2(p_Var1 IN VARCHAR2, p_Var2 IN
VARCHAR2)
IS  --Code declaration section
  v_Variable1  VARCHAR2(20);
  v_Variable2  VARCHAR2(20);

BEGIN  --Code execution section
  -- Assign values to variables
  v_Variable1:= p_Var1;
  v_Variable2:= p_Var2;

  -- Display to Oracle SQL Developer Message Window the content of variables
  DBMS_OUTPUT.PUT_LINE('Content of variable1 = '||v_Variable1|| ' and variable2 = '||
v_Variable2);

END sp_DisplayParemeterContentVer2;
```

Step 2: In the Oracle SQL Developer Script Windows Enter the Code & Compile Stored Procedure

- In SQL Developer:

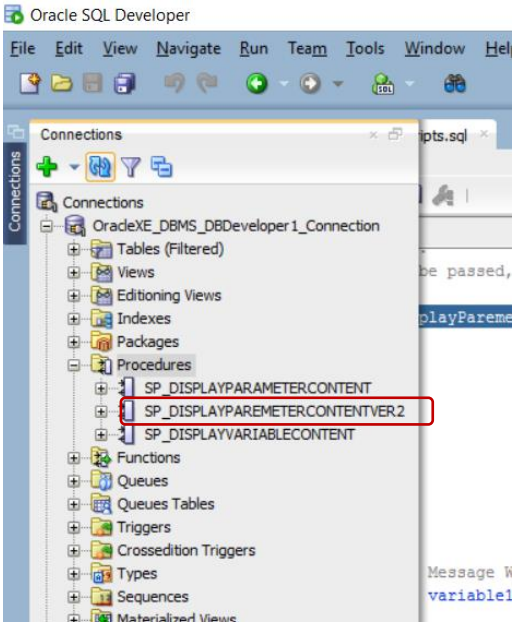
1) Enter code in script
2) Execute/Compile the Script
3) Verify Procedure was successfully compiled



Step 3: In the Oracle SQL Developer Navigator Window, Verify Schema Object was created for Stored Procedure

- In SQL Developer:

1) In the Navigator Window Verify Schema Object exists



Step 3: In the Oracle SQL Developer Script Windows TEST the Stored Procedure by EXECUTING it

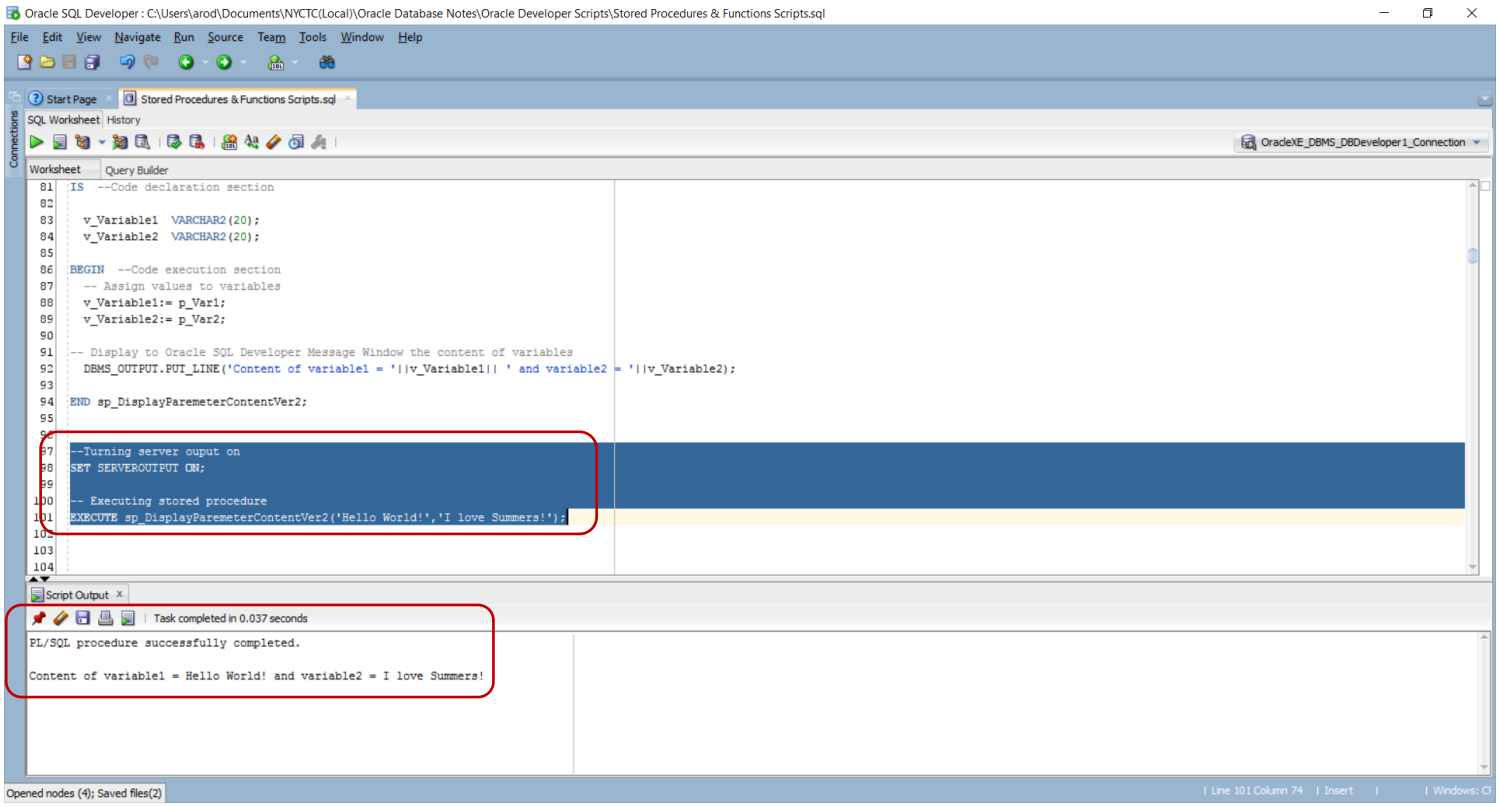
- In SQL Developer:
  - Execute as follows:

1) Since we are displaying data to the message window we need to **TURN SERVER OUTPUT ON**. Enter code to turn the server output on

2) Execute/Compile the Script using the following syntax

```
EXECUTE ProcedureName(arg1, arg2, etc.);
```

3) Message Window should indicate if execution was successful & show output of results



4.1.4 Creating Stored Procedures for Action Queries – Update, Insert & Delete Queries

- Now we focus on the exciting topic of creating stored procedures that execute queries.
- We begin with stored procedures that execute UPDATE, INSERT & DELETE queries.
- These types of stored procedures are straight forward and easy to create, compared to SELECT statements. I will explain in future section on SELECT statements.

SECTION TO BE COMPLETED IN FUTURE UPDATE

Examples Set # 2 – Creating Stored Procedures that execute action queries (INSERT, UPDATE & DELETE)

Example 2 – Create a Stored Procedure that inserts a record into a hire_employee table

Procedure Definition/Declaration

Example 3– Create a Stored Procedure that Updates a record in a table

Procedure Definition/Declaration

Example 4 – Create a Stored Procedure that Delete a record from table

Procedure Definition/Declaration

4.2 Stored Procedure that execute **SELECT SQL Statements** – A special case

4.2.1 Introduction to Stored Procedures that execute SELECT Statements

- SELECT queries/statements are **unique** compared to update, insert & delete statements as follows:
  - A **SELECT** query can **RETURNS ONE or MORE records (rows)**. Compared to insert, update & delete which just modified the records and returned no values. This will be a challenge in PL/SQL
  - **IMPORTANT**! – In Oracle, you **CANNOT** use a regular SELECT statements inside Stored Procedures, you **MUST USE** a **SELECT INTO** statement.
  - **IMPORTANT**! – **A SELECT INTO** statement **CAN ONLY RETURN ONE ROW**. This means that if you wish to execute a query that return multiple rows **YOU CANNOT DO IT.**
  - **IMPORTANT**! – A **SELECT INTO** statement **MUST PLACE THE COLUMN VALUES RETURNED FROM THE ONE ROW INTO VARIABLES**. Therefore, you need to ***CREATE VARIABLES*** inside your Stored Procedures to hold the values in the columns returned from a SELECT INTO statement.
  - **IMPORTANT**! – The CONTENT of the VARIABLES resulting from the **SELECT INTO** statement, **MUST BE CONSUMED OR DISPLAYED AS NEEDED USING PL/SQL LANGUAGE COMPONENTS**. They will not automatically appear as they do when you execute a select statement under normal circunstances. You must handle or display using PL/SQL code.
  - **IMPORTANT**! – If you need a **SELECT** statement inside a stored procedure, that return **MULTIPLE-ROWS**, which most likely you will, the way to do this in Oracle is to use the **CURSOR** mechanism & other **PL/SQL language components** to do so.
- Based on the statements above we come to the following conclusions/rules:
  1. For **SELECT** queries we need to use **SELECT INTO** in our stored procedures.
  2. **SELECT INTO** can only return **ONE ROW**.
  3. We MUST create **VARIABLES** in our stored procedure to store the column values returned from the **SELECT INTO** query. We can then manipulate these **variables** to get the data and process it (display, calculate, etc.) using PL/SQL code.
  4. To create stored procedures that contain **SELECT** statements that **RETURN MULTIPLE ROWS**, we need other **PL/SQL components such as CURSORS, LOOPS, IF STATEMENTS etc**. Is a more complex scenario.
- In the following sections, we will address SELECT statements as follows:
  - SELECT STATEMENTS that ONLY RETURN ONE ROW – we will work with simple select statements that return one row so we can understand how to use select into.
  - SELECT STATEMENTS that RETURN MULTIPLE ROWS – we will then address the more complex scenarios of using CURSORS and other PL/SQL language components to create SELECT statements that return multiple records.

4.2.2 SELECT INTO and Stored Procedures

- Here are the rules we need to follow to work with SELECT queries inside stored procedures:

1. SELECT statements cannot be used inside stored procedure, YOU MUST use **SELECT INTO** statement.
2. SELECT INTO statements can ONLY RETURN one record.
3. You will need to create Variables for results of SELECT INTO statement.
4. You will need PL/SQL components to process the content of the variables.
5. IF MORE THAN ONE RECORD IS REQUIRED, YOU NEED TO USE CURSORS & OTHER PL/SQL language components, which will be addressed in next section.

Select statements that return MORE THAN ONE RECORD CANNOT BE EXECUTED inside STORED PROCEDURE Whether SELECT INTO is USED OR NOT.

- This query you CANNOT execute within stored procedure because it returns more than one row:

  - Assuming we have the following table with data:

CUSTOMER_ID	NAME	BDATE	ADDRESS	PHONE	GENDER	EMAIL
1111	Joe Smith	01-JAN-71	111 Jay Street, Brooklyn NY 11201	718 111-1111	M	jsmith@xyz.com
2222	Angel Rodriguez	02-FEB-72	222 Flatbush Avenue, Brooklyn NY 11202	718 222-2222	M	arod@xyz.com
3333	Mary Jones	03-MAR-73	333 Flatlands Avenue, Brooklyn NY 11203	718 333-3333	F	mjones@xyz.com
4444	Samuel Adams	04-APR-74	444 Park Avenue, New York NY 11204	212 444-4444	M	(null)
5555	Nancy Rivera	05-MAY-75	555 Metropolitan Avenue, Brooklyn NY 11205	718 555-5555	F	nrivera@xyz.com

  - This query can be executed from the worksheet window in Oracle SQL Developer or SQL Plus and it will return the desire number of rows as shown below:

