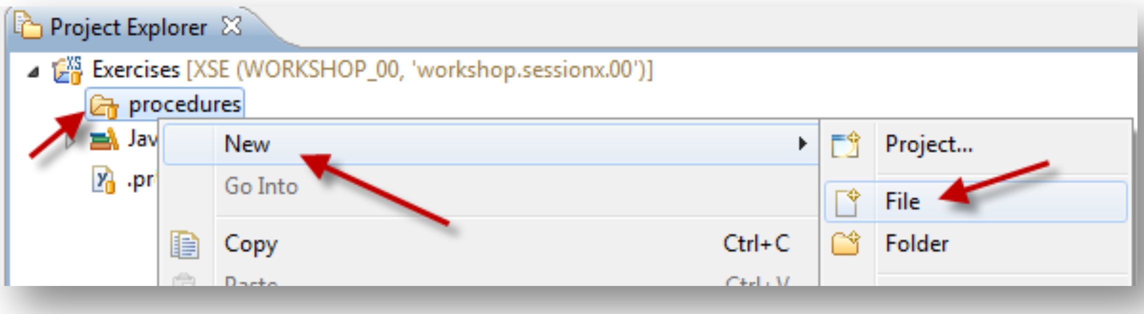
Exercise 3: Using SQLScript

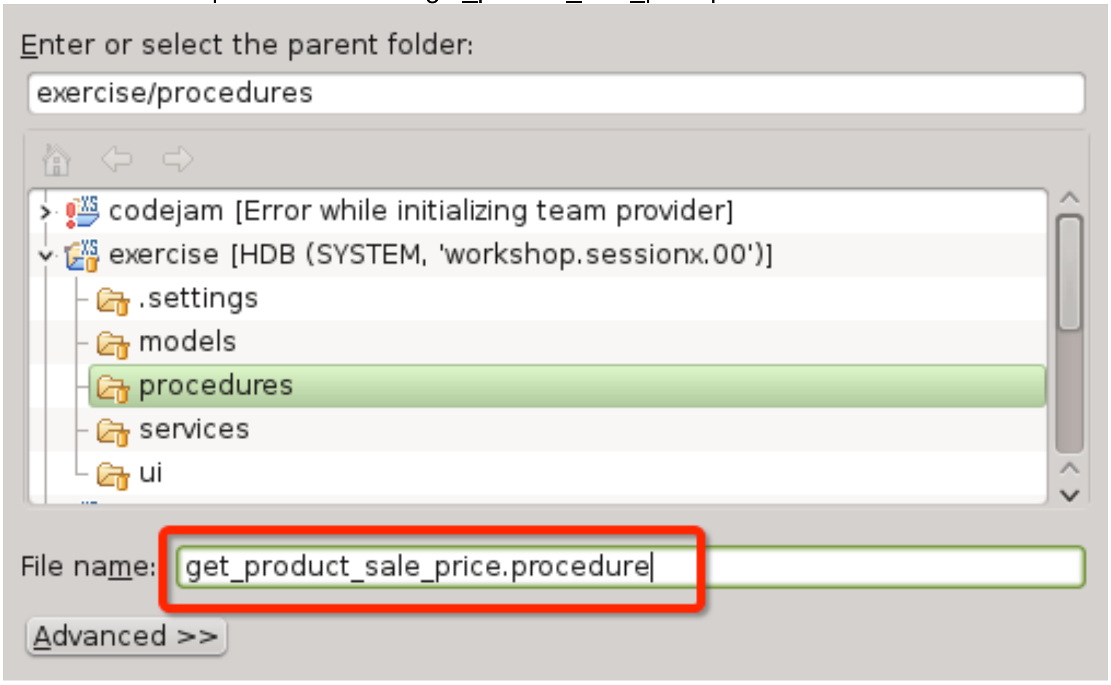
In this exercise you will create a SQLScript procedure for calculating the sales prices based on the discounts for different product categories. Also, use an input parameter to filter the data by product type.

Create SQLScript Procedure with SELECT

1. Create a new file in the “procedures” folder by right clicking the folder and choosing “New”, then “File”.



2. Give the name of the procedure file as “get_product_sale_price.procedure” and click “Finish”.


Enter or select the parent folder:

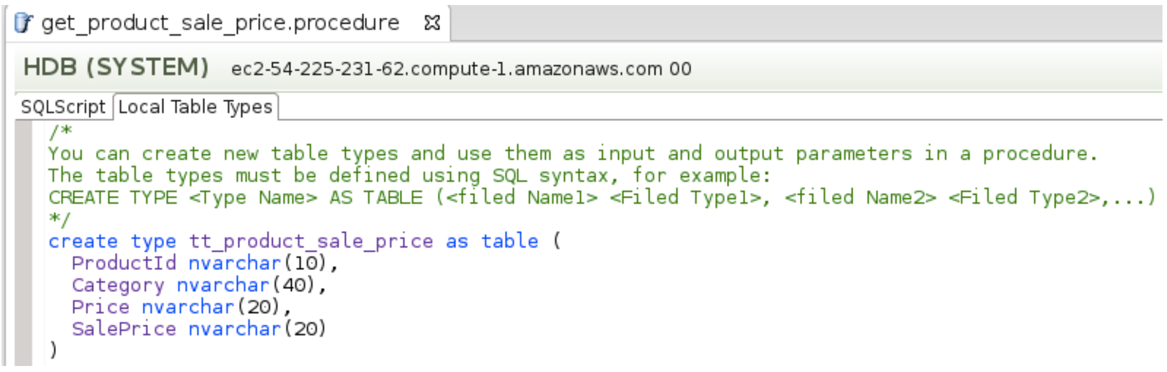
exercise/procedures

codejam [Error while initializing team provider]
exercise [HDB (SYSTEM, 'workshop.sessionx.00')]
.settings
models
procedures
services
ui

File name: get_product_sale_price.procedure

3. The SQLScript editor will be displayed. The procedure parameters are defined between the parentheses in the CREATE PROCEDURE statement. SQLScript coding should be placed between the BEGIN and END statements.

4. Click on the “Local Table Types” tab, define a table type by entering a CREATE TYPE statement as shown below.

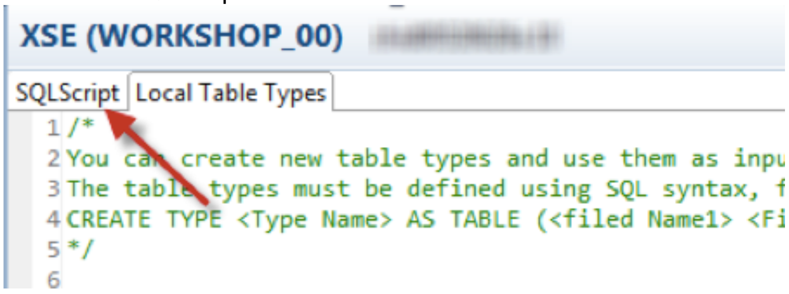


```
get_product_sale_price.procedure
HDB (SYSTEM) ec2-54-225-231-62.compute-1.amazonaws.com 00
SQLScript | Local Table Types
/*
You can create new table types and use them as input and output parameters in a procedure.
The table types must be defined using SQL syntax, for example:
CREATE TYPE <Type Name> AS TABLE (<filed Name1> <Filed Type1>, <filed Name2> <Filed Type2>,...)
*/
create type tt_product_sale_price as table (
  ProductId nvarchar(10),
  Category nvarchar(40),
  Price nvarchar(20),
  SalePrice nvarchar(20)
)
```

Source Code:

```
create type tt_product_sale_price as table (
 ProductId nvarchar(10),
 Category nvarchar(40),
 Price nvarchar(20),
 SalePrice nvarchar(20)
)
```

5. Click the "SQLScript" tab.



```
XSE (WORKSHOP_00)
SQLScript | Local Table Types
/*
You can create new table types and use them as inpu
The table types must be defined using SQL syntax, f
CREATE TYPE <Type Name> AS TABLE (<filed Name1> <Fi
*/
```

6. Define the parameters of this procedure as shown below. One input parameter called "productid", and one output parameter called "product_sale_price", which will use the local table type that you have defined earlier.