```
--Select Query with WHERE clause
SELECT Name
FROM CUSTOMER;
```

  - Results:

NAME
Josephine Smith (Jo)
Angel Rodriguez
Mary Jones
Nancy Rivera
Frank Hum

- But within a stored procedure, the following SELECT statement **CANNOT BE COMPILED**:

```
CREATE OR REPLACE PROCEDURE sp_GetCustomerNames
IS  --Code declaration section


BEGIN  --Code execution section

    --Get customer names
    SELECT Name
    FROM CUSTOMER;

END sp_GetCustomerNames;
```

- The first reason this stored procedure would NOT WORK is:

1. Because we **DID NOT** use **SELECT INTO** statement.
2. Nevertheless, even if we use **SELECT INTO** statement we CANNOT execute the stored procedure if the query returns **MORE THAN ONE ROW**. For example, the following stored procedure WILL NOT WORK:

```
CREATE OR REPLACE PROCEDURE sp_GetCustomerNames
IS  --Code declaration section

     --Variable to store result from Name column
  Cust_Name  VARCHAR2(50);

BEGIN  --Code execution section

    --Get customer name & place into variable
    SELECT Name INTO Cust_Name
    FROM CUSTOMER;

END sp_GetCustomerNames;
```

- So even if we use SELECT INTO statement inside the stored procedure, if the query returns more than one row, it WILL NOT WORK/COMPILE because multiple rows are being returned.
- To implement a **SELECT STATEMENT** in a **STORED PROCEDURE** these rules must be met:

1. Standard **SELECT STATEMENT** ***CANNOT*** be used inside stored procedure, YOU MUST use **SELECT INTO STATEMENT**.
2. **SELECT INTO STATEMENT MUST ONLY RETURN RECORD**.
3. IF MORE THAN ONE RECORD IS REQUIRED, YOU NEED TO USE CURSORS & OTHER PL/SQL language components, which will be addressed in next section.

Examples Set # 3 – Examples using Select Into statements within Stored Procedures to return ONE ROW

Attention! Use SELECT INTO Inside Stored Procedures for Queries that ALWAYS RETURN ONE RECORD ONLY!

Creating Stored Procedure with SELECT INTO statements that return ONE ROW ONLY

- **If your SELECT INTO returns ONE ROW, then we can work with it within a stored procedure**.
- To work with SELECT INTO statements that return ONE ROW you need to do the following:

1. Inside stored procedure, create a VARIABLE (S) to store the result of the column being returned from the ONE ROW.
2. Create the SELECT INTO statement and assign the VARIABLE(S) to store the value of the columns or resultant row.
3. Create the SELECT INTO statement using WHERE clause to limit results to ONE ROW
4. Use PL/SQL language components to process the results in the VARIABLE(S) as needed. If you wish to display the content of the variables use the PL/SQL**)** component to display the variable content to screen.

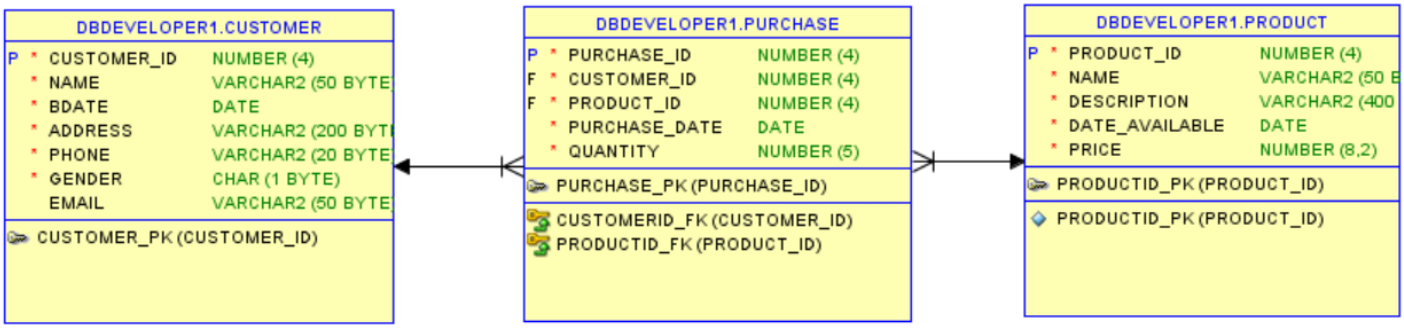
```
DBMS_OUTPUT.PUT_LINE(||variable1||, ||variable2||, ||variable(n)||);
```

- Let's look at some examples.

Schema used to demonstrate stored procedures

- To demonstrate stored procedure we will use the following schema:

Schema (Tables & Relationships)



Data Currently in Tables

- We also assume we have the following data in the **Customer Table**:

CUSTOMER_ID	NAME	BDATE	ADDRESS	PHONE	GENDER	EMAIL
1111	Josephine Smith (Jo)	01-JAN-81	111 Glendwood Road, Brooklyn NY 11210	718-282-1111	F	josephine.smith@comp.com
2222	Angel Rodriguez	02-FEB-72	222 Park Avenue, New York, NY 10030	212 222-2222	M	arod@xyz.com
3333	Mary Jones	03-MAR-73	333 Flatlands Avenue, Brooklyn NY 11203	718 333-3333	F	mjones@xyz.com
5555	Nancy Rivera	05-MAY-75	555 Metropolitan Avenue, Brooklyn NY 11205	718 555-5555	F	nrivera@xyz.com
6666	Frank Hum	04-JUN-76	666 74th Street, Flushing NY 1134	718 666-6666	M	fhum@comp.com

- We also assume we have the following data in the **Product Table**:

PRODUCT_ID	NAME	DESCRIPTION	DATE_AVAILABLE	PRICE
1111	Galaxy HD TV	High Definition TV	01-JAN-01	1000
2222	Tornato Washer Delux	High Power Washing Machine	02-FEB-02	2000
3333	Tornato Dryer Delux	High Power Dryer Machine	03-FEB-03	2000
4444	StarBright Computer	Laptop Computer	04-MAR-04	1500
5555	Galaxy Deluxe Freezer	Refrigerator	05-MAY-05	499.95

- Finally, we assume we have the following data in the **Purchase Table**:

PURCHASE_ID	CUSTOMER_ID	PRODUCT_ID	PURCHASE_DATE	QUANTITY
1	2222	2222	10-MAY-10	1
2	2222	3333	10-MAY-10	1
3	1111	1111	04-JUL-10	2
4	5555	4444	24-DEC-15	4

Example 1 – Stored Procedure that displays the name of customer 3333 (Not a practical scenario – THIS EXAMPLE WILL NOT BE USED IN THE REAL WORLD – FOR TEACHING PURPOSE ONLY!)

Target Code to Implement using Stored Procedure

- We start by simply creating a stored procedure that display the name of the customer whose ID = 3333.
- Target OUTPUT we wish stored procedure to provide & SELECT query that will facilitate:

```
--Select Query with WHERE clause
SELECT Name
FROM CUSTOMER
WHERE Customer_ID = 3333;
```

```
--Stored Procedure to display the following messages
     - Name of customer 3333 is Mary Jones!
```

- Of course, such query that always returns the name of customer 3333 is NOT PRACTICAL. Unless customer 3333 is someone very important in which a business must create a process just for them, hin/her is very rare.
- This is purely for teaching purpose.
- Normally such query should be universal or return the name of ANY customer, not just 3333.

Specification Implementation Steps

Step 1: Design the Stored Procedure Definition

- Design:
  - First to implement we would need the following:
    - SELECT INTO statement & WHERE clause to return only one row.
    - Variables to store the results of select into
    - PL/SQL language components to display the results.

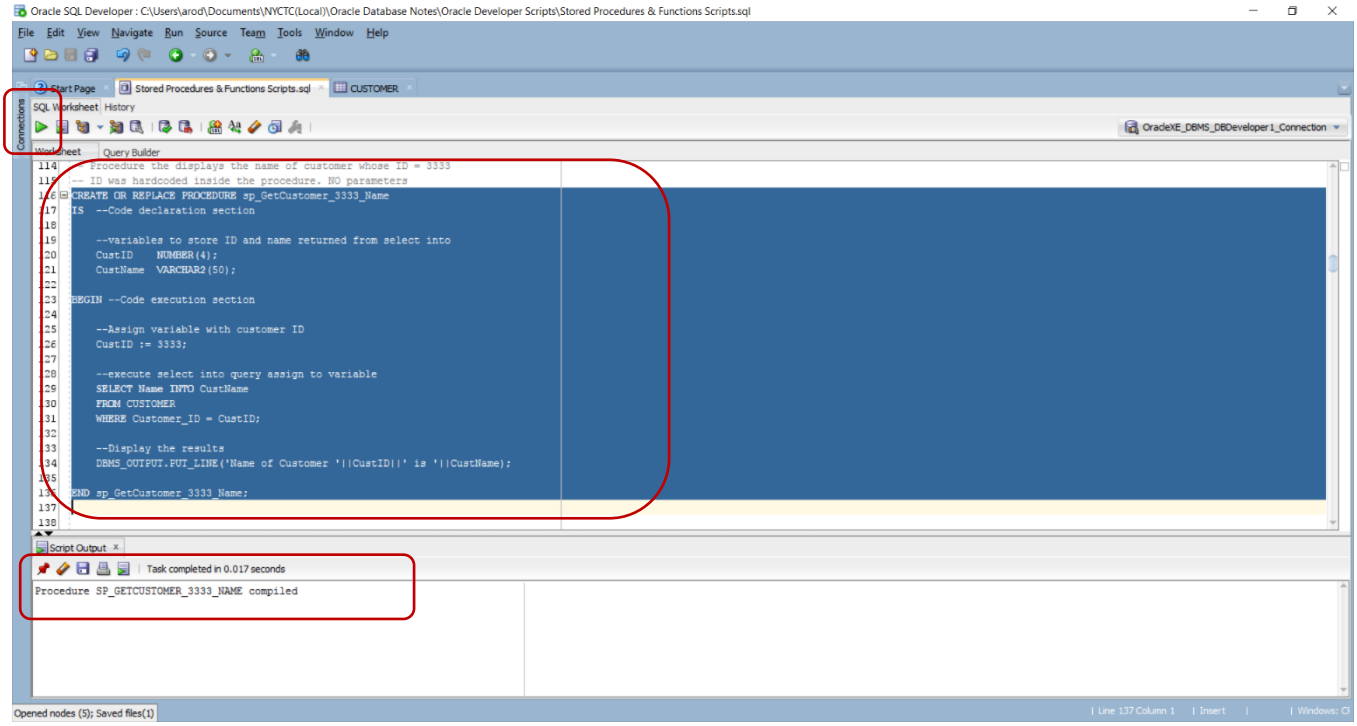
```
-- Procedure the displays the name of customer whose ID = 3333
-- ID was hardcoded inside the procedure. NO parameters
CREATE OR REPLACE PROCEDURE usp_GetCustomer_3333_Name
IS  --Code declaration section