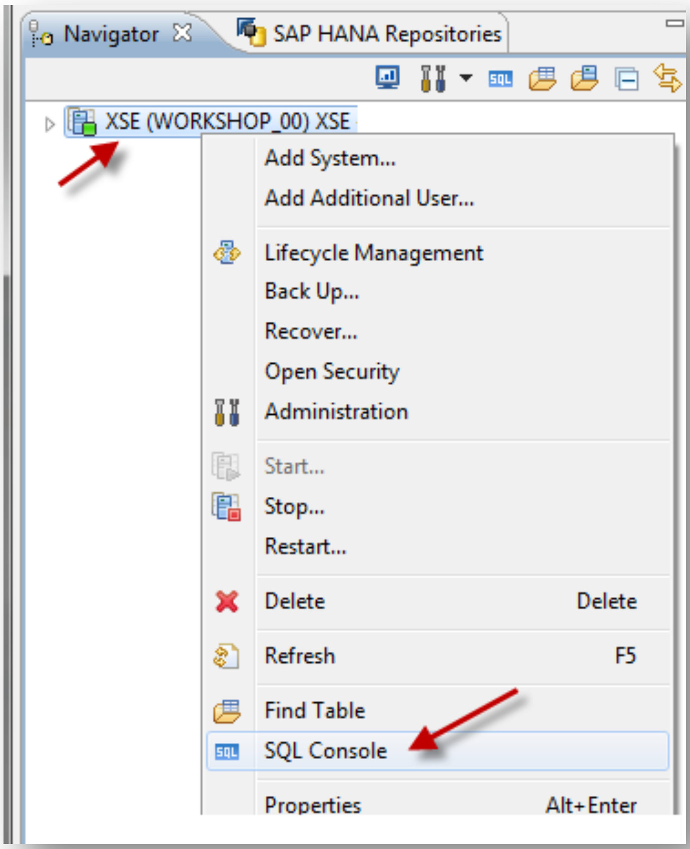
```
CREATE PROCEDURE get_product_sale_price (
    in productid nvarchar(10),
      out product_sale_price tt_product_sale_price )
    LANGUAGE SQLSCRIPT
    SQL SECURITY INVOKER
    READS SQL DATA AS
```

7. Next, insert the SELECT statement between the BEGIN and END statements. The completed coding should look very similar to the following, where we calculate the sale prices based on the discounts offered for the products. For example, 'Notebooks' will get 20% off.

```
CREATE PROCEDURE get_product_sale_price (
    in productid nvarchar(10),
    out product_sale_price tt_product_sale_price )
    LANGUAGE SQLSCRIPT
    SQL SECURITY INVOKER
    READS SQL DATA AS

BEGIN
/*****************************
    Write your procedure logic
 *****************************/
declare lv_category nvarchar(40) := null;
declare lv_discount decimal(15,2) := 0;

lt_product = select "ProductId", "Category", "Price"
                from "SAP_HANA_EPM_DEMO"."sap.hana.democontent.epm.data::products"
                    where "ProductId" = :productid;

select "Category" into lv_category from :lt_product;

if :lv_category = 'Notebooks' then
    lv_discount := .20;
elseif :lv_category = 'Handhelds' then
    lv_discount := .25;
elseif :lv_category = 'Flat screens' then
    lv_discount := .30;
elseif :lv_category like '%printers%' then
   lv_discount := .30;
else
   lv_discount := 1.00;  -- No discount
end if;

product_sale_price =
     select "ProductId", "Category", "Price",
            "Price" - cast(("Price" * :lv_discount) as decimal(15,2)) as "SalePrice"
               from :lt_product;

END;
```

8. Save the procedure, commit it to the HANA repository and activate it.

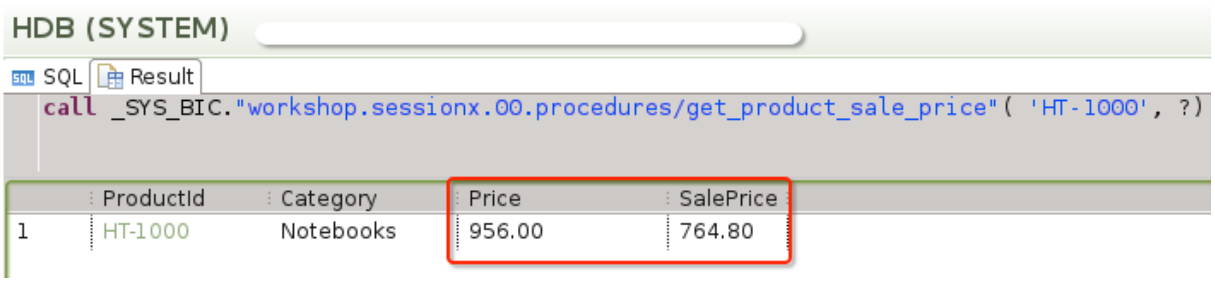
9. The procedure is now activated and a runtime object has been created in the _SYS_BIC schema. You can now call your procedure from the SQL Console. Right-click on the system connection from the Navigator view. Choose "SQL Console".



10. Enter the following CALL statement into the SQL Console. Make sure to replace the session letter and group number.

    ```
    call _SYS_BIC."workshop.sessionx.00.procedures/get_product_sale_price"( 'HT-1000', ?);
    ```

11. Click “Execute” and the results of the procedure call will be displayed.


HDB (SYSTEM)

SQL | Result

```
call _SYS_BIC."workshop.sessionx.00.procedures/get_product_sale_price"( 'HT-1000', ?)
```

	ProductId	Category	Price	SalePrice
1	HT-1000	Notebooks	956.00	764.80

Using the SQLScript Debugger

In this exercise you will debug a SQLScript procedure using the SQLScript debugger. Currently, there is a limitation which restricts the use of the debugger to procedures which do not contain input parameters. A future revision will address this issue, but until then you must use a work around to debug such procedures. You must use a wrapper procedure which directly calls the procedure which you would like to debug.

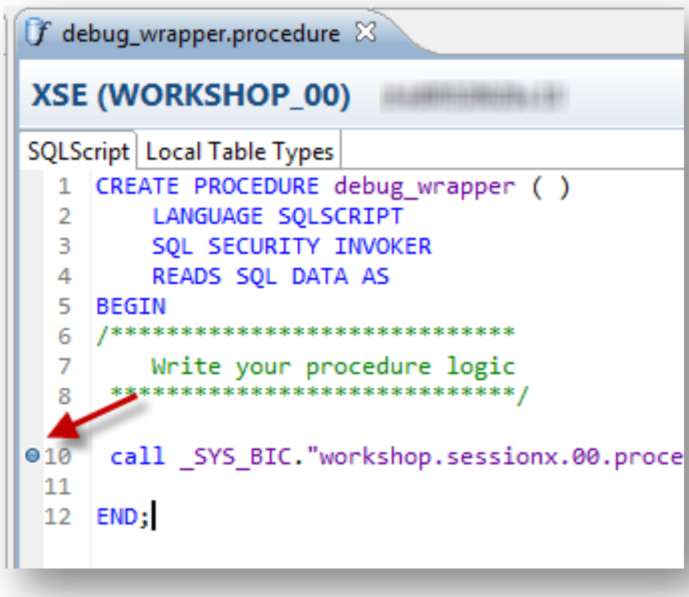
1. Before you can use the debugger, you user must have the relevant privileges. If you are using the SYSTEM user, just skip this step and start from step 2. Otherwise, right click the system and open a SQL Console and execute the following statements, replace the "YOUR_USER" with your own user:
   Source Code:
   ```
   GRANT EXECUTE ON REPOSITORY_REST TO YOUR_USER WITH GRANT OPTION;
   GRANT EXECUTE ON DEBUG TO YOUR_USER WITH GRANT OPTION;
   ```

2. Use what you have learned from the previous exercises and create a new file called "debug_wrapper.procedure".

3. Enter the CALL statement which makes a call to the procedure which you created in the last exercise.

```
CREATE PROCEDURE debug_wrapper ( )
        LANGUAGE SQLSCRIPT
        SQL SECURITY INVOKER
        READS SQL DATA AS
BEGIN
/*****************************
        Write your procedure logic
 *****************************/

        call _SYS_BIC."workshop.sessionx.00.procedures/get_product_sale_price"( 'HT-1000', ?);

END;
```

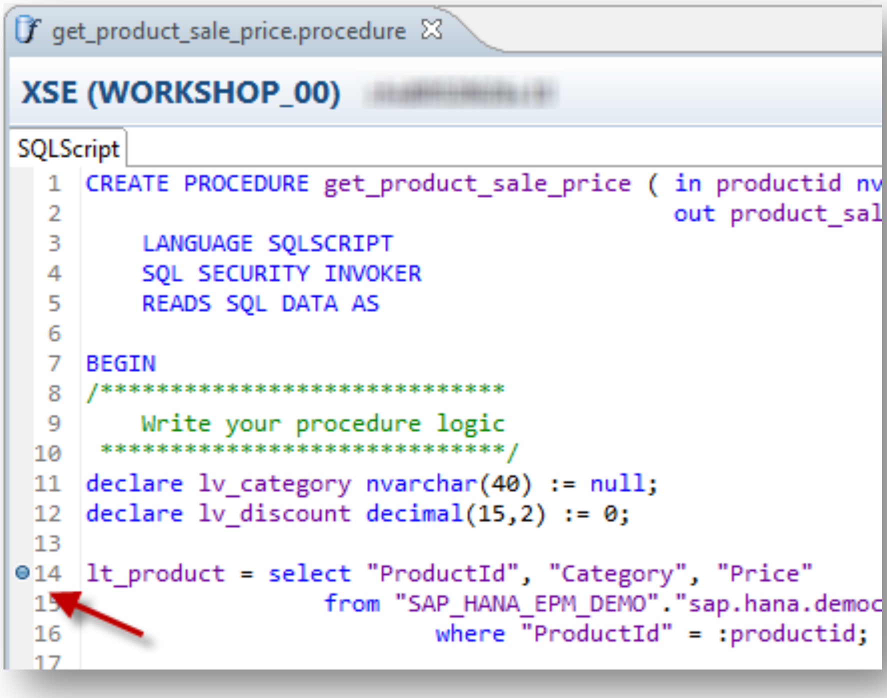
4. Save, commit and activate this procedure. After activation, place a breakpoint on the line call the procedure "get_product_sale_[price" by double clicking next to the line number.



```
 1  CREATE PROCEDURE debug_wrapper ( )
 2      LANGUAGE SQLSCRIPT
 3      SQL SECURITY INVOKER
 4      READS SQL DATA AS
 5  BEGIN
 6  /*****************************
 7      Write your procedure logic
 8   *****************************/
10   call _SYS_BIC."workshop.sessionx.00.proce
11
12  END;
```

5. Under your "Exercises" project, double click the "get_product_sale_price.procedure" file to open the procedure in the editor.

6. Place a breakpoint on line 14 by double-clicking just to the left of the line number. A breakpoint marker will then appear.

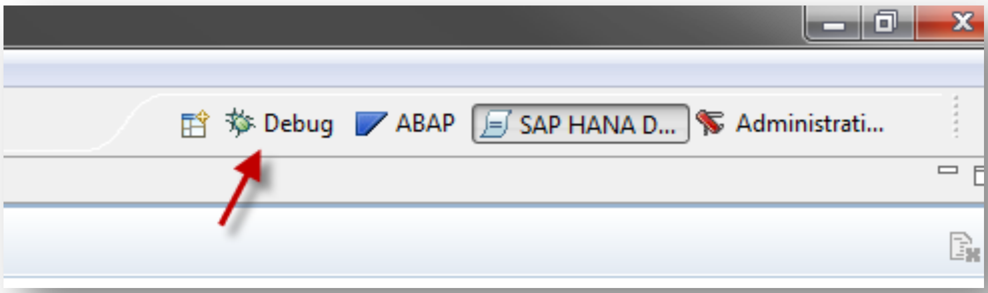

get_product_sale_price.procedure

XSE (WORKSHOP_00) [redacted]

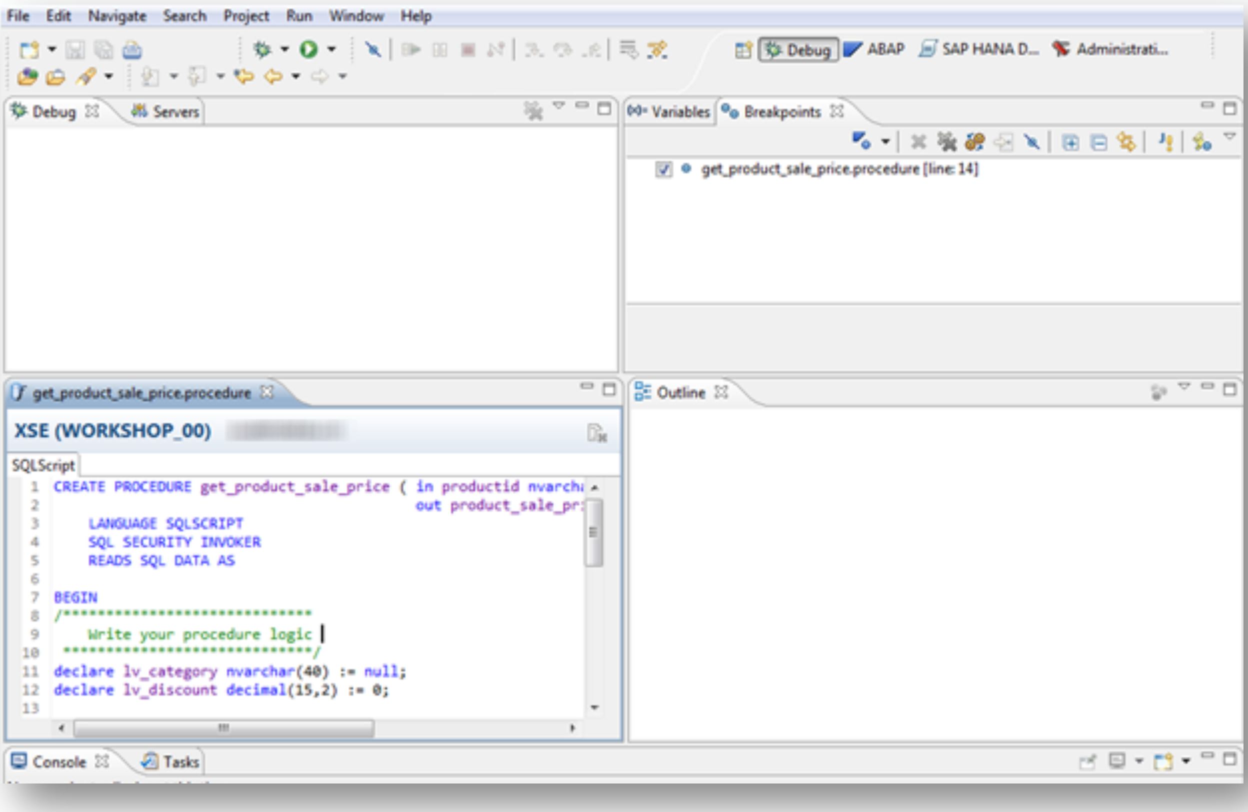
SQLScript

```
 1 CREATE PROCEDURE get_product_sale_price ( in productid nv
 2                                            out product_sal
 3     LANGUAGE SQLSCRIPT
 4     SQL SECURITY INVOKER
 5     READS SQL DATA AS
 6
 7 BEGIN
 8 /******************************
 9     Write your procedure logic
10  ******************************/
11 declare lv_category nvarchar(40) := null;
12 declare lv_discount decimal(15,2) := 0;
13
14 lt_product = select "ProductId", "Category", "Price"
15                  from "SAP_HANA_EPM_DEMO"."sap.hana.democ
16                          where "ProductId" = :productid;
17
```