    --variables to store ID and name returned from select into
    vCustID    NUMBER(4);
    vCustName  VARCHAR2(50);

BEGIN --Code execution section
    --Assign variable with customer ID
    CustID := 3333;

    --execute select into query assign to variable
    SELECT Name INTO vCustName
    FROM CUSTOMER
    WHERE Customer_ID = vCustID;

    --Display the results
    DBMS_OUTPUT.PUT_LINE('Name of Customer '||vCustID||' is '||vCustName);

END sp_GetCustomer_3333_Name;
```

Step 2: In the Oracle SQL Developer Script Windows Enter the Code & Compile Stored Procedure

- In SQL Developer:

  - Select **File|Open** to navigate to the script file you want to enter code, or Select **File|New** if is a new script file & do the following:

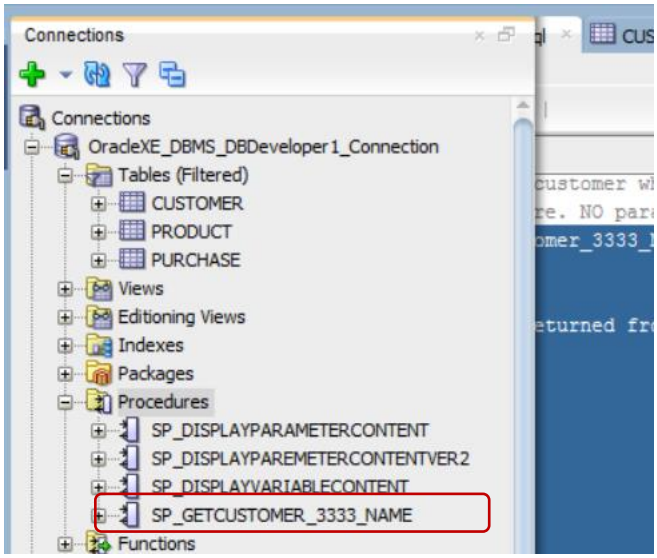
    1) Enter code in script
    2) Execute/Compile the Script
    3) Verify Procedure was successfully compiled



Step 3: In the Oracle SQL Developer Navigator Window, Verify Schema Object was created for Stored Procedure

- In SQL Developer:

  1) In the Navigator Window Verify Schema Object exists



Step 4: In the Oracle SQL Developer Script Windows TEST the Stored Procedure by EXECUTING it

- In SQL Developer:
  - Execute as follows:

1) Since we are displaying data to the message window we need to **TURN SERVER OUTPUT ON**. Enter code to turn the server output on

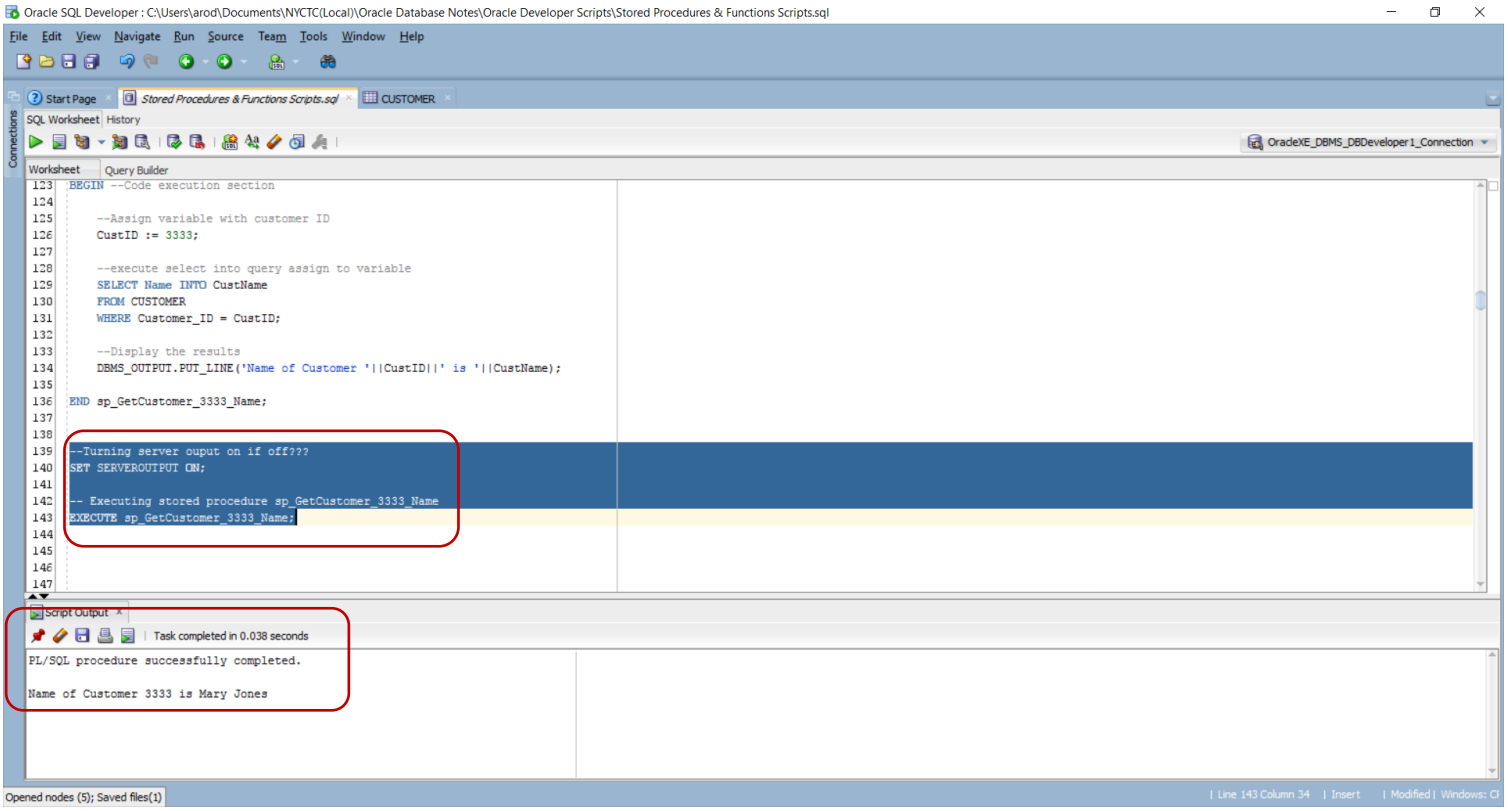
2) Execute/Compile the Script using the following syntax

```
EXECUTE ProcedureName();
```

3) The statement looks as follows:

```
EXECUTE sp_GetCustomer_3333_Name;
```

4) Message Window should indicate if execution was successful & show output of results



Example 2 – Stored Procedure that displays the name of ANY customer (PRACTICAL SCENARIO – THIS IMPLEMENTATION EXAMPLE IS USED IN THE REAL-WORLD SCENARIOS!)

Target Code to Implement using Stored Procedure

- We modify the previous example to display the name of any customer whose ID is passed as a parameter.
- Target OUTPUT we wish stored procedure to provide & SELECT query that will facilitate:

```
--Select Query with WHERE clause
SELECT Name
FROM CUSTOMER
WHERE Customer_ID = ?????;
```