7. Click on the “Debug” perspective.



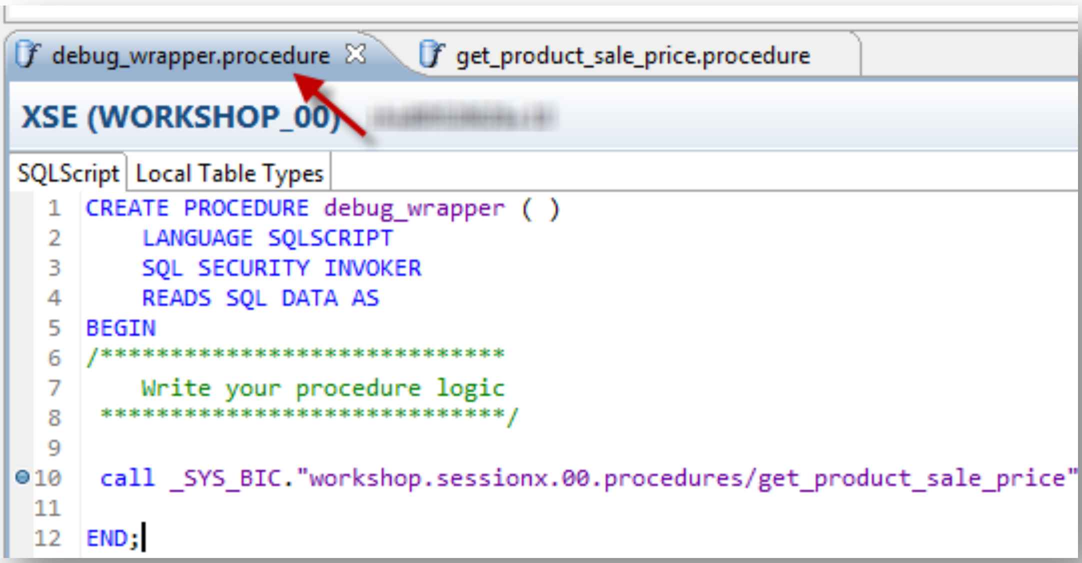
8. The debug perspective should now be visible. You can see the SQLscript procedure in the middle left section, and the breakpoints in the “breakpoint” tab.


File Edit Navigate Search Project Run Window Help
Debug
ABAP
SAP HANA D...
Administrati...
Debug
Servers
Variables
Breakpoints
get_product_sale_price.procedure [line: 14]
get_product_sale_price.procedure
Outline
SQLScript
Console
Tasks

XSE (WORKSHOP_00) [redacted]

```
CREATE PROCEDURE get_product_sale_price ( in productid nvarch
                                          out product_sale_pr

    LANGUAGE SQLSCRIPT
    SQL SECURITY INVOKER
    READS SQL DATA AS

BEGIN
/*****************************
    Write your procedure logic
 *****************************/
declare lv_category nvarchar(40) := null;
declare lv_discount decimal(15,2) := 0;

```

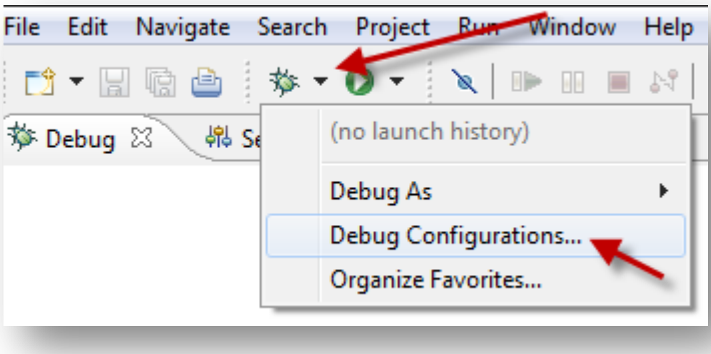
9. Make sure to click on the “debug_wrapper.procedure” tab and make sure that it is the focus.



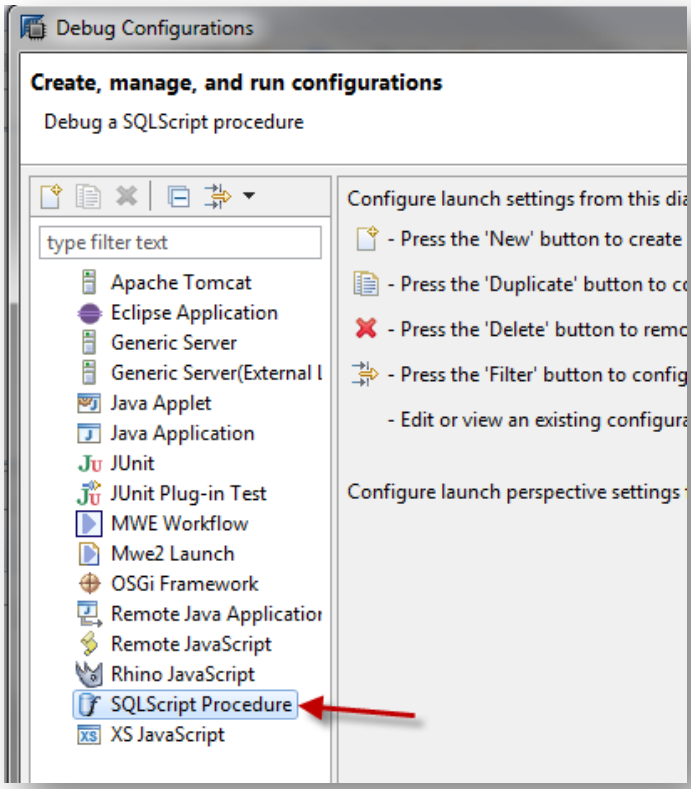
XSE (WORKSHOP_00) [redacted]

```
CREATE PROCEDURE debug_wrapper ( )
    LANGUAGE SQLSCRIPT
    SQL SECURITY INVOKER
    READS SQL DATA AS
BEGIN
/*****************************
    Write your procedure logic
 *****************************/

 call _SYS_BIC."workshop.sessionx.00.procedures/get_product_sale_price"

END;
```

10. Next, create a “Debug Configuration”. Click the drop down next to the “Debug” icon, and choose “Debug Configuration”.



11. In the following dialog, double-click “SQLScript Procedure”.



Create, manage, and run configurations

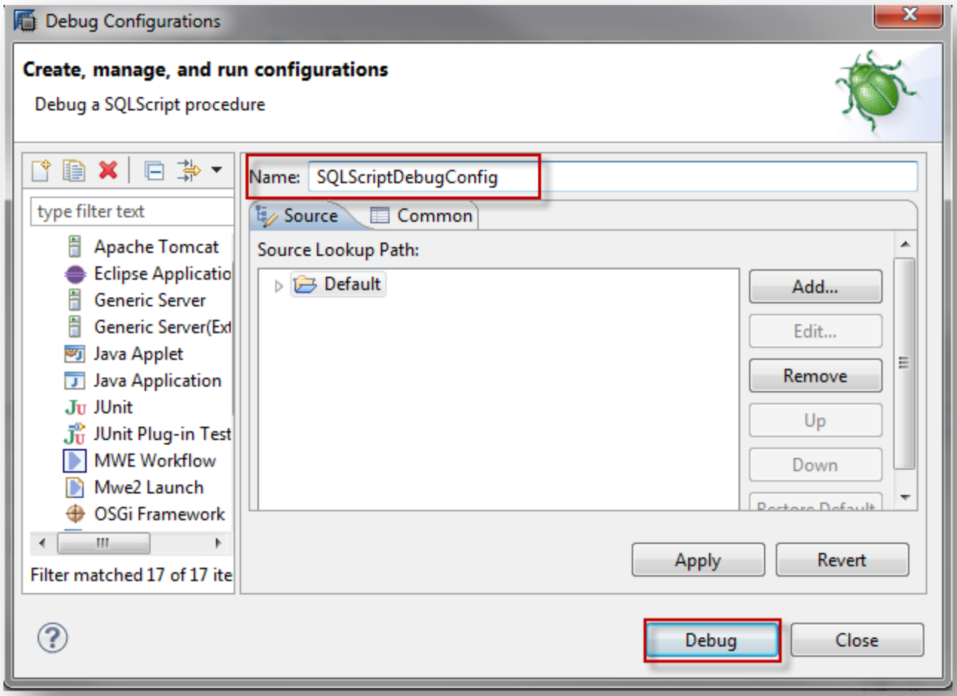
Debug a SQLScript procedure

Configure launch settings from this di

- Press the 'New' button to create
- Press the 'Duplicate' button to c
- Press the 'Delete' button to remo
- Press the 'Filter' button to config
- Edit or view an existing configur

Configure launch perspective settings

12. You can change the name of the configuration or simply keep the default. Next, click “Debug”.


Debug Configurations

Create, manage, and run configurations

Debug a SQLScript procedure

type filter text

- Apache Tomcat
- Eclipse Applicatio
- Generic Server
- Generic Server(Ext
- Java Applet
- Java Application
- JUnit
- JUnit Plug-in Test
- MWE Workflow
- Mwe2 Launch
- OSGi Framework

Filter matched 17 of 17 ite

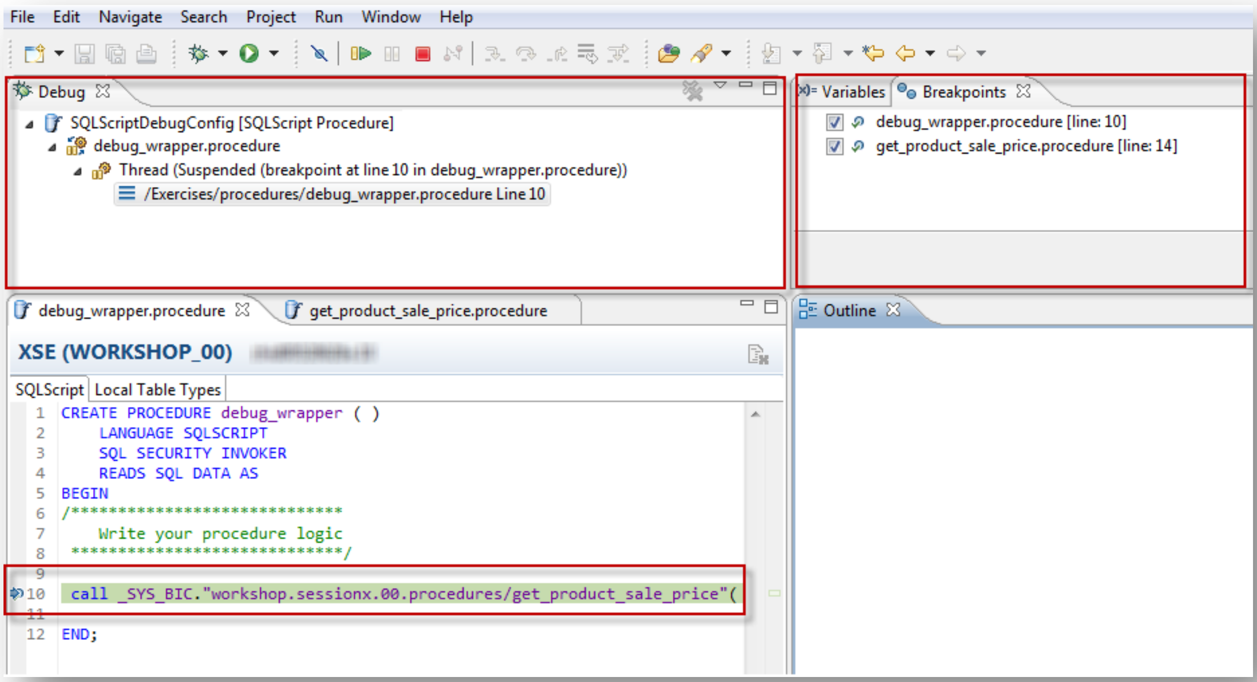
Name: SQLScriptDebugConfig

Source | Common

Source Lookup Path:

Default

13. The procedure is now executed in debug mode. You can see the thread information in the “Debug” tab as well as all of the breakpoints in the “Breakpoint” tab. Also, notice that execution has stopped at line 14 at the first breakpoint.



Debug

```
SQLScriptDebugConfig [SQLScript Procedure]
  debug_wrapper.procedure
    Thread (Suspended (breakpoint at line 10 in debug_wrapper.procedure))
      /Exercises/procedures/debug_wrapper.procedure Line 10
```

Variables | Breakpoints

```
☑ debug_wrapper.procedure [line: 10]
☑ get_product_sale_price.procedure [line: 14]
```

debug_wrapper.procedure | get_product_sale_price.procedure

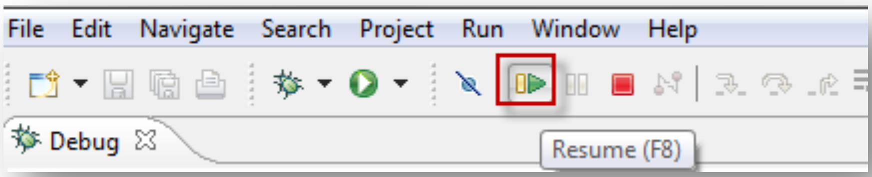
Outline

XSE (WORKSHOP_00) [redacted]