```
--Stored Procedure to display the following messages
    - Name of customer ???? is ?????!
```

- This version will return the name of any customer whose ID is passed as parameter to the stored procedure which is more practical.

Specification Implementation Steps

Step 1: Design the Stored Procedure Definition

- Design:
  - First to implement we would need the following:
    - SELECT INTO statement & WHERE clause to return only one row.
    - Variables to store the results of select into
    - PL/SQL language components to display the results.

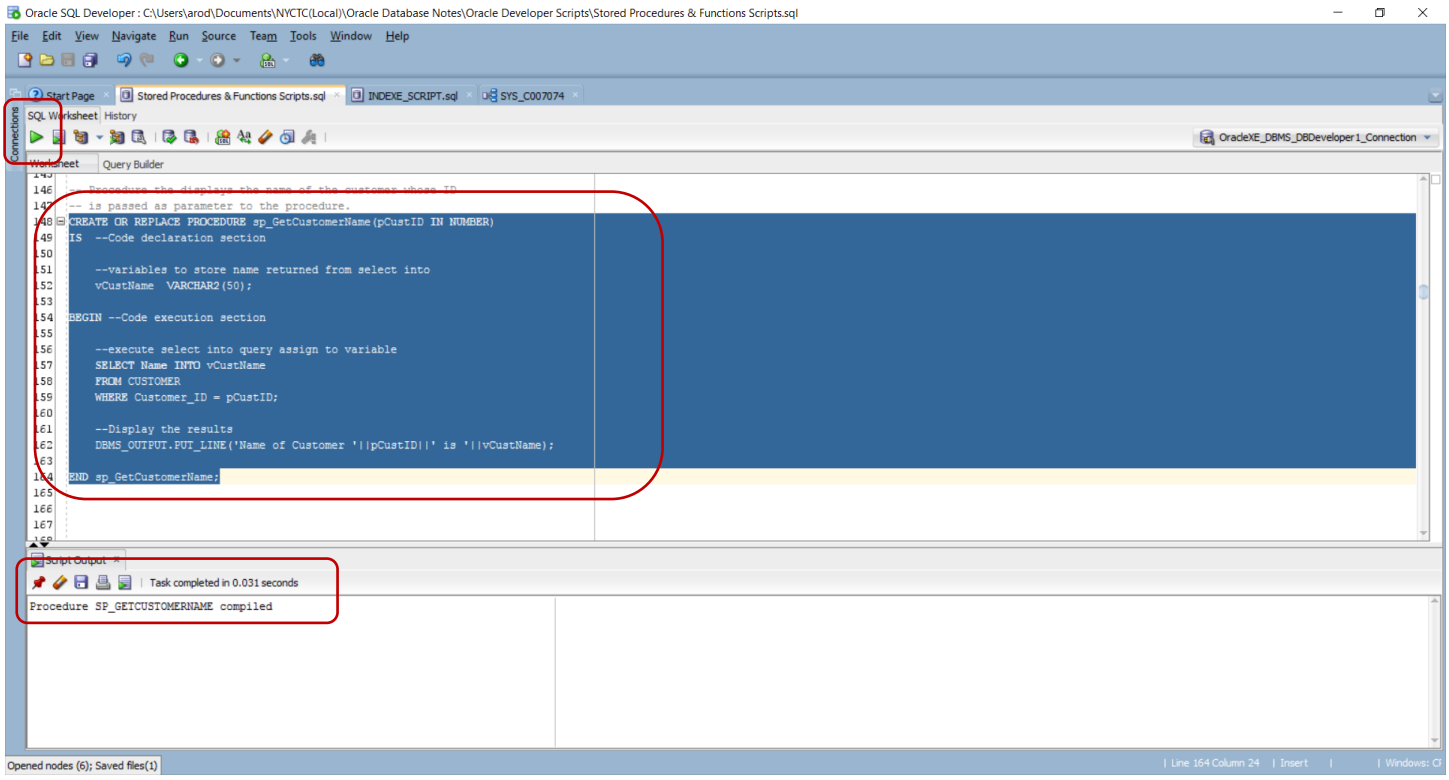
```
-- Procedure the displays the name of the customer whose ID
-- is passed as parameter to the procedure.
CREATE OR REPLACE PROCEDURE usp_GetCustomerName (pCustID IN NUMBER)
IS  --Code declaration section

    --variables to store name returned from select into
    vCustName  VARCHAR2(50);

BEGIN --Code execution section

    --execute select into query assign to variable and parameter to query
    SELECT Name INTO vCustName
    FROM CUSTOMER
    WHERE Customer_ID = pCustID;    --Parameter

    --Display the results
    DBMS_OUTPUT.PUT_LINE('Name of Customer '||pCustID||' is '||vCustName);

END sp_GetCustomerName;
```

Step 2: In the Oracle SQL Developer Script Windows Enter the Code & Compile Stored Procedure

- In SQL Developer:

  - Select **File|Open** to navigate to the script file you want to enter code, or Select **File|New** if is a new script file & do the following:

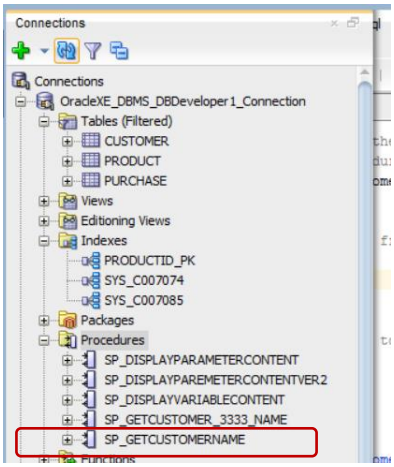
    1) Enter code in script
    2) Execute/Compile the Script
    3) Verify Procedure was successfully compiled



Step 3: In the Oracle SQL Developer Navigator Window, Verify Schema Object was created for Stored Procedure

- In SQL Developer:

  1) In the Navigator Window Verify Schema Object exists



Step 4: In the Oracle SQL Developer Script Windows TEST the Stored Procedure by EXECUTING it

- In SQL Developer:
  - Execute as follows:

1) Since we are displaying data to the message window we need to **TURN SERVER OUTPUT ON**. Enter code to turn the server output on

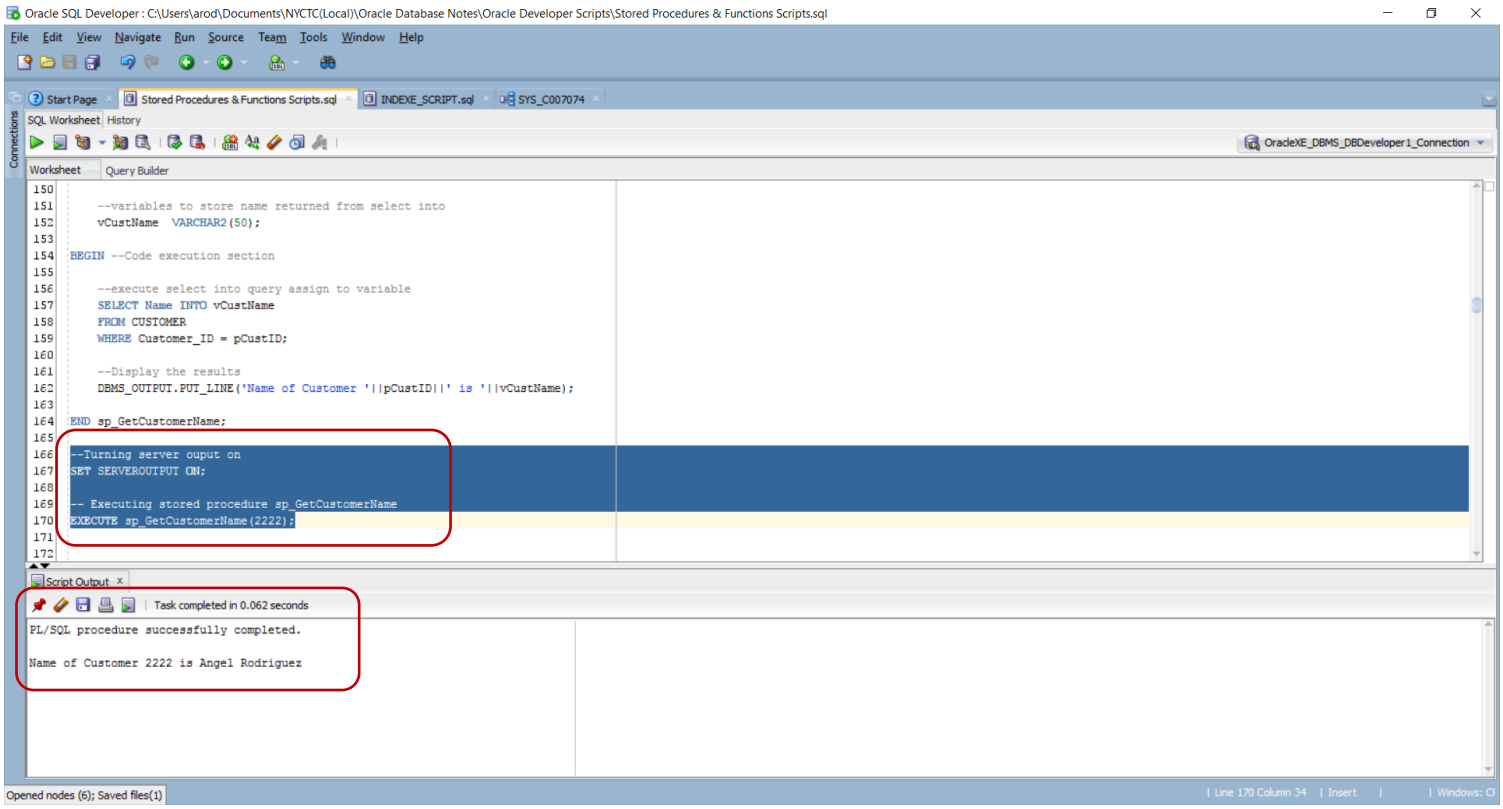
2) Execute/Compile the Script using the following syntax

```
EXECUTE ProcedureName(arg1, arg2, etc.);
```

3) The statement looks as follows:

```
EXECUTE usp_GetCustomerName(2222);
```

4) Message Window should indicate if execution was successful & show output of results



Example 3 – Stored Procedure that displays the entire customer record (PRACTICAL SCENARIO – THIS IMPLEMENTATION EXAMPLE IS USED IN THE REAL-WORLD SCENARIOS!)

Target Code to Implement using Stored Procedure

- In this example, we display the entire record of any customer whose ID is passed as a parameter.
- Target OUTPUT we wish stored procedure to provide & SELECT query that will facilitate:

```
--Select Query with WHERE clause
SELECT Customer_ID, Name, BDate, Address, Phone, Gender, Email
FROM CUSTOMER
WHERE Customer_ID = ?????;
```