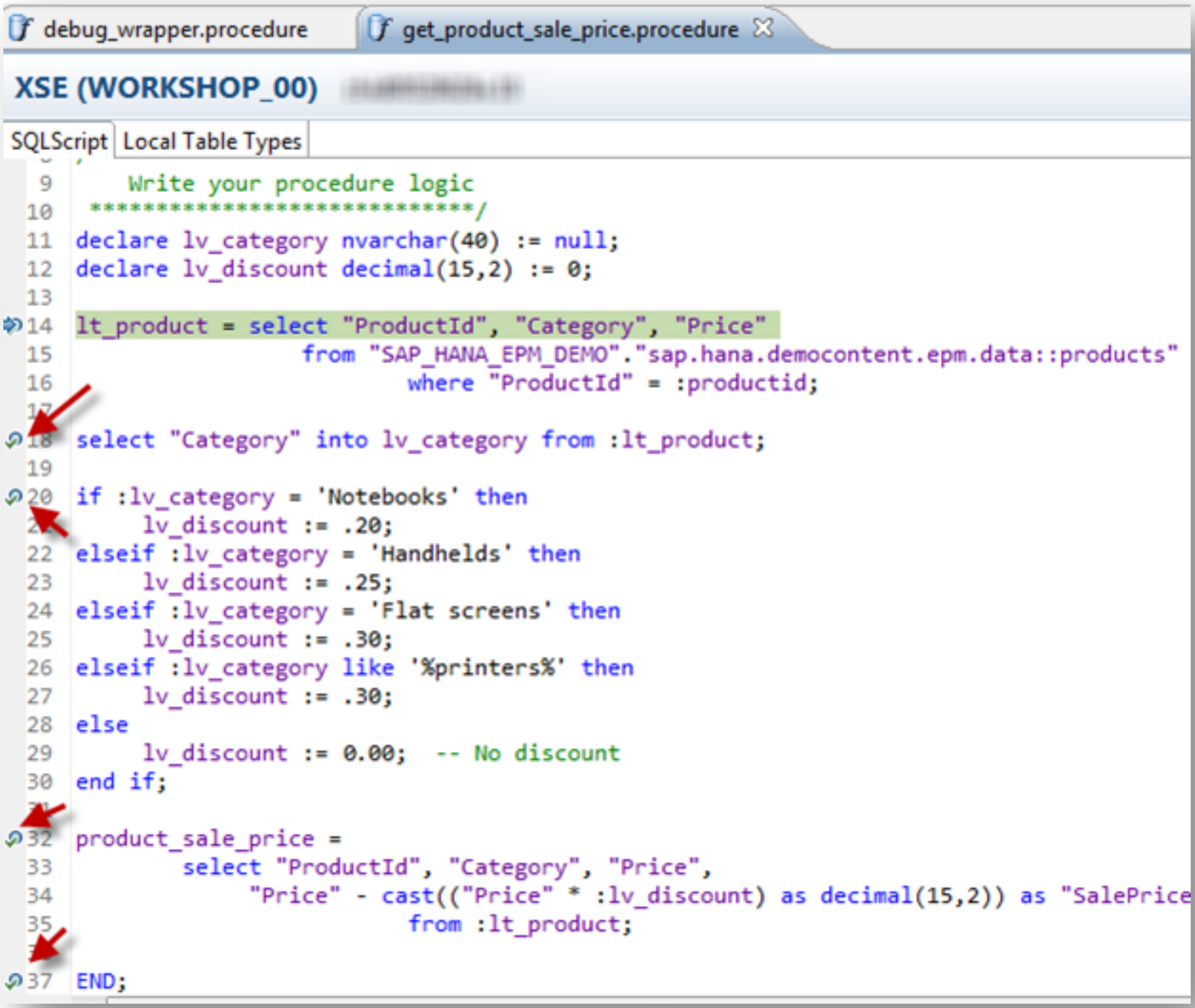
SQLScript | Local Table Types

```
1  CREATE PROCEDURE debug_wrapper ( )
2      LANGUAGE SQLSCRIPT
3      SQL SECURITY INVOKER
4      READS SQL DATA AS
5  BEGIN
6  /*****************************
7      Write your procedure logic
8   *****************************/
9
10  call _SYS_BIC."workshop.sessionx.00.procedures/get_product_sale_price"(
11
12 END;
```

14. Next, click the “Resume” button to resume to the next breakpoint.



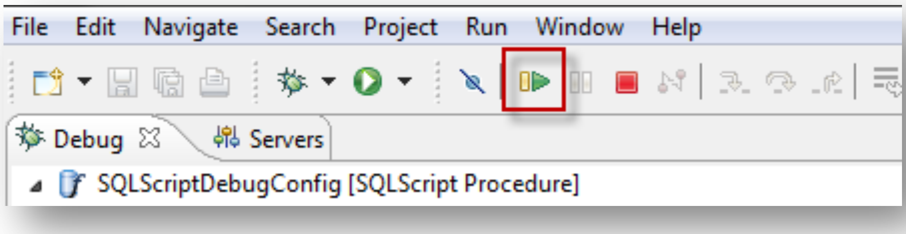
15. You will notice control is now passed over to the called procedure. Set some new breakpoints at lines 18, 20, 32 and 37 as shown below.



XSE (WORKSHOP_00) [redacted]

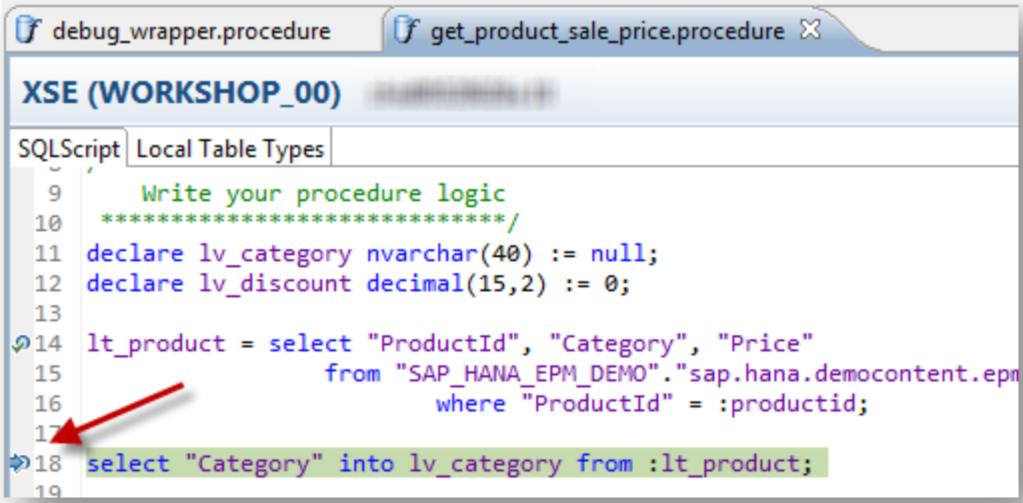
```
 9      Write your procedure logic
10   *******************************/
11  declare lv_category nvarchar(40) := null;
12  declare lv_discount decimal(15,2) := 0;
13
14  lt_product = select "ProductId", "Category", "Price"
15                   from "SAP_HANA_EPM_DEMO"."sap.hana.democontent.epm.data::products"
16                          where "ProductId" = :productid;
17
18  select "Category" into lv_category from :lt_product;
19
20  if :lv_category = 'Notebooks' then
21       lv_discount := .20;
22  elseif :lv_category = 'Handhelds' then
23       lv_discount := .25;
24  elseif :lv_category = 'Flat screens' then
25       lv_discount := .30;
26  elseif :lv_category like '%printers%' then
27       lv_discount := .30;
28  else
29       lv_discount := 0.00;   -- No discount
30  end if;
31
32  product_sale_price =
33          select "ProductId", "Category", "Price",
34               "Price" - cast(("Price" * :lv_discount) as decimal(15,2)) as "SalePrice
35                          from :lt_product;
36
37  END;
```

16. Next, click the “Resume” button..



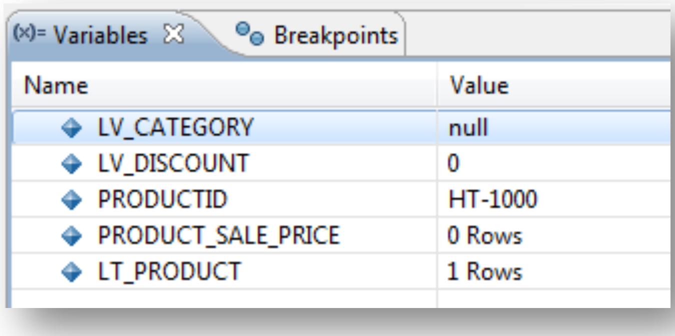
SQLScriptDebugConfig [SQLScript Procedure]

17. Execution will then stop at the next breakpoint.



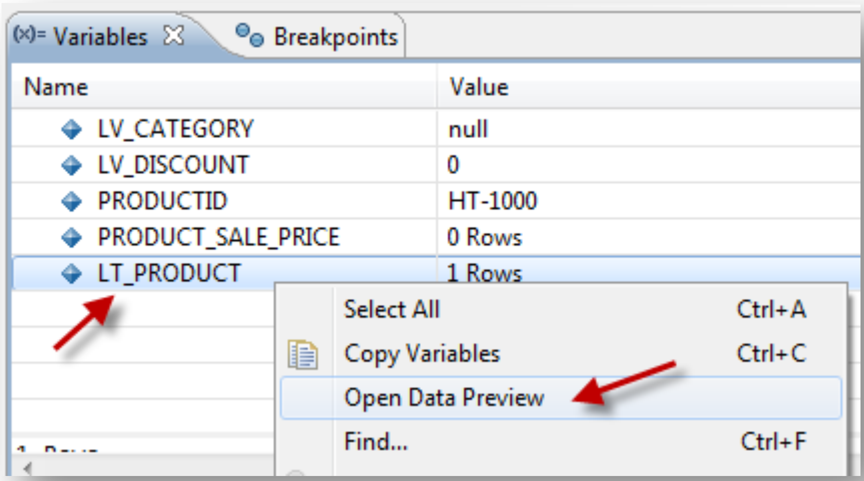
```
    Write your procedure logic
 *****************************/
declare lv_category nvarchar(40) := null;
declare lv_discount decimal(15,2) := 0;

lt_product = select "ProductId", "Category", "Price"
                from "SAP_HANA_EPM_DEMO"."sap.hana.democontent.epm
                        where "ProductId" = :productid;

select "Category" into lv_category from :lt_product;
```

18. Click on the “Variables” tab. All scalar and table variables, including input and output parameters will be shown here along with the actual runtime value. Table variables will show the number of rows as the value.



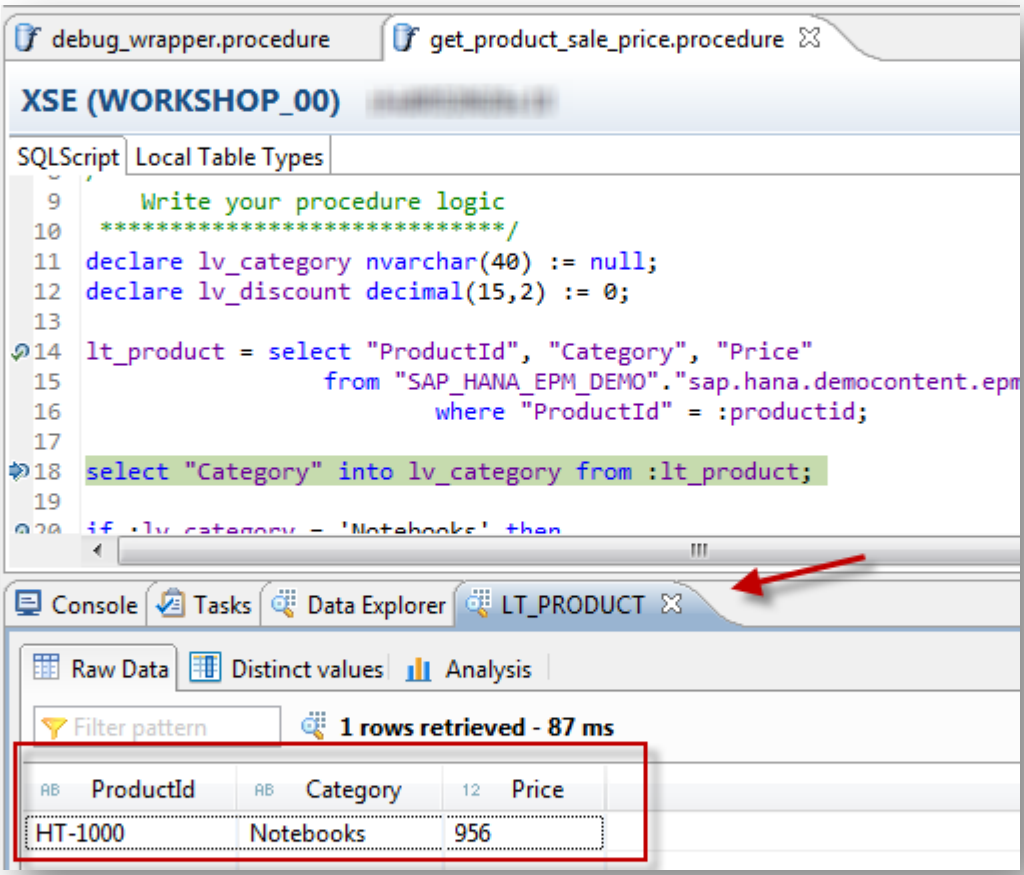
Name	Value
LV_CATEGORY	null
LV_DISCOUNT	0
PRODUCTID	HT-1000
PRODUCT_SALE_PRICE	0 Rows
LT_PRODUCT	1 Rows

19. Right-click on LT_PRODUCT variable, and choose “Open Data Preview”.



Name	Value
LV_CATEGORY	null
LV_DISCOUNT	0
PRODUCTID	HT-1000
PRODUCT_SALE_PRICE	0 Rows
LT_PRODUCT	1 Rows

20. The data will then be shown in a new tab at the bottom of the screen.

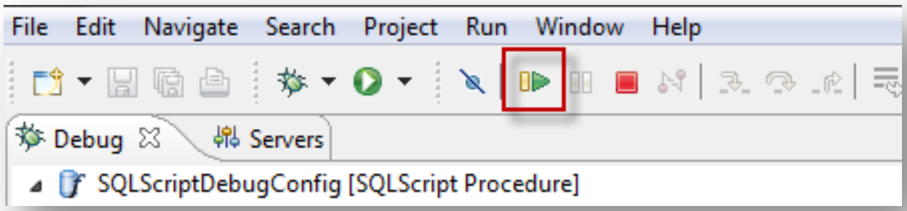


```
   Write your procedure logic
 *****************************/
declare lv_category nvarchar(40) := null;
declare lv_discount decimal(15,2) := 0;

lt_product = select "ProductId", "Category", "Price"
               from "SAP_HANA_EPM_DEMO"."sap.hana.democontent.epm
                      where "ProductId" = :productid;

select "Category" into lv_category from :lt_product;

if :lv_category = 'Notebooks' then
```

1 rows retrieved - 87 ms

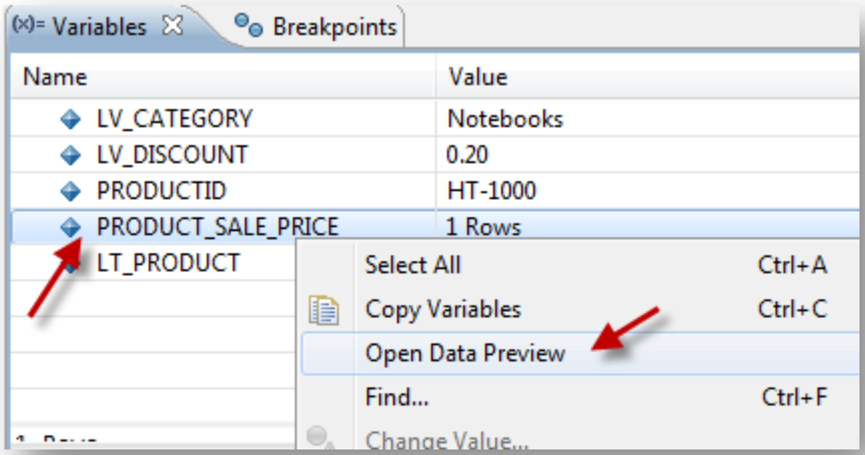
ProductId	Category	Price
HT-1000	Notebooks	956

21. Click the resume button to continue execution, at each breakpoint evaluate the variables in the variables tab. Watch them get updated as you continue to the next breakpoint.