```
-- Stored Procedure to display the following messages
    - Customer ID: ????
    - Customer Name: ????
    - Customer Date of Birth: ????
    - Etc.
```

- This version will return all columns of the records from any customer whose ID is passed as parameter to the stored procedure which is more practical.

Specification Implementation Steps

Step 1: Design the Stored Procedure Definition

- Design:

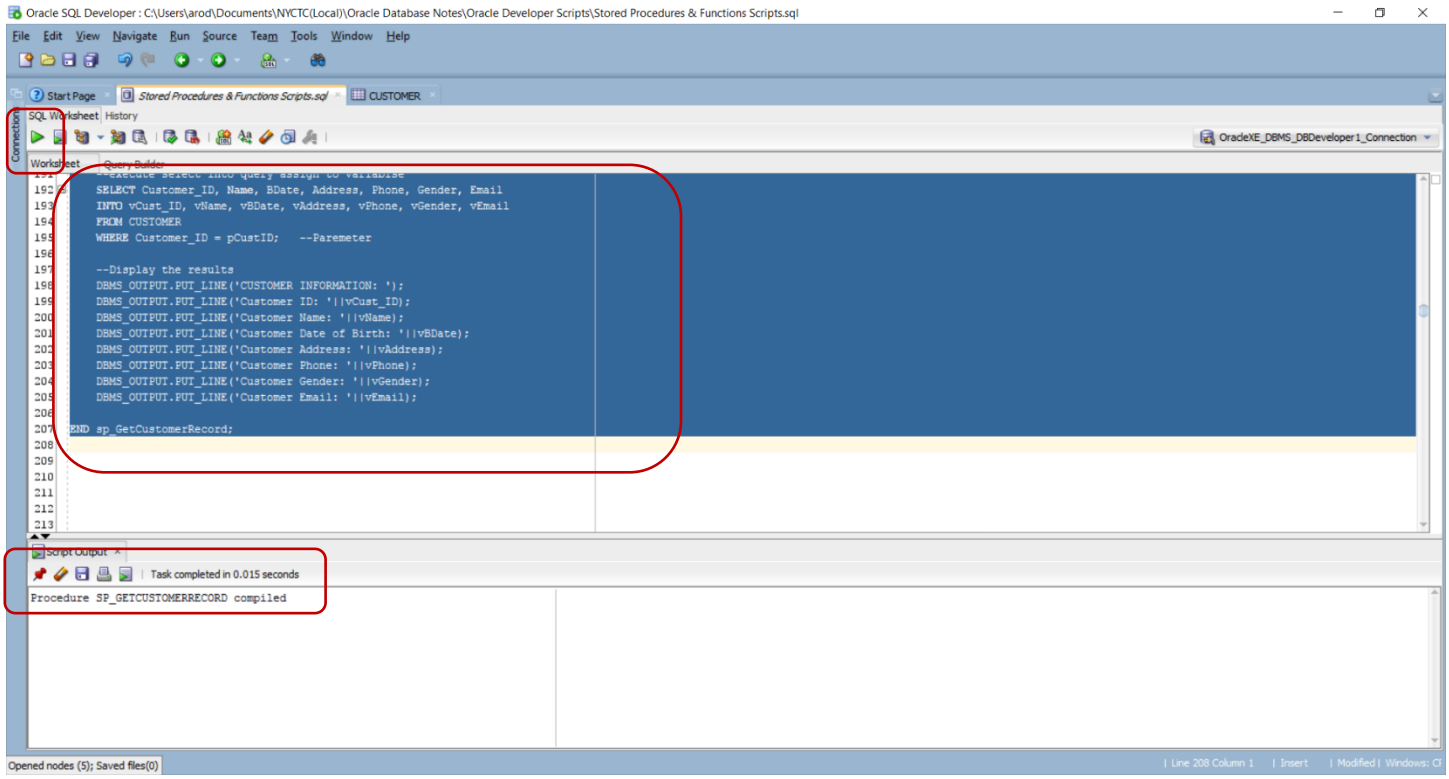
```
-- Procedure the displays all the columns or record of the customer whose ID
-- is passed as parameter to the procedure.
CREATE OR REPLACE PROCEDURE usp_GetCustomerRecord(pCustID IN NUMBER)
IS  --Code declaration section

    --variables to store column values returned from select into
      vCust_ID     NUMBER(4);
      vName        VARCHAR2(50);
      vBDate       DATE;
      vAddress     VARCHAR2(200);
      vPhone       VARCHAR2(20);
      vGender      CHAR(1);
      vEmail       VARCHAR2(50);

BEGIN --Code execution section

    --execute select into query assign to variablse
    SELECT Customer_ID, Name, BDate, Address, Phone, Gender, Email
    INTO vCust_ID, vName, vBDate, vAddress, vPhone, vGender, vEmail
    FROM CUSTOMER
    WHERE Customer_ID = pCustID;   --Paremeter

    --Display the results
    DBMS_OUTPUT.PUT_LINE('CUSTOMER INFORMATION: ');
    DBMS_OUTPUT.PUT_LINE('Customer ID: '||vCust_ID);
    DBMS_OUTPUT.PUT_LINE('Customer Name: '||vName);
    DBMS_OUTPUT.PUT_LINE('Customer Date of Birth: '||vBDate);
    DBMS_OUTPUT.PUT_LINE('Customer Address: '||vAddress);
    DBMS_OUTPUT.PUT_LINE('Customer Phone: '||vPhone);
    DBMS_OUTPUT.PUT_LINE('Customer Gender: '||vGender);
    DBMS_OUTPUT.PUT_LINE('Customer Email: '||vEmail);

END sp_GetCustomerRecord;
```

Step 2: In the Oracle SQL Developer Script Windows Enter the Code & Compile Stored Procedure

- In SQL Developer:

  - Select **File|Open** to navigate to the script file you want to enter code, or Select **File|New** if is a new script file & do the following:

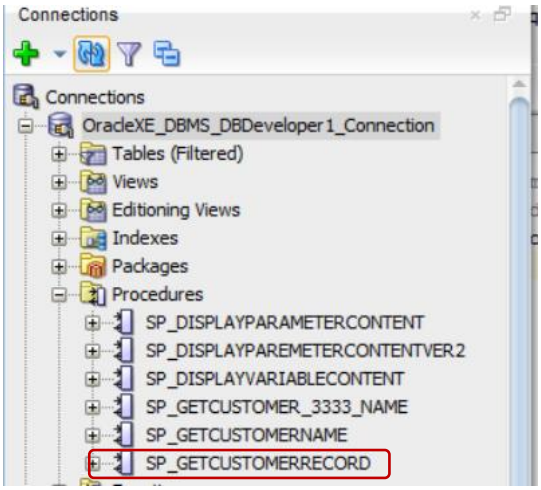
    1) Enter code in script
    2) Execute/Compile the Script
    3) Verify Procedure was successfully compiled



Step 3: In the Oracle SQL Developer Navigator Window, Verify Schema Object was created for Stored Procedure

- In SQL Developer:

  1) In the Navigator Window Verify Schema Object exists



Step 4: In the Oracle SQL Developer Script Windows TEST the Stored Procedure by EXECUTING it

- In SQL Developer:
  - Execute as follows:

1) Since we are displaying data to the message window we need to **TURN SERVER OUTPUT ON**. Enter code to turn the server output on

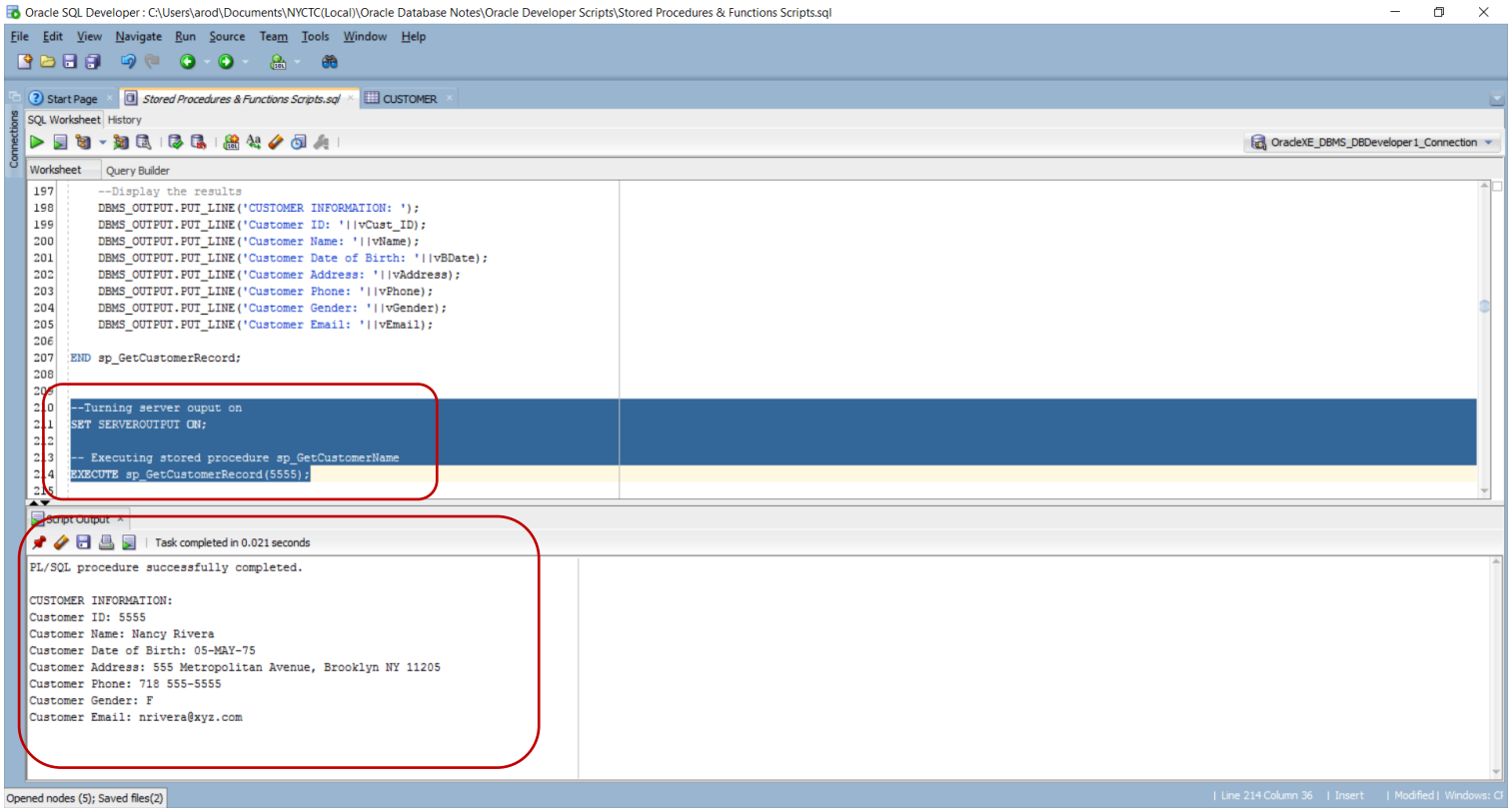
2) Execute/Compile the Script using the following syntax

```
EXECUTE ProcedureName(arg1, arg2, etc.);
```

3) The statement looks as follows:

```
EXECUTE usp_GetCustomerName(2222);
```

4) Message Window should indicate if execution was successful & show output of results



Example 4 – Stored Procedure that displays the entire customer record Version 2 (PRACTICAL SCENARIO – THIS IMPLEMENTATION EXAMPLE IS USED IN THE REAL-WORLD SCENARIOS!)

Target Code to Implement using Stored Procedure

- This example is a modification of the previous example 3. We display the entire record of any customer whose ID is passed as a parameter but this time we use the SELECT * syntax, just to proof we could do this as well. But we still need a list of variables to store the results.
- Target OUTPUT we wish stored procedure to provide & SELECT query that will facilitate:

```
--Select Query with WHERE clause
SELECT *
FROM CUSTOMER
WHERE Customer_ID = ?????;
```

```
--Stored Procedure to display the following messages
    - Customer ID: ????
    - Customer Name: ????
    - Customer Date of Birth: ????
    - Etc.
```

- This version will return all columns of the records from any customer whose ID is passed as parameter to the stored procedure using Select *.

Specification Implementation Steps

Step 1: Design the Stored Procedure Definition

- Design:

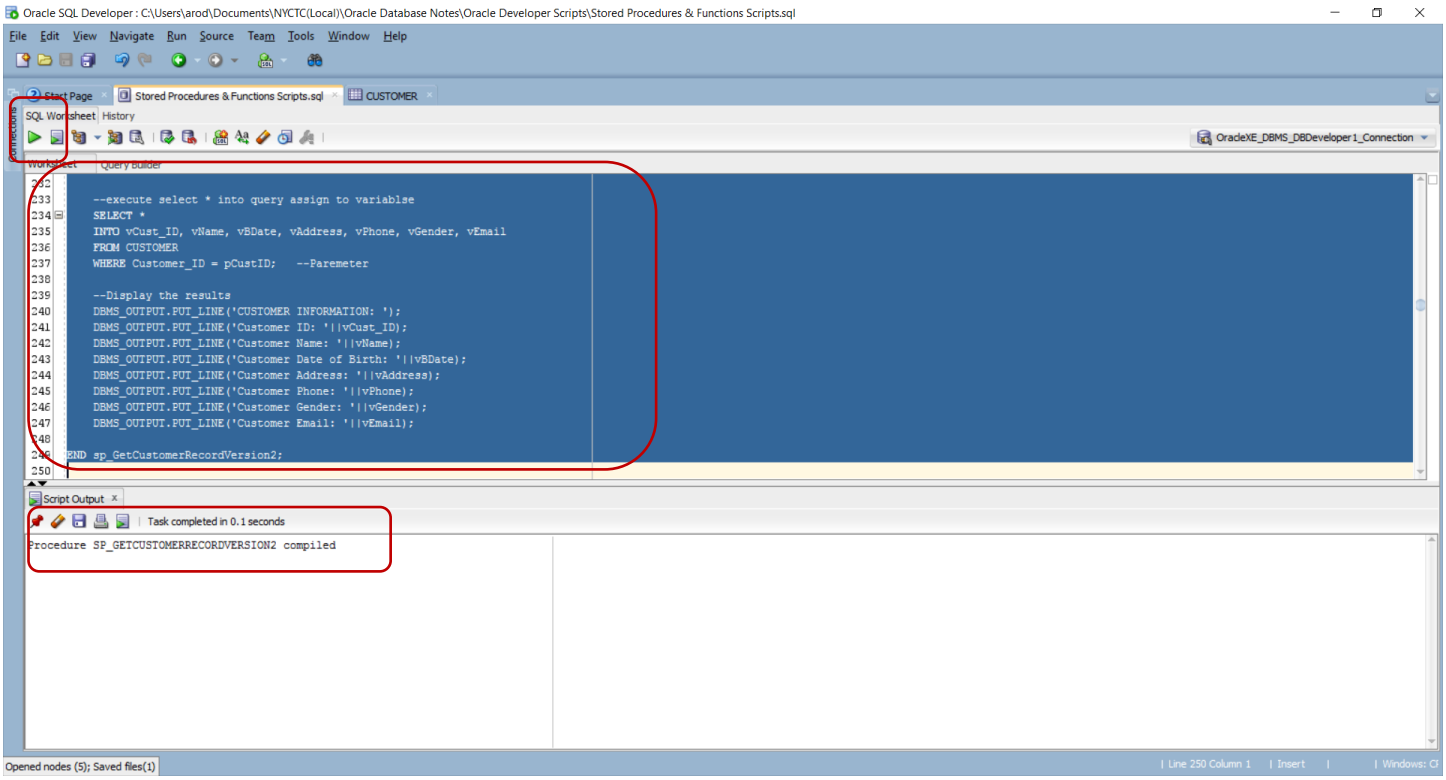
```
-- Procedure the displays all the columns or record of the customer whose ID
-- is passed as parameter to the procedure.
CREATE OR REPLACE PROCEDURE usp_GetCustomerRecord(pCustID IN NUMBER)
IS  --Code declaration section

    --variables to store column values returned from select into
      vCust_ID     NUMBER(4);
      vName        VARCHAR2(50);
      vBDate       DATE;
      vAddress     VARCHAR2(200);
      vPhone       VARCHAR2(20);
      vGender      CHAR(1);
      vEmail       VARCHAR2(50);

BEGIN --Code execution section

    --execute select * into query assign to variablse
    SELECT *
    INTO vCust_ID, vName, vBDate, vAddress, vPhone, vGender, vEmail
    FROM CUSTOMER
    WHERE Customer_ID = pCustID;    --Paremeter

    --Display the results
    DBMS_OUTPUT.PUT_LINE('CUSTOMER INFORMATION: ');
    DBMS_OUTPUT.PUT_LINE('Customer ID: '||vCust_ID);
    DBMS_OUTPUT.PUT_LINE('Customer Name: '||vName);
    DBMS_OUTPUT.PUT_LINE('Customer Date of Birth: '||vBDate);
    DBMS_OUTPUT.PUT_LINE('Customer Address: '||vAddress);
    DBMS_OUTPUT.PUT_LINE('Customer Phone: '||vPhone);
    DBMS_OUTPUT.PUT_LINE('Customer Gender: '||vGender);
    DBMS_OUTPUT.PUT_LINE('Customer Email: '||vEmail);

END sp_GetCustomerRecord;
```

Step 2: In the Oracle SQL Developer Script Windows Enter the Code & Compile Stored Procedure

- In SQL Developer:

  - Select **File|Open** to navigate to the script file you want to enter code, or Select **File|New** if is a new script file & do the following:

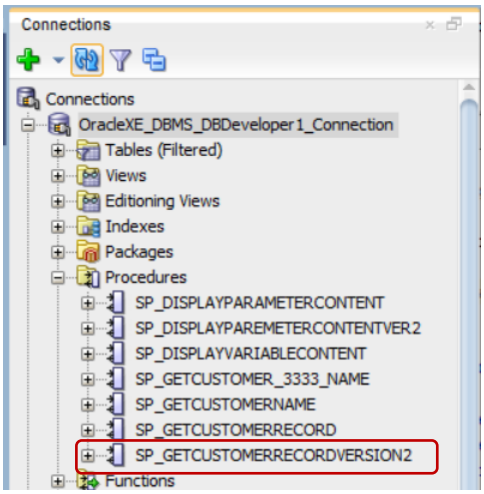
    1) Enter code in script
    2) Execute/Compile the Script
    3) Verify Procedure was successfully compiled



Step 3: In the Oracle SQL Developer Navigator Window, Verify Schema Object was created for Stored Procedure

- In SQL Developer:

  1) In the Navigator Window Verify Schema Object exists



Step 4: In the Oracle SQL Developer Script Windows TEST the Stored Procedure by EXECUTING it

- In SQL Developer:
  - Execute as follows:

1) Since we are displaying data to the message window we need to **TURN SERVER OUTPUT ON**. Enter code to turn the server output on

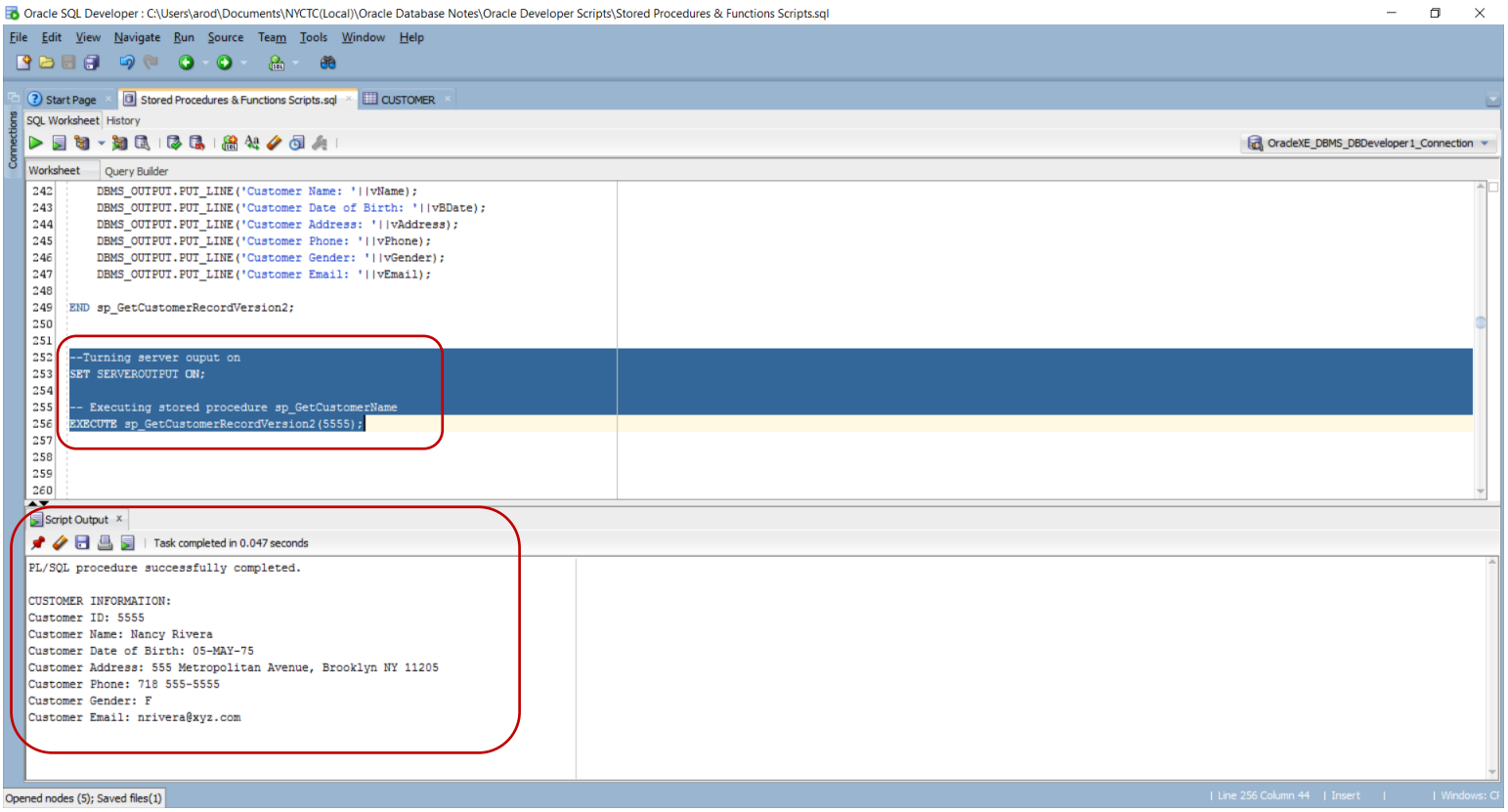
2) Execute/Compile the Script using the following syntax