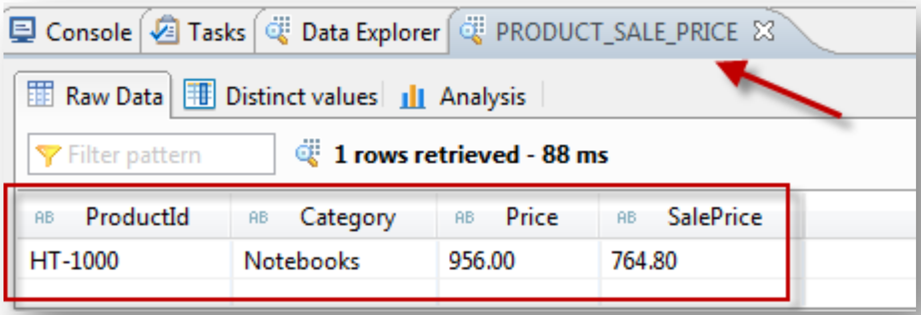
SQLScriptDebugConfig [SQLScript Procedure]

22. When you reach line 37, display the data for the output parameter called “product_sale_price” by right clicking and choosing “Open Data Preview”.



Name	Value
LV_CATEGORY	Notebooks
LV_DISCOUNT	0.20
PRODUCTID	HT-1000
PRODUCT_SALE_PRICE	1 Rows
LT_PRODUCT	

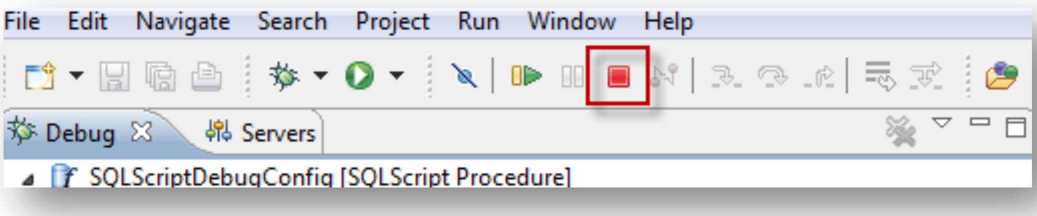
23. You are now viewing the results table, or output parameter for this procedure.



1 rows retrieved - 88 ms

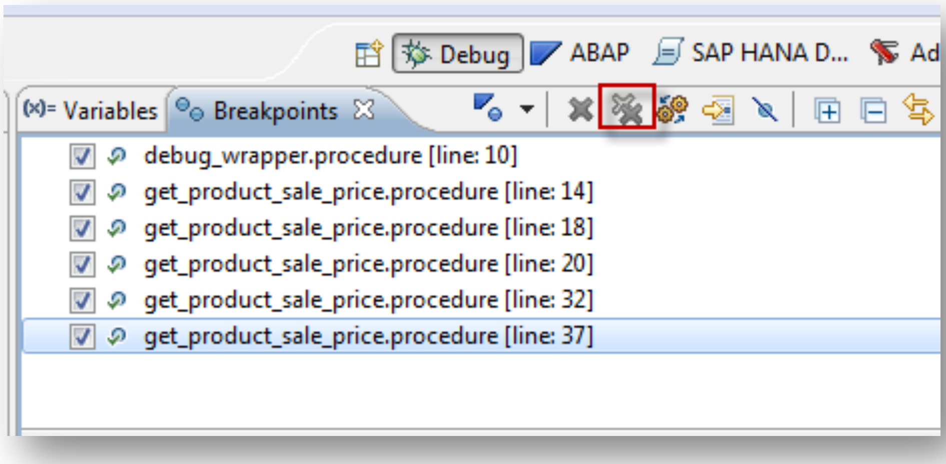
ProductId	Category	Price	SalePrice
HT-1000	Notebooks	956.00	764.80

24. Next, terminate the debug session by clicking the “Terminate” button.



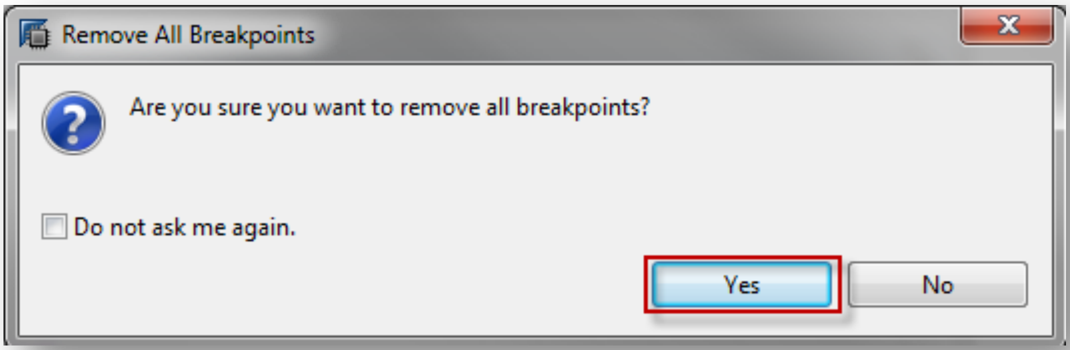
SQLScriptDebugConfig [SQLScript Procedure]

25. Delete all breakpoints by clicking the “Remove All Breakpoints” button in the “Breakpoints” tab.



debug_wrapper.procedure [line: 10]
get_product_sale_price.procedure [line: 14]
get_product_sale_price.procedure [line: 18]
get_product_sale_price.procedure [line: 20]
get_product_sale_price.procedure [line: 32]
get_product_sale_price.procedure [line: 37]

26. Click “Yes”.

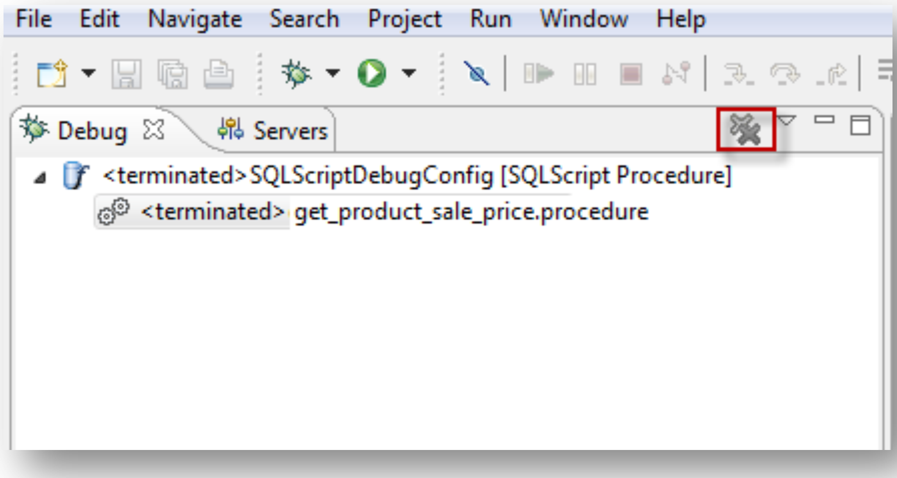


Remove All Breakpoints

Are you sure you want to remove all breakpoints?

Do not ask me again.

27. Remove the terminated sessions by clicking the “Remove All Terminated Launches” button in the “Debug” tab.



<terminated>SQLScriptDebugConfig [SQLScript Procedure]
<terminated> get_product_sale_price.procedure