```
EXECUTE ProcedureName(arg1, arg2, etc.);
```

5) The statement looks as follows:

```
EXECUTE usp_GetCustomerRecord(5555);
```

3) Message Window should indicate if execution was successful & show output of results



4.2.2 Using Cursors to Execute SELECT Statements that return multiple records in a SCRIPT

- Inside Stored Procedures SELECT STATEMENTS can ONLY RETURN one record.
- If you need to create a Stored Procedure with a SELECT STATEMENT that returns MORE THAN ONE RECORD, than YOU NEED TO USE A CURSOR.
- To understand CURSORS inside stored procedures, we first need to understand how CURSORS work.
- This we will do NOT using STORED PROCEDURES, but using CURSORS in a script.
- This will help us understand CURSORS. After, we then use CURSORS inside STORED PROCEDURES.

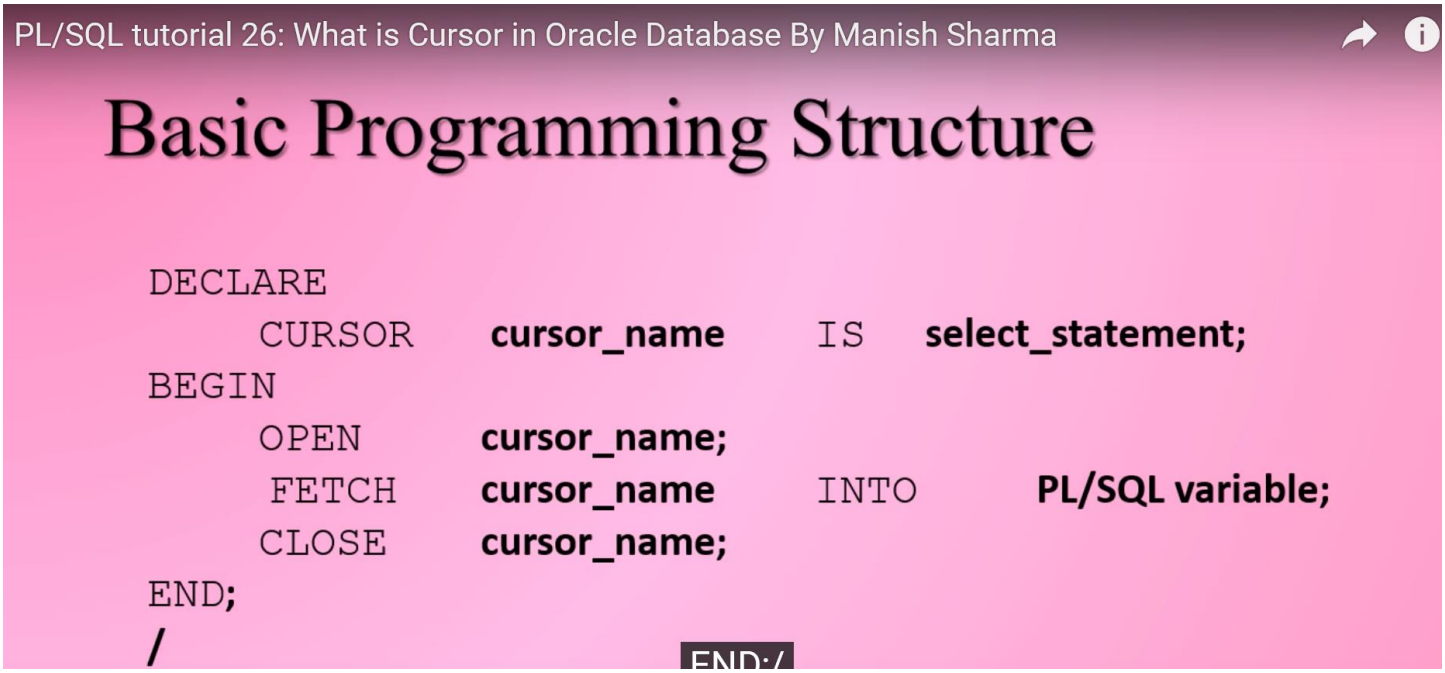
Understanding Cursors

- A CURSOR is a POINTER to a memory location where the results of a QUERY is located.

- CURSOR basics:
  - CURSORS must be declared inside a block of code, for example:
    - Within a block of code defined by the **DECLARE**, **BEGIN** & **END** statement created in a script.
    - Stored Procedure
  - To understand CURSORS, we will start with declaring a block of code inside a **SCRIPT** using the **DECLARE**, **BEGIN** & **END** statement

Cursor Syntax for a Script File

- This is the CURSOR syntax when you are using it within a SCRIPT (NOT STORED PROCEDURE):



Example 1 – Using Cursor to execute SELECT QUERY that returns multiple records of all Female Customers Names

Target Code to Implement using Cursor

- This example we display the name of all female customers in the business using cursor.
- Target SELECT QUERY & expected OUTPUT is shown below:

```
--Select Query with WHERE clause
SELECT Name
FROM CUSTOMER
WHERE Gender = 'F';
```

```
--Script using CURSOR to display the following messages
Name of Female Customer Josephine Smith (Jo)
Name of Female Customer Mary Jones
Name of Female Customer Nancy Rivera
```

Implementation Steps

Step 1: Design/define the CURSOR declaration within a DECLARE, BEGIN & END Block of Code

- Design:

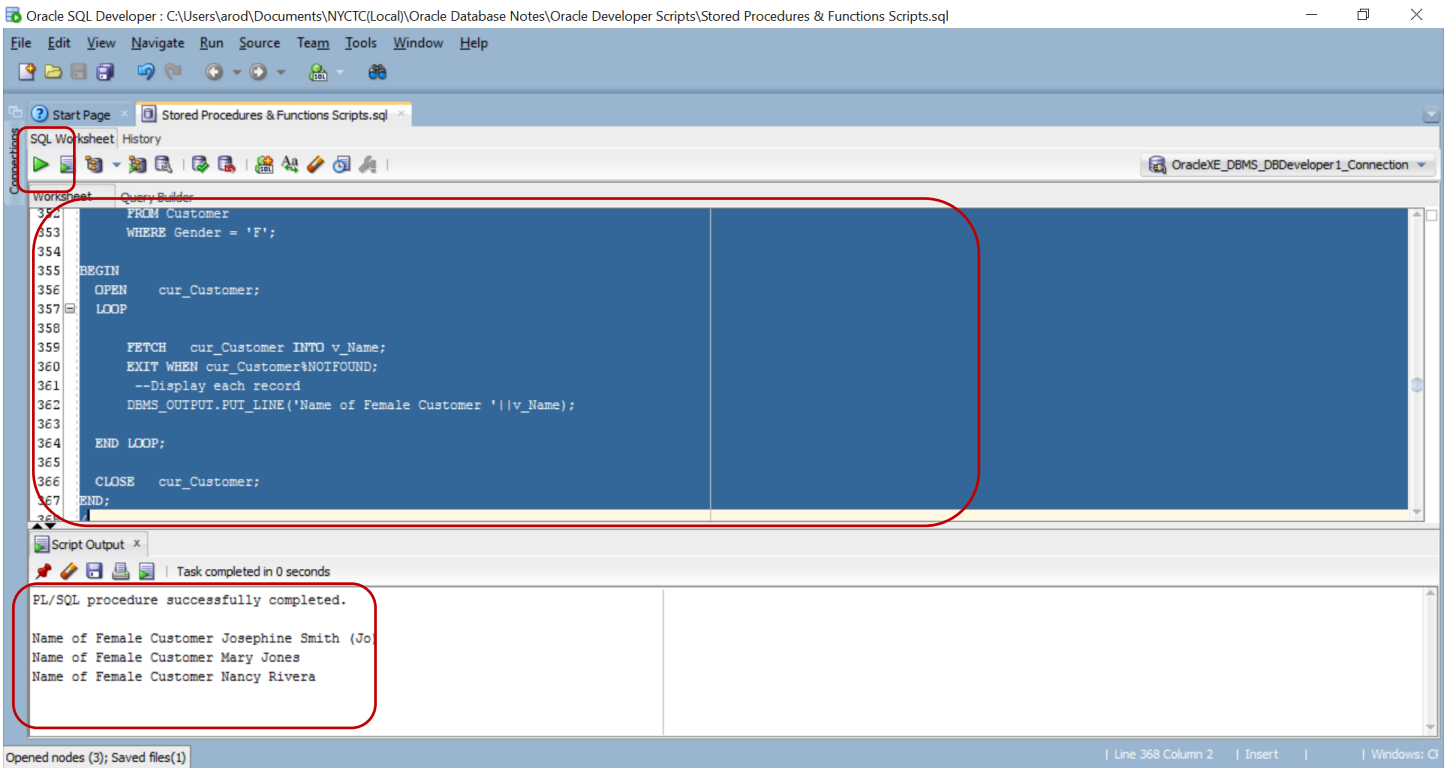
```
--script that displays all female customers in the business using cursor

--Turning server output on so we can see the results in the script
SET SERVEROUTPUT ON;

--Define Block of Code
DECLARE
--declare variable needed for SELECT INTO statement
v_Name VARCHAR2(50);

      --Declare Cursor
      CURSOR cur_Customer IS
            -- Query cursor will point to results
            SELECT Name
            FROM Customer
            WHERE Gender = 'F';
BEGIN
      --Open Cursor
        OPEN    cur_Customer;
      --Use PL/SQL language control or loop to display each record pointed by cursor
        LOOP
            --Fetch Cursor
            FETCH   cur_Customer INTO v_Name;
            --Exit loop if pointing to END OF FILE (cursor pointing to null)
            EXIT WHEN cur_Customer%NOTFOUND;

             --Display each record
            DBMS_OUTPUT.PUT_LINE('Name of Female Customer '||v_Name);

        END LOOP;

      --Close curson when done
        CLOSE   cur_Customer;
END;  --End script
```

Step 2: In the Oracle SQL Developer Script Windows Enter the Code & Compile Stored Procedure

- In SQL Developer:
  - Inside a script, do the following:

    1) Enter code in script
    2) Execute/Compile the Script
    3) View results on the output window

  - Execution of these steps shown below:



Example 2 – Using Cursor to execute SELECT QUERY that returns multiple records will all Female Customers & all their attributes/columns

Target Code to Implement using Cursor

- This example we display the name of all female customers in the business using cursor.
- Target SELECT QUERY & expected OUTPUT is shown below:

```
--Select Query with WHERE clause
SELECT Customer_ID, Name, BDate, Address, Phone, Gender, Email
FROM CUSTOMER
WHERE Gender = 'F';
```

```
-- Script using CURSOR to display the following messages
Female Customer info: 1111 Josephine Smith (Jo) 01-JAN-81 111
Glendwood Road, Brooklyn NY 11210 718-282-1111 F
josephine.smith@comp.com

Female Customer info: 3333 Mary Jones 03-MAR-73 333 Flatlands
Avenue, Brooklyn NY 11203 718 333-3333 F mjones@xyz.com

Female Customer info: 5555 Nancy Rivera 05-MAY-75 555
Metropolitan Avenue, Brooklyn NY 11205 718 555-5555 F
nrivera@xyz.com
```

Implementation Steps

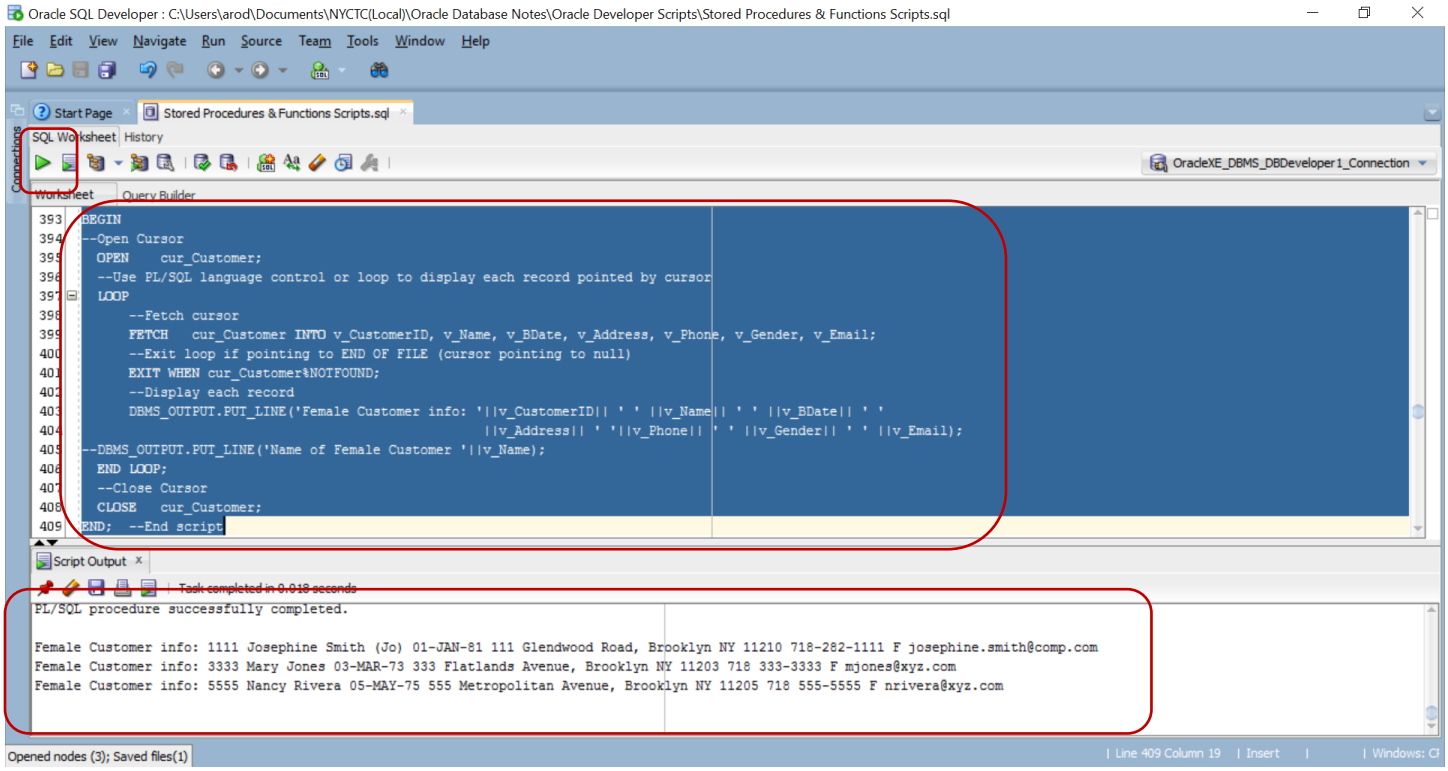
Step 1: Design/define the CURSOR declaration within a DECLARE, BEGIN & END Block of Code

- Design:

```
-- Script using CURSOR that displays the all attributes of all female customers
--Turning server ouput on
SET SERVEROUTPUT ON;

--Declare block of code
DECLARE
      --declare variable needed for SELECT INTO statement
      v_CustomerID  NUMBER(4);
      v_Name        VARCHAR2(50);
      v_BDate       DATE;
      v_Address     VARCHAR2(200);
      v_Phone       VARCHAR2(20);
      v_Gender      CHAR(1);
      v_Email       VARCHAR2(50);

      --Declare Cursor
      CURSOR cur_Customer IS
            -- Query cursor will point to results
            SELECT Customer_ID, Name, Bdate, Address, Phone, Gender, Email
            FROM Customer
            WHERE Gender = 'F';

BEGIN
      --Open Cursor
      OPEN    cur_Customer;
      --Use PL/SQL language control or loop to display each record pointed by cursor
        LOOP
            --Fetch cursor
            FETCH   cur_Customer INTO v_CustomerID, v_Name, v_BDate, v_Address, v_Phone,
            v_Gender, v_Email;
            --Exit loop if pointing to END OF FILE (cursor pointing to null)
            EXIT WHEN cur_Customer%NOTFOUND;
            --Display each record
            DBMS_OUTPUT.PUT_LINE('Female Customer info: '||v_CustomerID|| ' ' ||v_Name||
            ' ' ||v_BDate|| ' ' ||v_Address|| ' ' ||v_Phone|| ' ' ||v_Gender|| ' '
            ||v_Email);
        END LOOP;
        --Close Cursor
        CLOSE   cur_Customer;
END; --End script
```

Step 2: In the Oracle SQL Developer Script Windows Enter the Code & Compile Stored Procedure

- In SQL Developer:
  - Inside a script, do the following:
    1) Enter code in script
    2) Execute/Compile the Script
    3) View results on the output window
  - Execution of these steps shown below:



Example 3 – Using Cursor to execute SELECT QUERY that returns multiple records will all Female Customers attributes/columns (Version 2)

Target Code to Implement using Cursor

- This example we display the attributes or columns of all female customers in the business using cursor.
- Target SELECT QUERY & expected OUTPUT is shown below:

```
--Select Query with WHERE clause
SELECT Customer_ID, Name, BDate, Address, Phone, Gender, Email
FROM CUSTOMER
WHERE Gender = 'F';
```

```
-- Script using CURSOR to display the following messages
Female Customer info: 1111 Josephine Smith (Jo) 01-JAN-81 111
Glendwood Road, Brooklyn NY 11210 718-282-1111 F
josephine.smith@comp.com

Female Customer info: 3333 Mary Jones 03-MAR-73 333 Flatlands
Avenue, Brooklyn NY 11203 718 333-3333 F mjones@xyz.com

Female Customer info: 5555 Nancy Rivera 05-MAY-75 555
Metropolitan Avenue, Brooklyn NY 11205 718 555-5555 F
nrivera@xyz.com
```

Implementation Steps

Step 1: Design/define the CURSOR declaration within a DECLARE, BEGIN & END Block of Code

- Design:

```
-- Script using CURSOR that displays the all attributes of all female customers using
Select *
--Turning server ouput on
SET SERVEROUTPUT ON;

--Declare block of code
DECLARE
      --declare variable needed for SELECT INTO statement
      v_CustomerID  NUMBER(4);
      v_Name        VARCHAR2(50);
      v_BDate       DATE;
      v_Address       VARCHAR2(200);
      v_Phone           VARCHAR2(20);
      v_Gender      CHAR(1);
      v_Email       VARCHAR2(50);

      --Declare Cursor
      CURSOR cur_Customer IS
            -- Query cursor will point to results
            SELECT *
            FROM Customer
            WHERE Gender = 'F';

BEGIN
        --Open Cursor
        OPEN    cur_Customer;
        --Use PL/SQL language control or loop to display each record pointed by cursor
        LOOP
                --Fetch Cursor
            FETCH   cur_Customer INTO v_CustomerID, v_Name, v_BDate, v_Address, v_Phone,
            v_Gender, v_Email;

            --Exit loop if pointing to END OF FILE (cursor pointing to null)
                  EXIT WHEN cur_Customer%NOTFOUND;

            --Display each record
            DBMS_OUTPUT.PUT_LINE('Female Customer info: '||v_CustomerID|| ' ' ||v_Name||
            ' ' ||v_BDate|| ' ' ||v_Address|| ' '||v_Phone|| ' ' ||v_Gender|| ' '
            ||v_Email);

        END LOOP;
        --Close Cursor
        CLOSE   cur_Customer;
END; --End of Script
```

Step 2: In the Oracle SQL Developer Script Windows Enter the Code & Compile Stored Procedure

- In SQL Developer:
  - Inside a script, do the following:
    1) Enter code in script
    2) Execute/Compile the Script
    3) View results on the output window
  - Execution of these steps shown below: