

DevOps on Z

Proof of Technology (PoT)

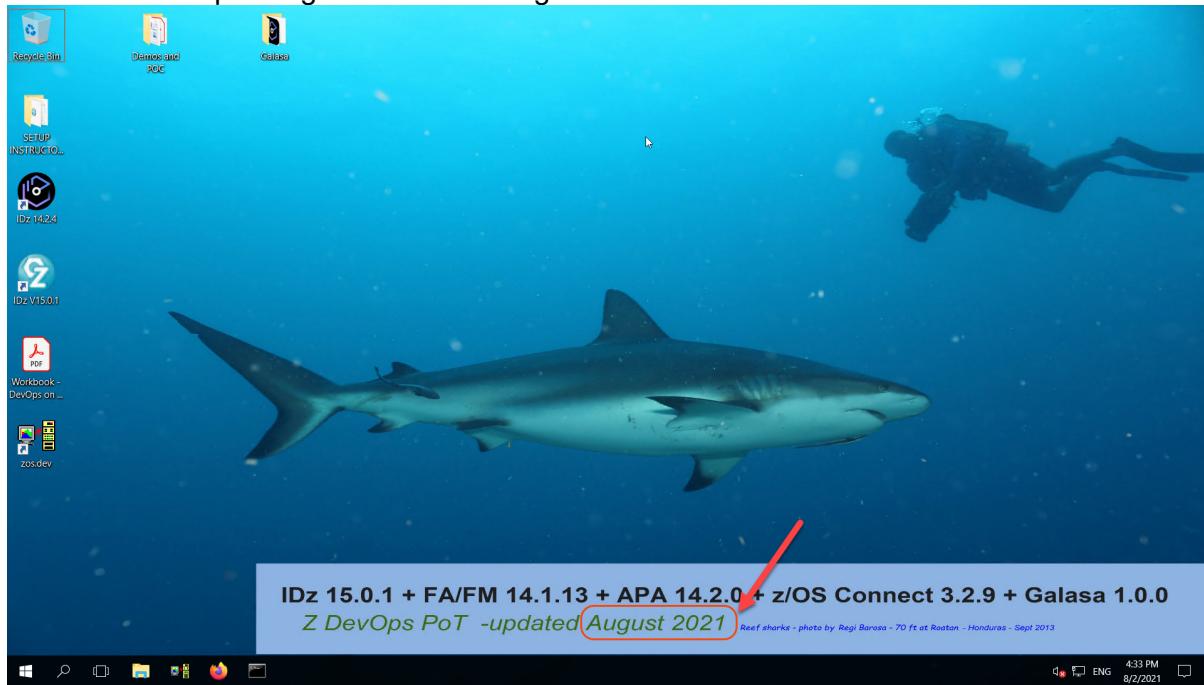
OVERVIEW	4
LAB 1 - WORKING WITH MAINFRAME USING COBOL AND DB2 (90 - 120 MINUTES).....	5
SECTION 1 – CONNECT TO A Z/OS SYSTEM	6
____ 1.1 WORKING WITH THE IDZ WORKSPACE	6
____ 1.2 CONNECTING TO THE Z/OS REMOTE SYSTEM	8
SECTION 2 EXECUTE THE DB2/COBOL BATCH PROGRAM AND VERIFY THE ABEND.....	10
____ 2.1 SUBMIT A COBOL/DB2 BATCH TO EXECUTE	11
____ 2.2 ADD Z/OS RESOURCES TO THE MVS SUBPROJECT	14
____ 2.3 SUBMIT A JCL TO EXECUTE THE COBOL PROGRAM	16
SECTION 3. USE FAULT ANALYZER TO IDENTIFY THE CAUSE OF THE ABEND	20
____ 3.1 USING FAULT ANALYZER PERSPECTIVE	21
SECTION 4. USING THE IBM Z/OS DEBUGGER FOR A TEMPORARY FIX	31
____ 4.0 BE SURE THAT IDZ CLIENT IS LISTENING ON PORT 8001.....	31
____ 4.1 SUBMIT THE JCL TO INVOKE THE DEBUG	31
____ 4.2 (OPTIONAL) USING THE VISUAL DEBUGGER FOR STACK PATTERN BREAKPOINTS	42
____ 4.3 ACCESSING THE Z/OS JES	48
SECTION 5. MODIFY THE COBOL CODE TO FIX THE BUG.....	52
____ 5.1 USING THE EDITOR WITH Z/OS COBOL PROGRAMS	52
____ 5.2 FIXING THE PROGRAM REGI0B	62
____ 5.3 COMPILE AND LINK REGI0B.....	67
____ 5.4 EXECUTE THE COBOL/DB2 PROGRAM	73
SECTION 6. USING THE CODE COVERAGE	77
____ 6.1 CREATING A JCL TO RUN THE CODE COVERAGE.....	77
____ 6.2 SUBMIT THE JCL FOR Z/OS EXECUTION	81
SECTION 7. (OPTIONAL) EXECUTING SQL STATEMENTS WHEN EDITING THE PROGRAM.....	88
____ 7.0 STARTING IDZ VERSION 14 AND CONNECT TO Z/OS.....	88
____ 7.1 CREATING A CONNECTION TO THE DB2 ON Z/OS	90
____ 7.2 RUNNING A SQL QUERY FROM COBOL PROGRAM.....	92
____ 7.3 USING THE SQL OUTLINE VIEW	100
____ 7.4 USING SQL VISUAL EXPLAIN	101
____ 7.5 REVERSE ENGINEER THE QUERY	103
____ 7.6 DISPLAYING DB2 TABLE CONTENT	106
SECTION 8. (OPTIONAL) USING FILE MANAGER.....	110
____ 8.1 VERIFY THAT YOU ARE CONNECTED TO THE PDTTOOLS COMMON COMPONENTS	110
____ 8.2 USING VIEW LOAD MODULE UTILITY	111
____ 8.3 WORKING WITH Z/OS DATA SETS	115
____ 8.4 WORKING WITH TEMPLATES.....	136
LAB 2A – (OPTIONAL) USING IBM ZUNIT TO UNIT TEST A COBOL CICS/DB2 PROGRAM USING REMOTE ASSETS (60 MINUTES).....	146
OVERVIEW OF DEVELOPMENT TASKS	146
SECTION 1. GET FAMILIAR WITH THE APPLICATION USING THE 3270 TERMINAL	147
____ 1.1 CONNECT TO Z/OS AND EMULATE A CICS 3270 TERMINAL.....	147
____ 1.2 RUN CICS TRANSACTION HCAZ	151
SECTION 2 – IMPORT A Z/OS PROJECT	154
____ 2.1 IMPORTING THE LAB6B Z/OS PROJECT	154
SECTION 3 – RECORD DATA INTERACTION USING THE CICS APPLICATION.....	158
____ 3.1 UNDERSTANDING THE COBOL PROGRAM THAT READS FROM DB2 TABLE.....	158
____ 3.2 RECORDING THE COBOL PROGRAM THAT SENDS THE MESSAGE	160
SECTION 4. GENERATE, BUILD AND RUN THE UNIT TEST.....	173
____ 4.1 GENERATING, BUILDING AND RUNNING THE TEST CASE PROGRAM.....	173
____ 4.2 RUNNING ZUNIT TEST CASE WITH CODE COVERAGE	176
SECTION 5. INTRODUCE A BUG IN THE PROGRAM AND RERUN THE UNIT TEST.....	179
____ 5.1 MODIFYING THE PROGRAM AND INTRODUCE A BUG	179
____ 5.2 REBUILDING THE CHANGED PROGRAM WITHOUT DEPLOYING TO CICS	180

5.3 RERUNNING THE TEST CASE	181
5.4 RUNNING zUNIT TEST CASE WITH DEBUGGING	183
SECTION 6. RUN THE UNIT TEST FROM A BATCH JCL.....	188
6.1 RUNNING THE UNIT TEST FROM A BATCH JCL	188
LAB 2B – (OPTIONAL) USING IBM ZUNIT TO UNIT TEST A COBOL/DB2 BATCH PROGRAM (60 MINUTES).....	192
OVERVIEW OF DEVELOPMENT TASKS.....	192
PART #1 – UNIT TEST ON PROGRAM DB2BATCH AND INTRODUCE A BUG.....	193
SECTION 1. RUN THE COBOL/DB2 BATCH PROGRAM USING JCL.....	193
1.0 CONNECT TO z/OS USING IDz	193
1.1 SUBMIT A PROVIDED JCL FOR EXECUTION	194
SECTION 2 – USE zUNIT TO RECORD THE BATCH EXECUTION.....	197
2.1 UNDERSTANDING THE MAIN COBOL PROGRAM THAT READS FROM DB2 TABLE	197
2.2 RECORDING THE BATCH JCL EXECUTION	199
SECTION 3. GENERATE, BUILD, AND RUN THE UNIT TEST GENERATED PROGRAM.....	205
3.1 GENERATING THE COBOL TEST CASE PROGRAMS.....	205
3.2 GENERATE, BUILD, AND RUN THE UNIT TEST GENERATED PROGRAM.....	206
3.3 RUNNING THE TEST CASE,	209
3.4 VERIFY THE JCL SUBMITTED THAT RUNS THE TEST CASE	211
SECTION 4. MODIFY THE COBOL/DB2 PROGRAM (INTRODUCE A BUG) AND RERUN THE UNIT TEST	212
4.1 MODIFYING THE PROGRAM AND INTRODUCE A BUG	212
4.2 RE-COMPILe AND LINK THE CHANGED PROGRAM	214
4.3 RUNNING THE TEST CASE AGAIN	217
SECTION 5. RUN THE BATCH PROGRAM AND VERIFY THE BUG	219
5.1 VERIFY THE BUG ON THE PRINTED REPORT	219
SECTION 6. USE IDz TO FIX THE BUG, RECOMPILE THE COBOL/DB2 PROGRAM.....	221
6.1 MODIFYING THE PROGRAM TO ELIMINATE THE BUG	221
6.2 RE-COMPILe AND LINK THE CHANGED PROGRAM	222
SECTION 7. RERUN THE zUNIT AND VERIFY THAT THE BUG IS ELIMINATED.....	224
7.1 RUNNING THE TEST CASE AGAIN	224

Overview

- Integrated application development and problem analysis ([ADFz](#))
- Managing your source code using modern tools (including [Git](#))
- Building automation ([DBB](#)) for COBOL or PL/I without a specific source code manager or pipeline automation tool (including [Groovy](#) and [Jenkins](#))
- Using a unit testing framework ([zUnit](#)) for COBOL or PLI as part of the development environment
- Using IBM Z Virtual Test Platform ([VTP](#)) for application Integration Testing of a COBOL/CICS/DB2 application
- Using [GitLab CI](#) along with DBB, ZUnit and UrbanCode Deploy (UCD) on z/OS.
- Integration testing for z/OS powered hybrid cloud applications using [Galasa](#)
- Application deployments automation to many environments ([UCD](#))
- Expose Mainframe legacy applications via RESTful APIs ([z/OS Connect](#))
- Understand how Mainframe applications use their resources to improve performance ([APA](#))

This material was built using a Windows on cloud that was updated on **August 2021**
 Here the desktop background of this image:



The following symbols appear in this document at places where additional guidance is available.

Icon	Purpose	Explanation
	Important!	This symbol calls attention to a particular step or command. For example, it might alert you to type a command carefully because it is case sensitive.
	Information	This symbol indicates information that might not be necessary to complete a step, but is helpful or good to know.
	Trouble-shooting	This symbol indicates that you can fix a specific problem by completing the associated troubleshooting information.

LAB 1 - Working with mainframe using COBOL and DB2

(90 - 120 minutes)

Updated September 24, 2021 (created by [Regi](#), Reviewed by [Wilbert](#))

This lab will take you through the steps of using the [Application Delivery Foundation for z Systems](#) (ADFz) to work with a z/OS system. It will familiarize you with some of the capabilities of this product using a DB2 COBOL batch program that is ABENDING.

On this lab you will connect to a remote z/OS system, submit and execute a program (which ABENDs), identify the program abend, set up a MVS project, edit, compile, and debug a COBOL application. The process would be similar for a PL/I program.

Overview of development tasks

To complete this tutorial, you will perform the following tasks:

- 1. Connect to a z/OS System.**
→ You will connect to the z/OS system using a provided z/OS logon userid.
- 2. Execute the DB2/COBOL batch program and verify the ABEND.**
→ You will submit a COBOL/DB2 batch to execute and verify the bug.
- 3. Use Fault Analyzer to identify the cause of the ABEND**
→ You will use Fault Analyzer to identify what is causing the ABEND.
- 4. Use the IBM Debug for a temporary fix**
→ You will modify the field content to bypass the bug
- 5. Modify the COBOL code to fix the bug.**
→ You will be able to fix the bug changing the COBOL code.
- 6. Use Code Coverage**
→ You can now verify program areas covered by the test case with Code Coverage
- 7. (Optional) Execute SQL statement when editing the program.**
→ While editing the COBOL/DB2 program you will be able to execute SQL statements and verify the query results.
- 8. (Optional) Using File Manager**
→ An example of using File Manager against a z/OS data set.
File Manager provides formatted editors and viewers. It also provides a full complement of on-line and batch utilities to copy, extract, and load data, to create files and databases, compare and print, and many other utility functions.

Section 1 – Connect to a z/OS System

You will connect to the z/OS. This section is like LOGON to a TSO. You will use **empot01** as userid and password.

1.1 Working with the IDz workspace

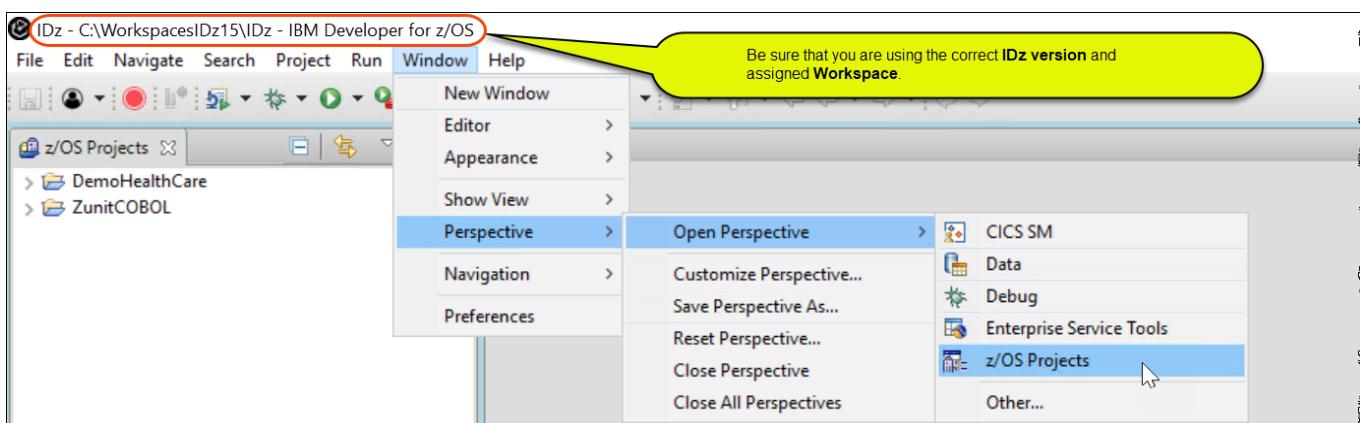
1.1.1 Start IBM Developer for z Systems version 15

- ▶ Using the windows desktop double click on **IDz V15** icon.
- ▶ Verify that the message indicates that it is **Version 15.0.1**

IMPORTANT -> This icon will start an eclipse workspace that has already some definitions required for this lab.
PLEASE DO NOT start IDz using other way than this icon.



1.1.2 ▶ Open the **z/OS Projects** perspective by selecting **Window > Perspective > Open Perspective > z/OS Projects**



z/OS Projects perspective

Use the z/OS Projects perspective to define the connection, connect to, and work with remote systems, and to create, edit, and build projects, subprojects, and files on remote UNIX, Linux for z Systems, and z/OS systems.

The z/OS Projects perspective contains the many views like Remote Systems view, z/OS Projects view, Properties view, Outline view, Remote Error List view, z/OS File System Mapping view, Remote System Details view, Team view, Property Group Manager view and Snippets view



You can close and open views to customize the perspective.

To open one of the views that were closed:

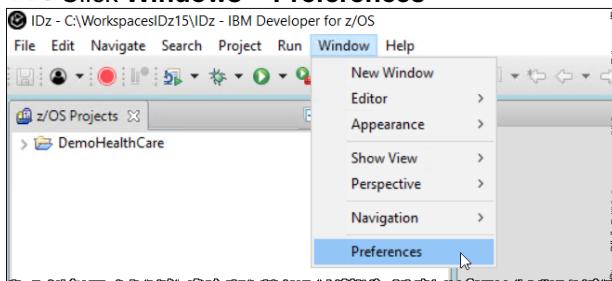
1. In the workbench, select **Window > Show View**.

A menu that lists the views associated with the z/OS Projects perspective is displayed.

2. Click the name of the view you want to open.

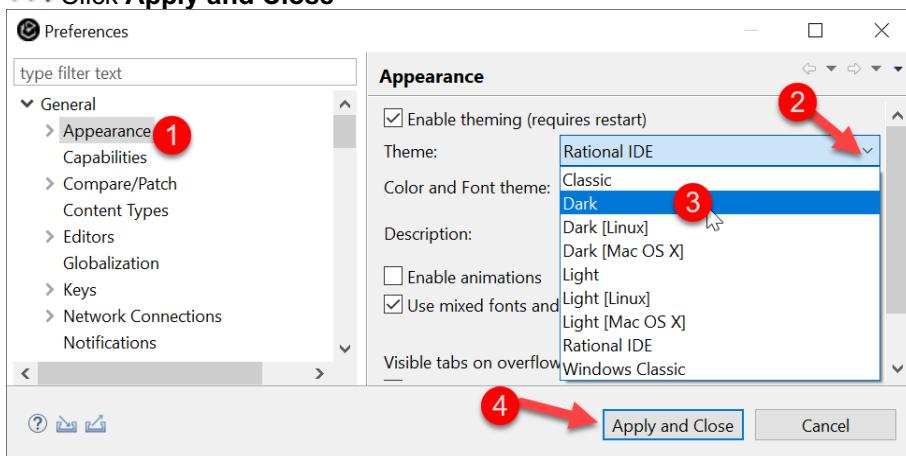
1.1.3 Starting on version 15 the developer can change the workspace appearance. If you want to try that

▶ Click **Windows > Preferences**



1.1.4 ▶ Under **General** select **Appearance** and on **Theme** select **Dark**.

▶ Click **Apply and Close**

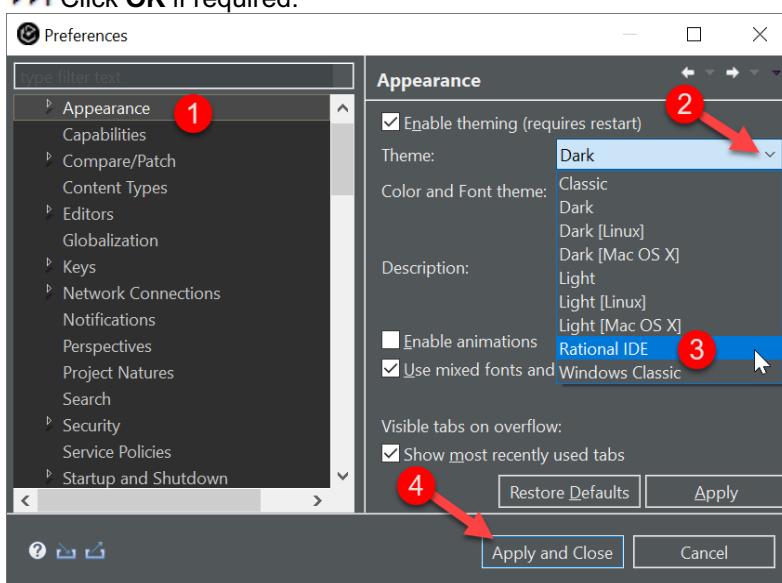


1.1.5 Since all picture here were done using “Rational IDE” theme we recommend to return to the default.

▶ Again, click **Windows > Preferences**

▶ Under **General** select **Appearance** and on **Theme** select **Rational IDE** and click **Apply and Close**

▶ Click **OK** if required.



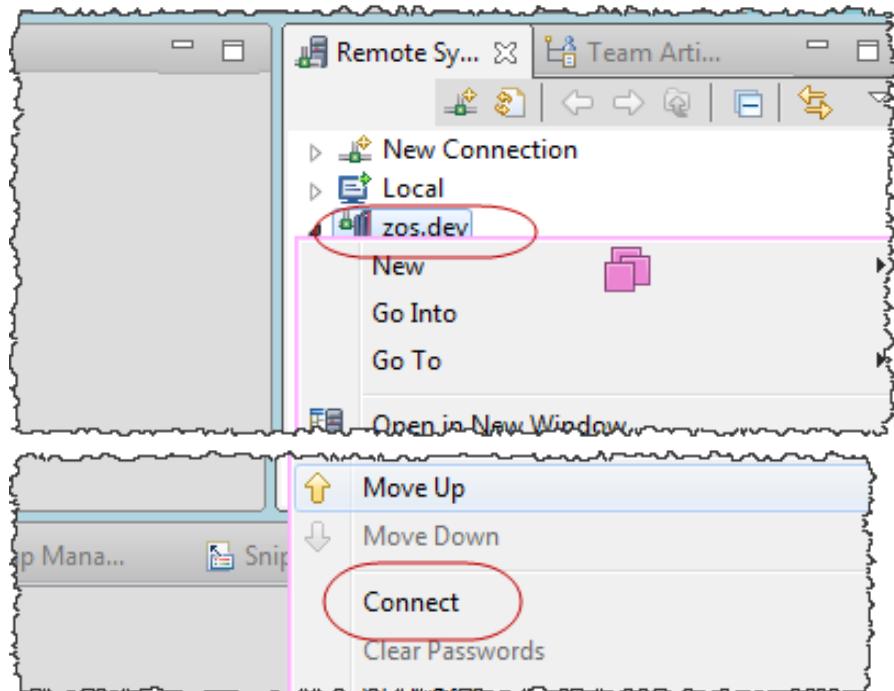
1.2 Connecting to the z/OS Remote system

1.2.1 To Connect to the z/OS system:

► Using the *Remote Systems* view on the top and right side of the screen:

Right-click on **zos.dev** and select **Connect**

Notice: Since you are using a cloud instance the response to the right click may be delayed first time.

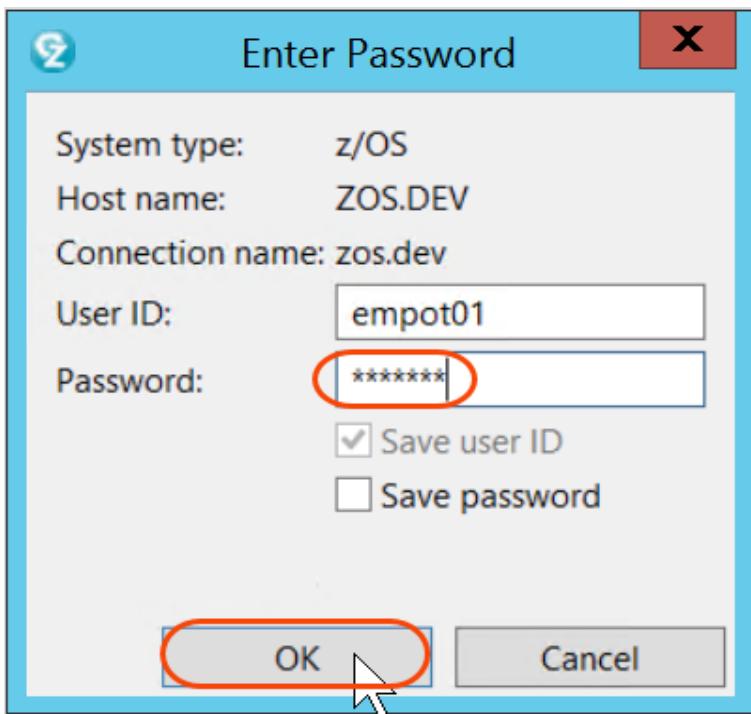


1.2.2 You will be prompted for your z/OS userid and password.

► Type **empot01** as userid (might be already there) and **empot01** as password.

The userid and password **can be any case; don't worry about having it in UPPER case.**

1.2.3 ► Click **OK** to connect to z/OS.



Be Patient! The connection could take a while depending on the network condition.

► Wait until connection is complete (look at the bottom and left until the green bar  disappears)

Notice that some folder icons changed after connected. A small **green arrow**  is added.



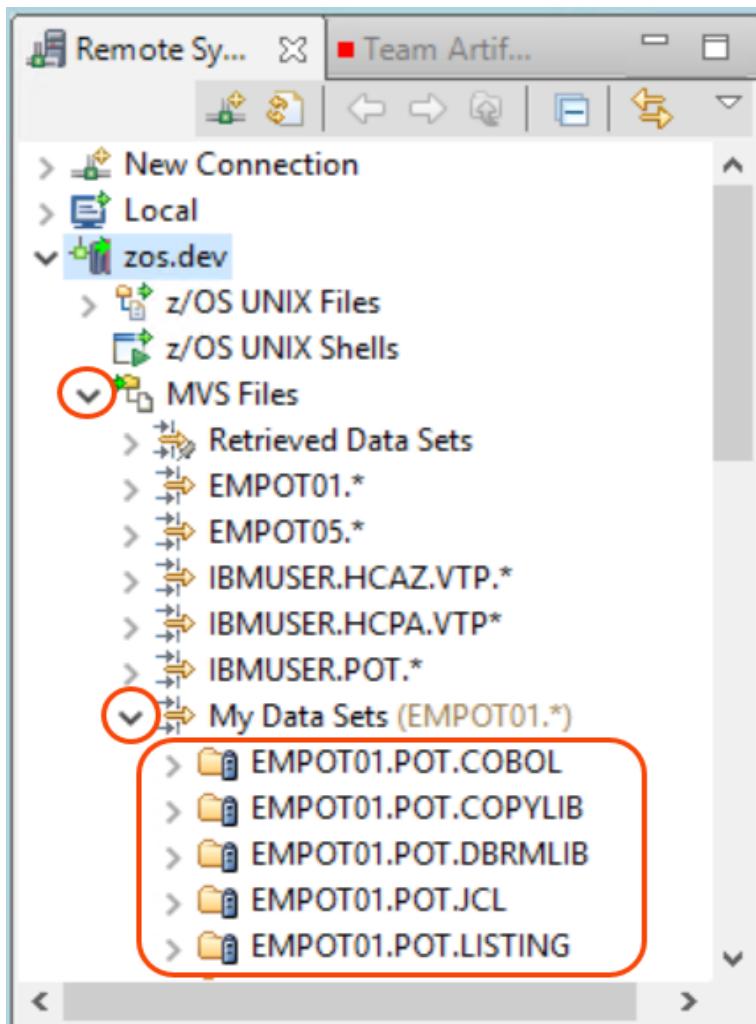
In case of connection errors...

If you have errors during the connection it is because the z/OS system is not available.
You also may have network issues.



An easy way to find if z/OS is up is opening a 3270 emulation clicking on the icon  that is on the base of your windows client. (zos.dev is not "pingable").
If you do not receive a reply, Contact the instructor. Your connection is broken.

1.2.5 ► Expand  MVS Files and  My Data Sets to see MVS data sets that starts with EMPOT01.*



Remote Systems view

The Remote Systems view shows all existing connections to remote systems. Connections are persisted, containing the information needed to access a remote host. The view contains a prompt to create new connections, and pop-up menu actions to rename, copy, delete, and reorder existing connections. Connections contain attributes, or data, that are saved between sessions of the workbench. These attributes are the connection name, the remote system's host name and system type, an optional description, and a user ID that is used by default by each subordinate subsystem, at connection time.



Section 2 Execute the DB2/COBOL batch program and verify the ABEND.

To make your job easier, you will group together all the assets that you will work with. This is a new development concept for TSO/ISPF users, since TSO/ISPF does not provide such capability. To accomplish this task, you will create a **z/OS project** and select which assets will be part of this project.

What is a z/OS project?

After you define a z/OS-connection and connect to that system, you can define a z/OS project under that system. You can define the z/OS project, however, only when you are connected to the system. Application Delivery Foundation for z Systems assigns a set of default properties from the set of system properties. However, changes that you make to system properties do not affect the definition of an existing project. If you change your system properties to reference a new compiler release, for example, the reference affects only those projects that are defined subsequent to the change. This isolation of system data from existing projects is beneficial because it lets you develop your code without disruption. You can introduce changes to the project definition at a time of your choosing.



Creating a new MVS subprocess

MVS subprojects contain the development resources that reside on an MVS system. You can create multiple subprojects in a z/OS project.

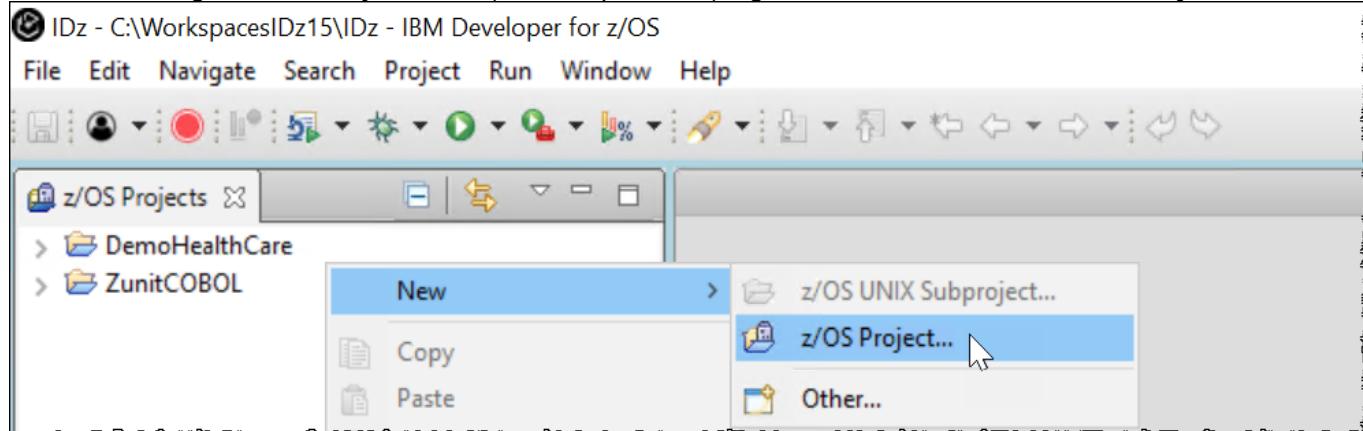
Before you create an MVS subproject, you need to have completed the following tasks:

- Connecting to a remote system
- Creating a z/OS project

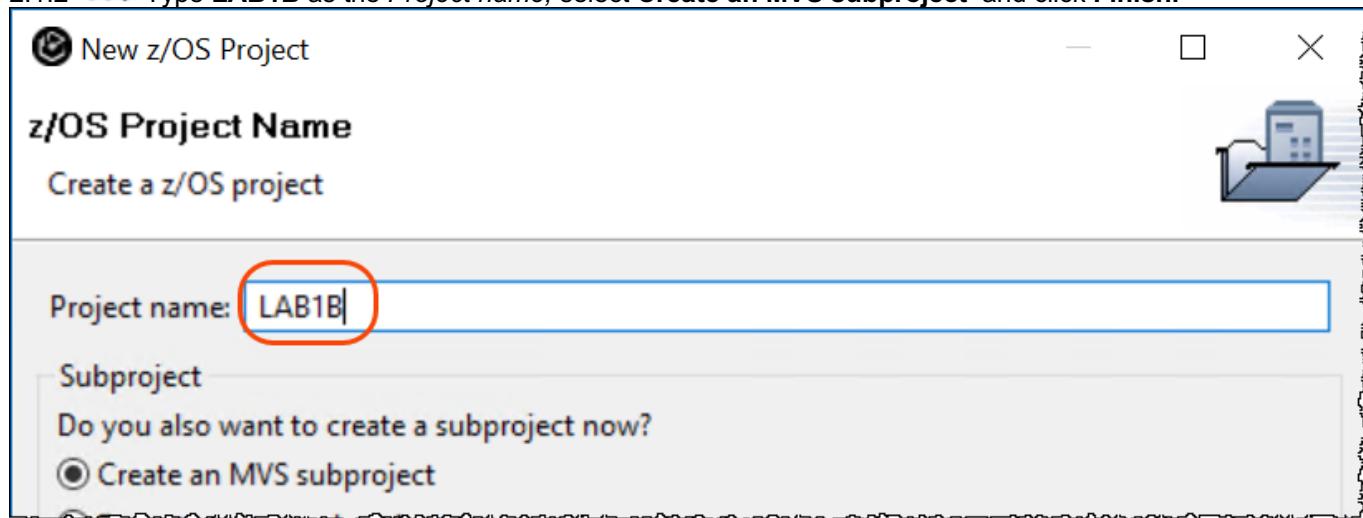
2.1 Submit a COBOL/DB2 batch to execute

The advantage of creating a z/OS Project is that we just focus on those datasets and members that are being constructed or updated, instead of having all the mainframe datasets or members. At any time if you need to see a dataset not added to the project, just go to the z/OS Systems view and add it. In addition, at any time, you can remove from your project the datasets that you don't need anymore.

2.1.1 Using the z/OS Projects view (on the top and left), right click and select New > z/OS Project...

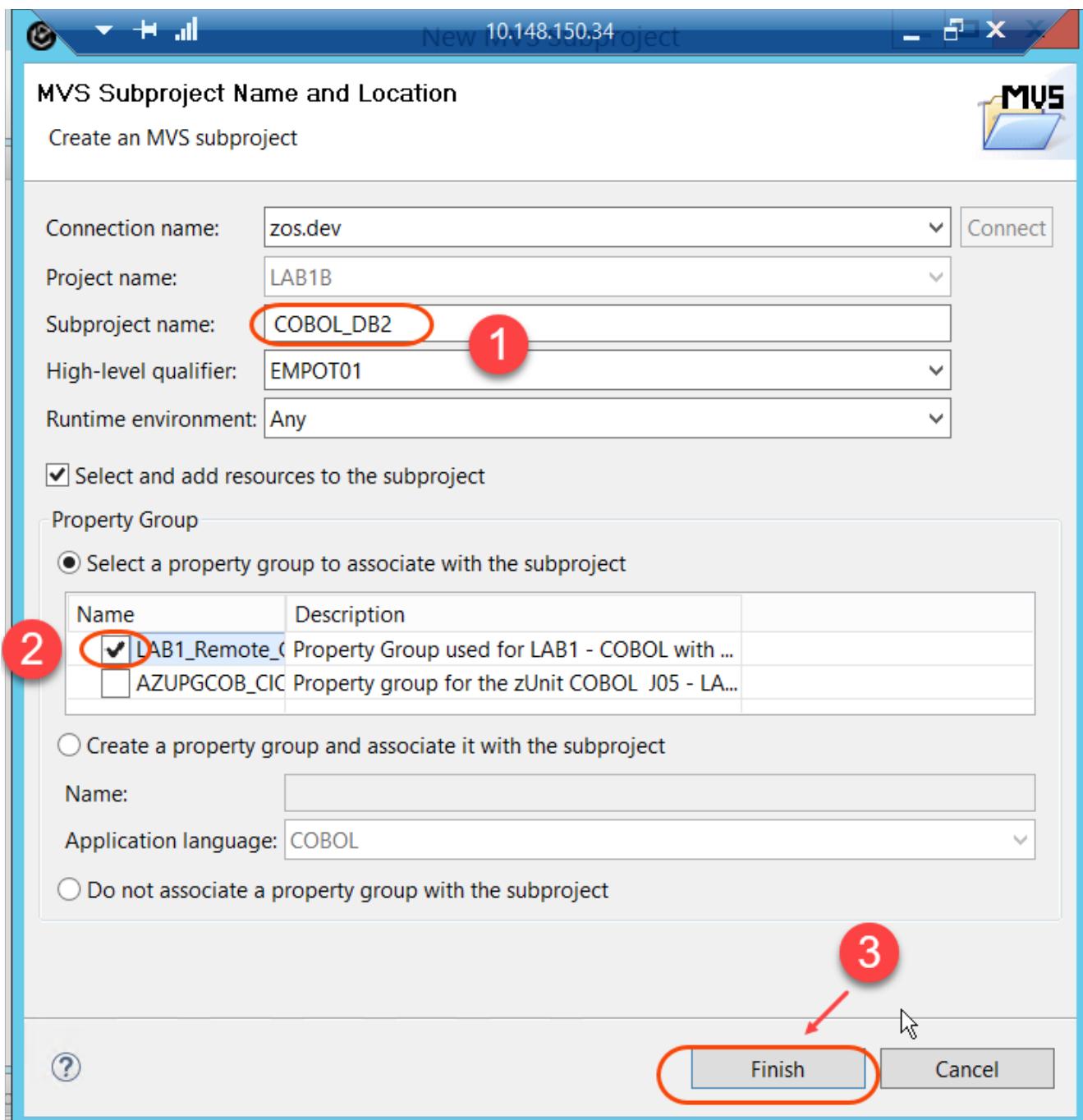


2.1.2 ➔ Type **LAB1B** as the *Project name*, select **Create an MVS subproject** and click **Finish**.



2.1.3 ➔ Type **COBOL_DB2** as *Subproject Name*, select **LAB1_Remote_Cobol..** as the property group and click **Finish**.

TIP: If using *Spanish or Brazilian* keyboard you must activate the correct language in the bottom, right corner, otherwise you will not be able to generate the “_”.



This dialog will be displayed:

Add Resources to Subproject

Select the data sets or members you want to add to the subproject

- > EMPOT01.*
- > EMPOT05.*
- > IBMUSER.HCAZ.VTP.*
- > IBMUSER.HCPA.VTP*
- > IBMUSER.POT.*
- > Retrieved Data Sets
- > My Data Sets

Can't you find the property group above?

It is because you are using a wrong workspace. Close IDz, go back to the step 1.1.1 and be sure you start IDz from the icon that is on the windows desktop.

PROPERTY GROUP

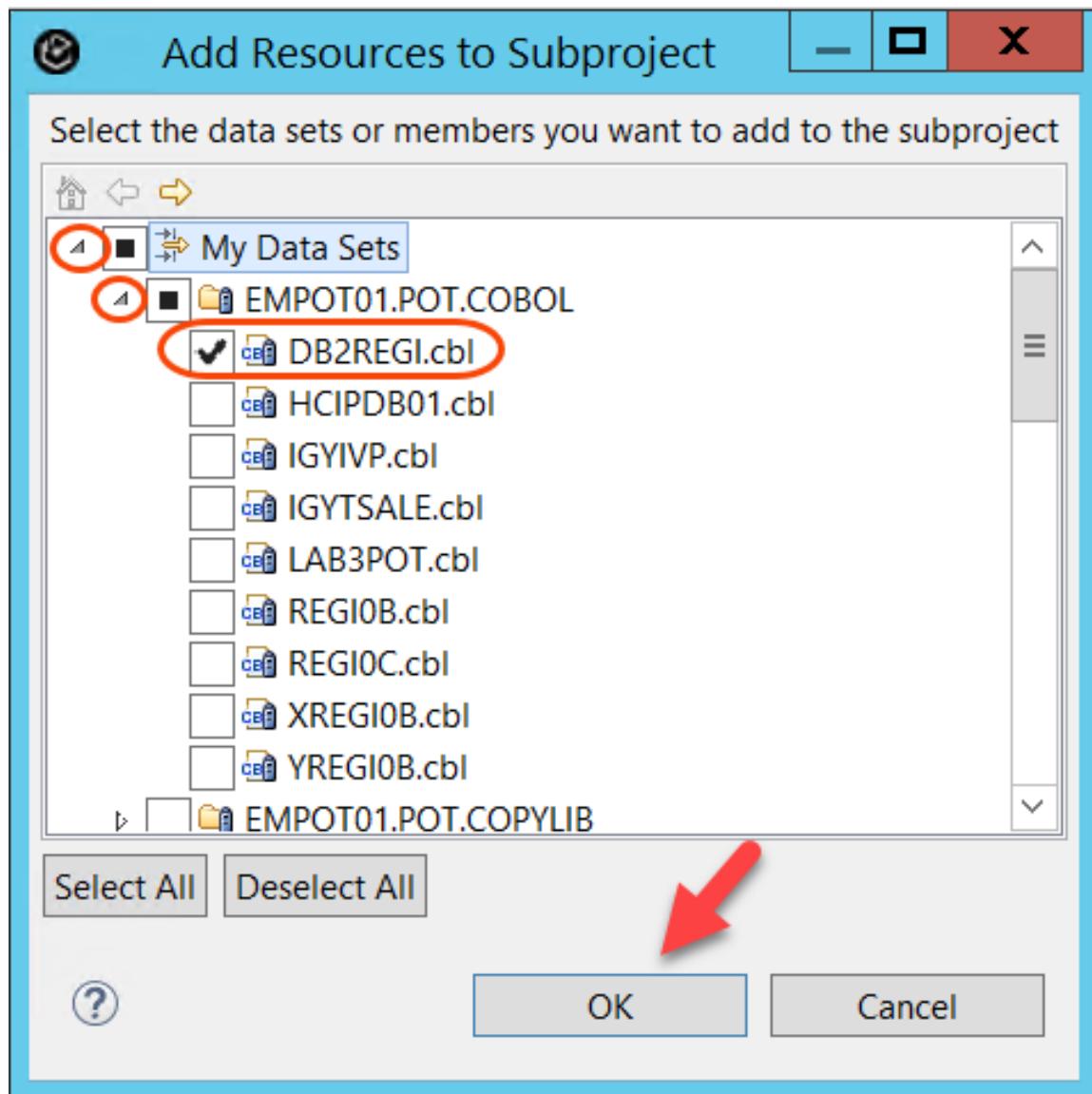
Property groups provide a way to manage resource properties, share them easily across systems, projects, resources, and users, and maintain consistency in your development and build environment.

You can, for example, define a property group that points to copybooks. This is something like //STEPLIB; otherwise, some of the variables used in the program are not resolved.

2.2 Add z/OS resources to the MVS subproject

To make the data sets available to your remote project named *LAB1B*, you will need to add them. For this lab we just want to add one member..

- 2.2.1 On the dialog below, expand **My Data Sets**, **EMPOT01.POT.COBOL** , select **DB2REGL.cbl**, and click **OK**

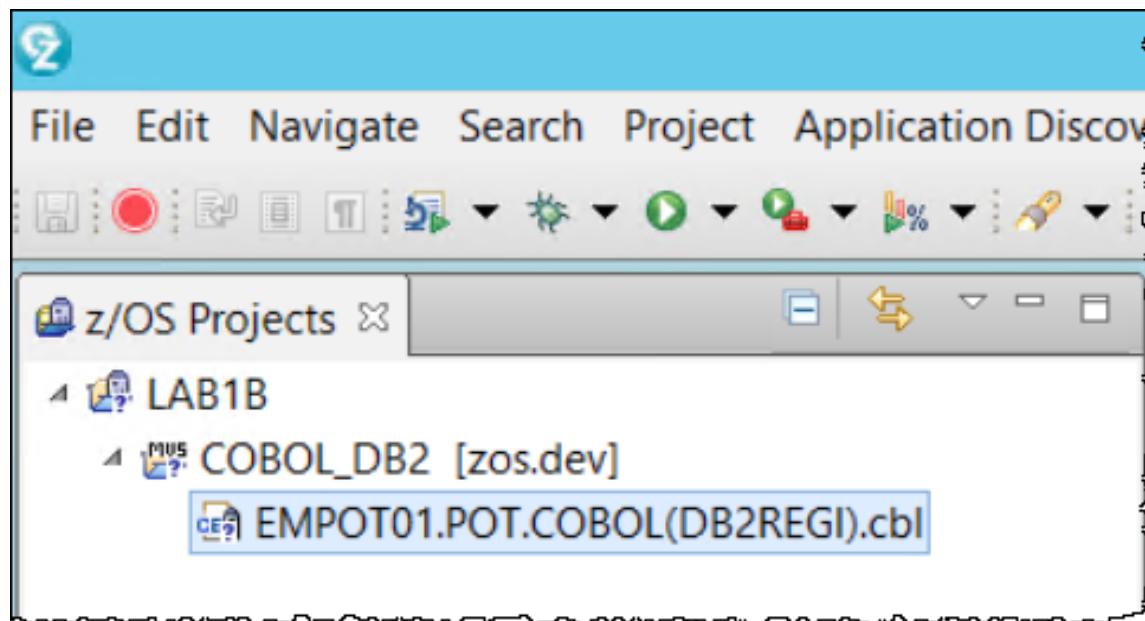


2.2.2 You should see a z/OS Project named **LAB1B** in your *z/OS Projects view*.

► Click **z/OS Projects view** (on left) and under **COBOL_DB2**, you will now see **DB2REGI** member added to your project.

Notice that creating a Remote project is a good practice to isolate the code you are working on, but you could work on your code without creating this project.

Our lab involves a small update so you could work directly on the *Remote System* perspective without having to create a Remote project.



2.3 Submit a JCL to execute the COBOL program

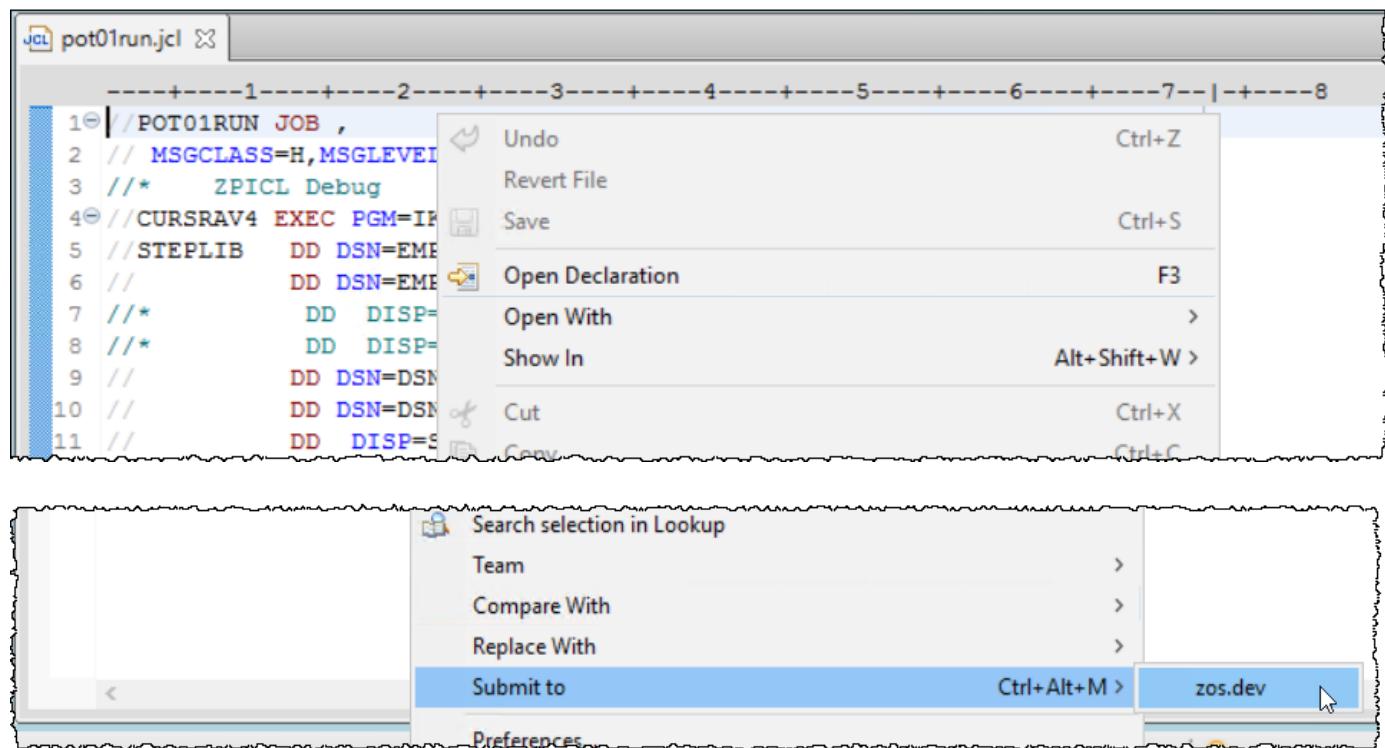
The JCL to execute the program is available on the local windows folder: "C:\ADF_POT\empot01". You will use this JCL to execute a batch COBOL/DB2 on z/OS.

2.3.1 ► Using *Remote System View* (on top and right), **scroll up** to locate the file **pot01run.jcl** under **Local/Local Files/ADF_POT/empot01** and **double click** to edit.

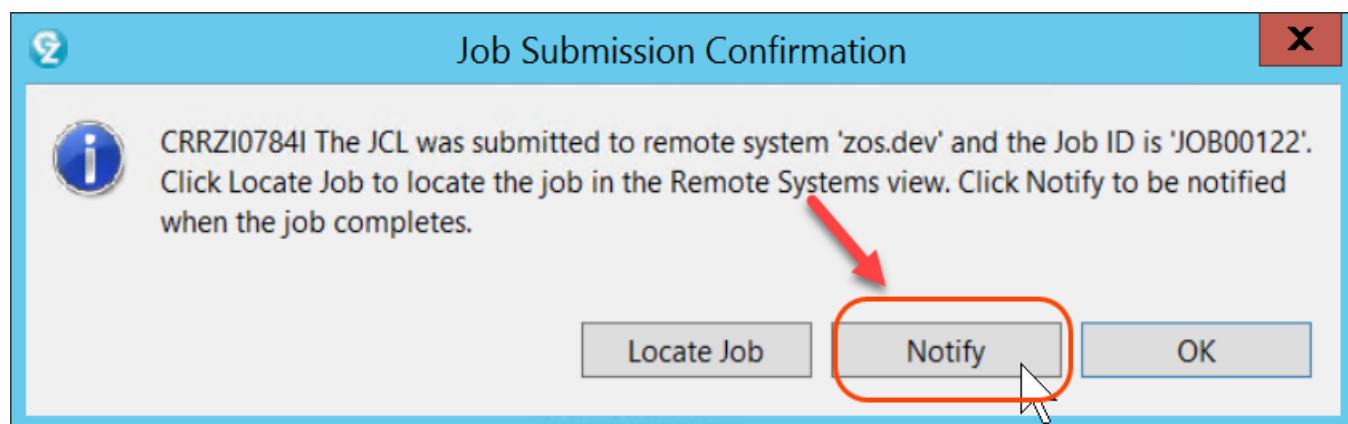
The screenshot shows the IBM Rational Application Developer interface. On the left, a code editor window titled "pot01run.jcl" displays a COBOL JCL script. The script defines a job named "POT01RUN" with various parameters like MSGCLASS, MSGLEVEL, TIME, REGION, and COND. It includes EXEC PGM=IKJEFT01, DD statements for STEPLIB and EMPOT, and various DISP=SHR options. A yellow callout bubble points to the file in the editor with the text: "Be sure that you are editing pot01run.jcl (NOT pot01db.jcl)". On the right, a "Remote Systems" view shows a file tree under "Local". The "Local Files" section contains several sub-directories and files, including "ADF_POT", "empot01" (which contains "alloc01.jcl", "pot01db.jcl", and "pot01run.jcl" - the latter is circled in red), and other directories like "empot02", "empot05", "ibmuser", and various LAB sub-directories. A red arrow points from the callout bubble towards the "pot01run.jcl" file in the file tree.

```
1 //> POT01RUN JOB ,  
2 // MSGCLASS=H,MSGLEVEL=(1,1),TIME=1440,REGION=70M,COND=(16,LT)  
3 /* ZPICL Debug  
4 //CURSRAV4 EXEC PGM=IKJEFT01,DYNAMNBR=20  
5 //STEPLIB DD DSN=EMPOT01.POT.LOAD,DISP=SHR  
6 // DD DSN=EMPOT.ZPICL.LOAD,DISP=SHR  
7 /* DD DISP=SHR,DSN=IBMUSER.VER1.SEQAMOD  
8 /* DD DISP=SHR,DSN=IBMUSER.VER1.SEQAAUTH  
9 // DD DSN=DSNB10.SDSNLOAD,DISP=SHR  
10// DD DSN=DSNB10.DBDBG.RUNLIB.LOAD,DISP=SHR  
11// DD DISP=SHR,DSN=FEK910.SFEKAUTH  
12//SYSPRINT DD SYSOUT=*<br/>  
13//SYSOUT DD SYSOUT=*<br/>  
14//SYSTSPRT DD SYSOUT=*<br/>  
15//SYSTSIN DD *  
16 ISOLIB ACTIVATE DA('DSNB10.SDSNLOAD')  
17 DSN SYSTEM(DBDBG)  
18 RUN PROGRAM(DB2REGI) PLAN(DB2REGI) -  
19 LIB('EMPOT.ZPICL.LOAD')  
20 END
```

2.3.2 ►| Right click on the JCL being edited and select **Submit to → zos.dev**



2.3.3 ►| When the dialog pop up appears, click **Notify**. This allows you to be notified when the execution is ended.



2.3.4 Under *Remote Console*, you will be notified when execution is completed.

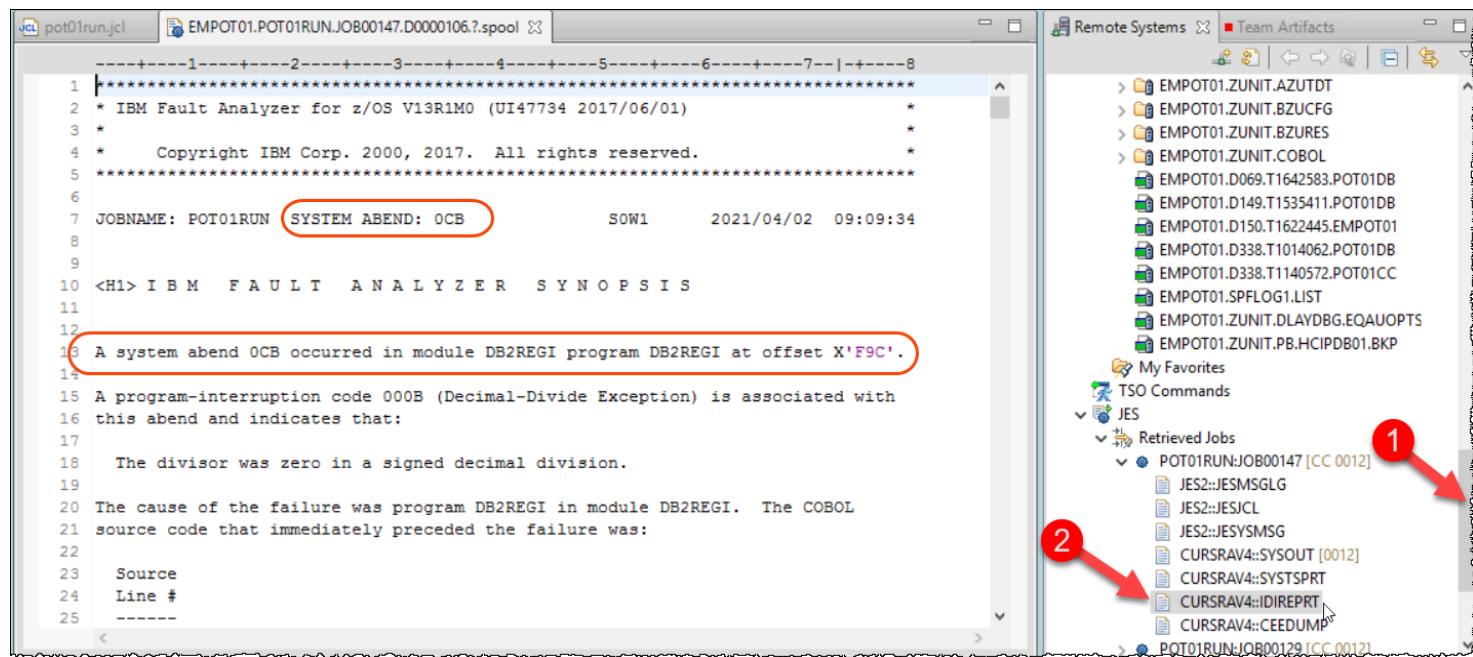
►| Once the execution ends, **click on the link POT01RUN:JOB00xxx** (where 00xxx is a number) to point to the Job output. This number vary depending on z/OS.

```
19 LIB('EMPOT.ZPICL.LOAD')
20 END
21
[5/24/19, 8:50 AM] Job JOB00123 is being submitted
[5/24/19, 8:51 AM] Job POT01RUN:JOB00123 ended with completion code CC 0012
```

Not that the return code must be 12

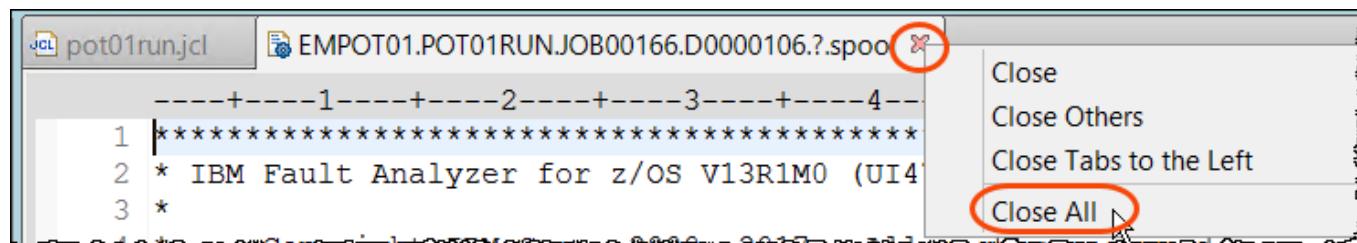
2.3.5 ► Under **Remote Systems** view **scroll down**, expand **JES > Retrieved Jobs > POT01RUN:JOB00xxx** and **double click CURSRAV4::IDIREPRT** step and you will see the *Fault Analyzer* report showing an **OCB ABEND**. Your mission is to fix that bug.

Tip: If there is no jobs under “Retrieved Jobs”, is because you did not click on link as stated at 2.3.4. You can also see this output once you right click on “My Jobs” under *JES* and select **Refresh**.



2.3.6 ► Close all the opened editors using **Ctrl + Shift + F4**,

Or right click on the and choose **Close all**.



Section 3. Use Fault Analyzer to identify the cause of the ABEND

You will now take advantage of **IBM Fault Analyzer** (that is part of Application Delivery Foundation- ADF) to verify the cause of the ABEND.

What is IBM Fault Analyzer for z/OS?

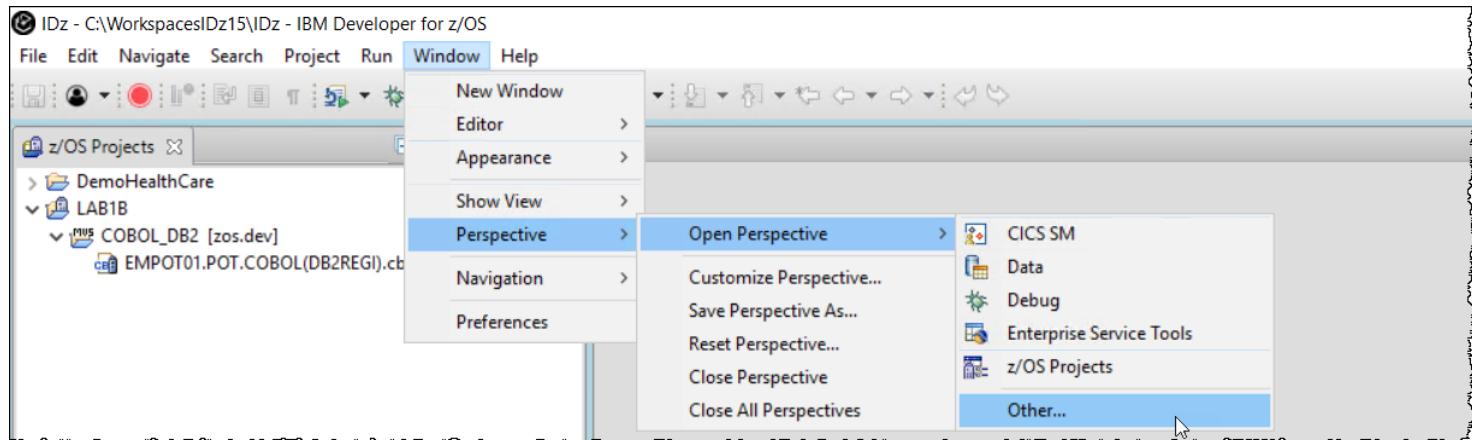


IBM Fault Analyzer for z/OS helps developers analyze and fix system and application failures for CICS, WebSphere MQ, IMS and DB2 environments. When an application ends abnormally, Fault Analyzer is automatically engaged, gathering real-time information about the event and its environment at the time of failure.

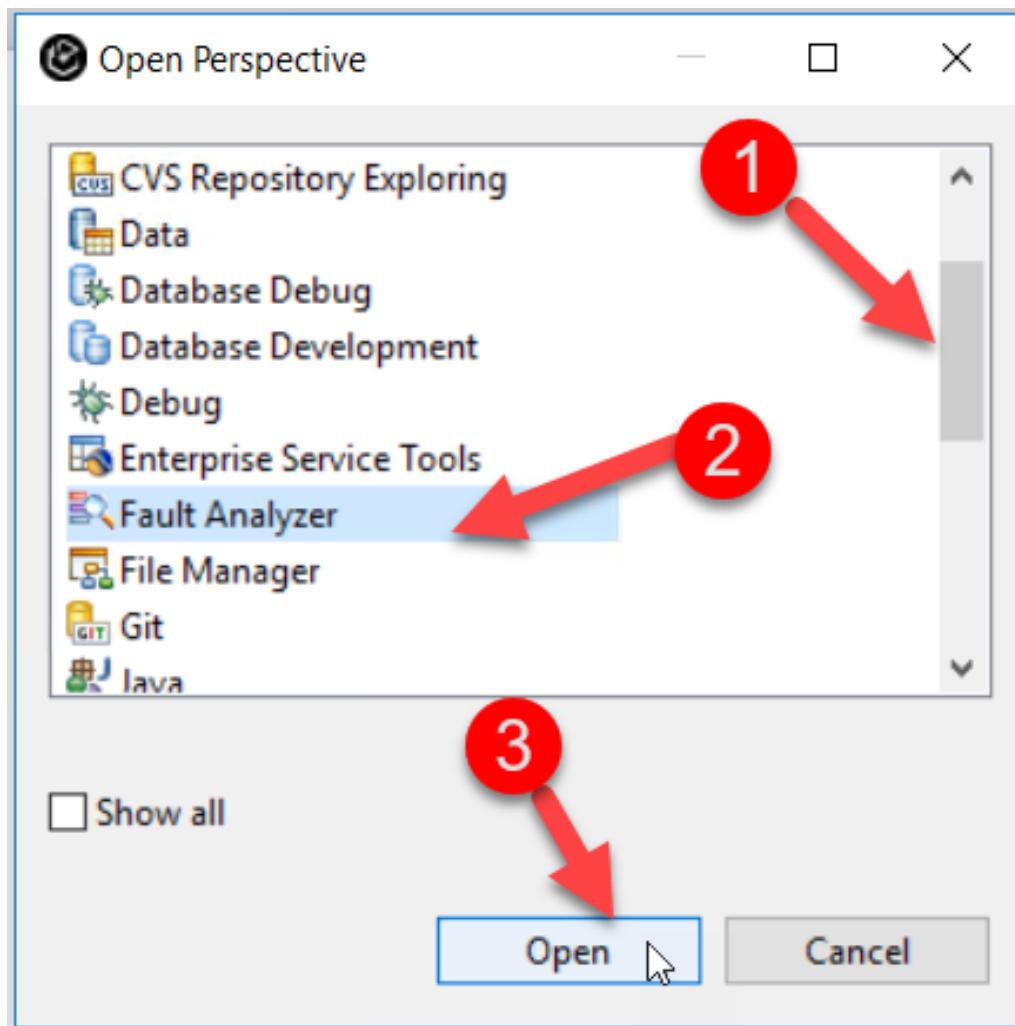
This helps the development team to identify the cause, analyze what went wrong and resolve the problem in a more timely and efficient manner to avoid costly interruptions that could jeopardize application schedules and outcomes.

3.1 Using Fault Analyzer perspective

3.1.1 ► Open the *Fault Analyzer* perspective using **Window > Perspective > Open Perspective > Other...**

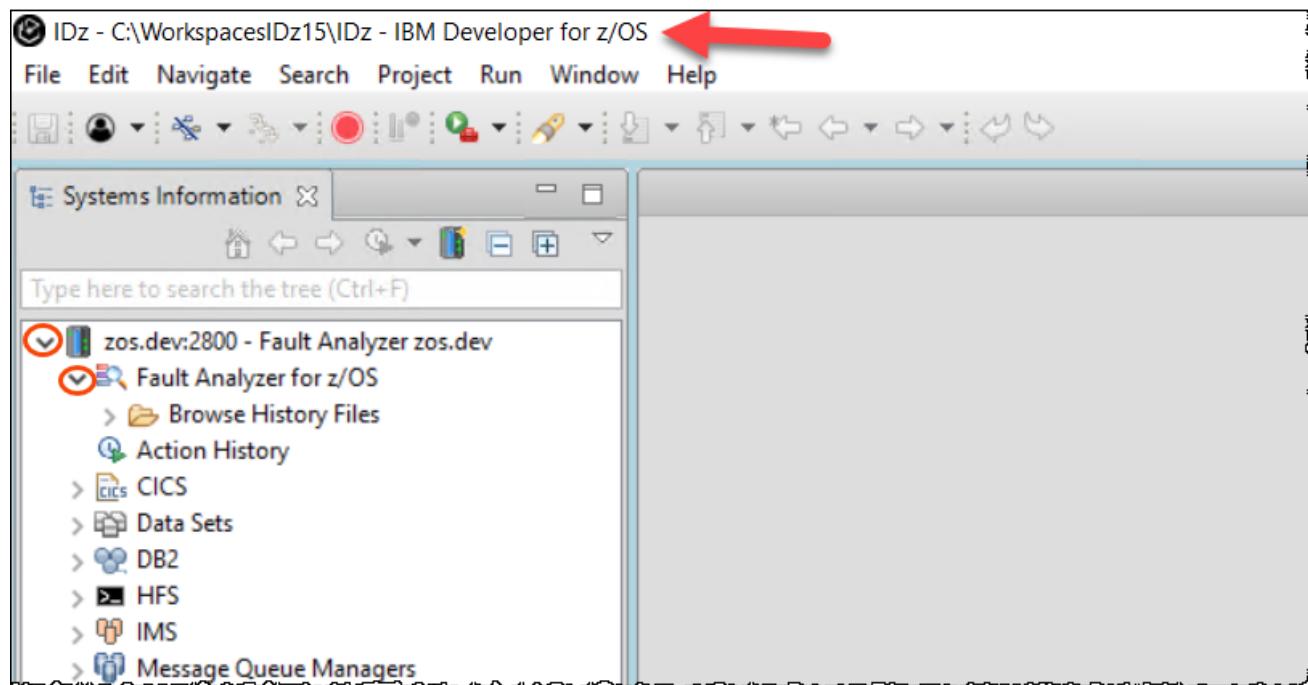


3.1.2 ► Scroll down, select **Fault Analyzer** and click **Open**

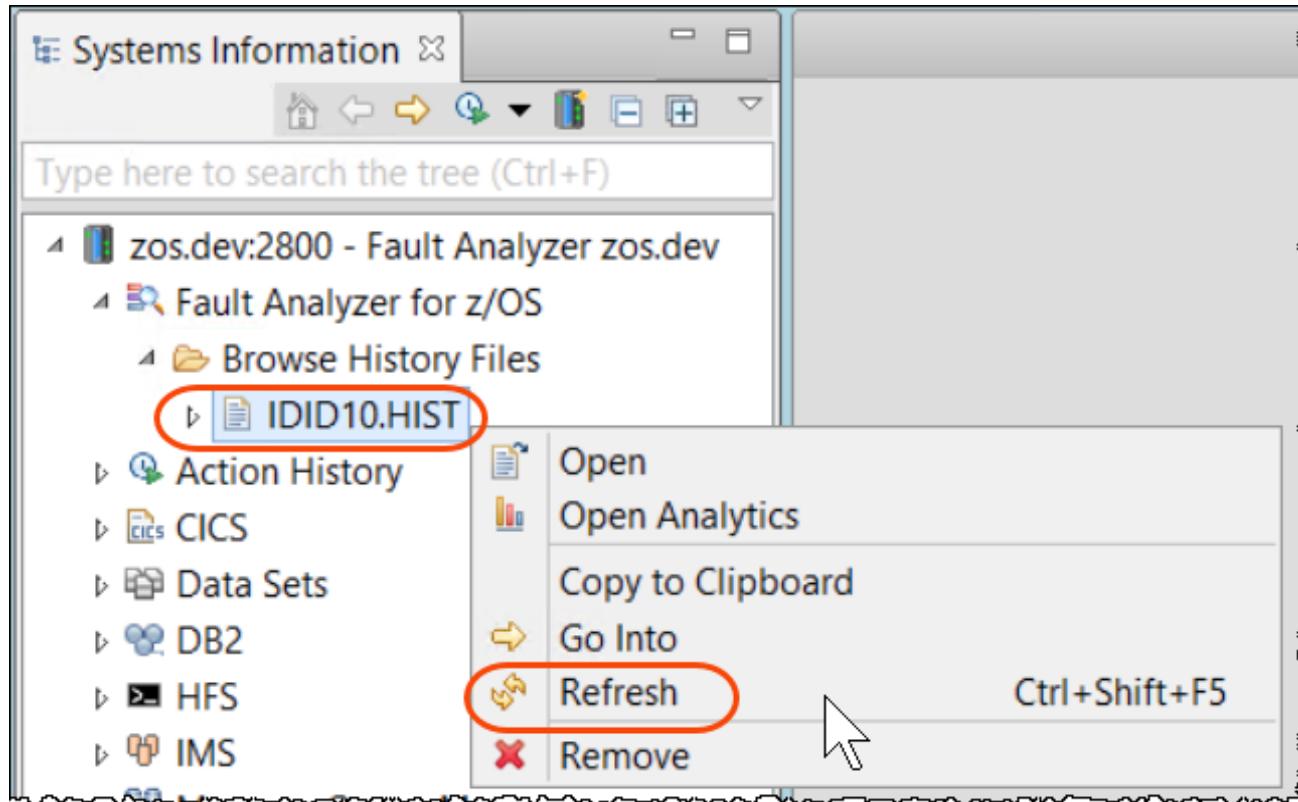


3.1.3 ► Under Systems Information view (on left) be sure that **Fault Analyzer for z/OS** is expanded

TIP: If you do not have this entry, it is because you are using a wrong IDz workspace. Please contact the instructor.



- 3.1.4 ► Right click **Fault Analyzer for z/OS** and select **Refresh**
- Expand **Browse History File**. You will see the folder **IDID10.HIST**
- Right click **IDID10.HIST** and select **Refresh** (or **Ctrl + Shift + F5**)



- 3.1.5 ► If you get a *Sign on* dialog for *Fault Analyzer* use **empot01** credentials and click **OK**
You see a list of *FAULT_IDs* on the z/OS system that you are connected to...

The screenshot shows the 'Systems Information' window of the Fault Analyzer for z/OS. On the left, a tree view displays nodes such as 'zos.dev:2800 - Fault Analyzer zos.dev', 'Fault Analyzer for z/OS', 'Browse History Files' (with 'IDID10.HIST' selected), 'Action History', 'CICS', 'Data Sets', 'DB2', 'HFS', 'IMS', and 'Message Queue Managers'. A red arrow points from the 'IDID10.HIST' node to the right panel. The right panel contains a table titled 'ZOS.DEV : 2800/IDID10.HIST' with the following data:

FAULT_ID	JOB/TRAN	USER_ID	SYS/JOB	ABEND	I_ABEND	JOB_ID	JOBNAME
F00302	POT01RUN	EMPOT01	S0W1	S0CB	S0CB	JOB00123	POT01RUN
F00282	HCMA	IBMUSER	CICSTS53	4038	4038	STC00045	CICSTS53

3.1.6 ► On the **ZOS.DEV:2800/IDID10.HIST** view locate the **latest** job name **POT01RUN** and **double click** on it.

You will see the report being downloaded from the z/OS to your Windows client. The report below will be displayed

The screenshot shows the IBM Fault Analyzer interface. On the left, there's a navigation tree under 'zos.dev:2800 - Fault Analyzer zos.dev' with nodes like 'Fault Analyzer for z/OS', 'Browse History Files', 'IDID10.HIST', 'Action History', 'CICS', 'Data Sets', 'DB2', 'HFS', 'IMS', and 'Message Queue Managers'. Below the tree is an 'Outline' panel with sections: Prolog, Summary, Synopsis, Event summary, and Event details. The main panel displays a fault report for 'POT01RUN'. The report header includes copyright information for IBM Fault Analyzer for z/OS V13R1M0 (UI47734 2017/06/01). It details a system abend (S0CB) at offset X'F9C' in module DB2REGI, program DB2REGI, source line # 365. A red arrow points to the 'Main Report' tab at the bottom of the report panel. The 'Event Details' tab is also visible. At the bottom, a table lists fault details:

FAULT_ID	JOB/TRAN	USER_ID	SYS/JOB	ABEND	I_ABEND	JOB_ID	JOBNAME	USERNAME
> F00353	POT01RUN	EMPOT01	S0W1	S0CB	S0CB	JOB00212	POT01RUN	
F00352	HCMA	IBMUSER	CICSTS53	4038	4038	STC00044	CICSTS53	

3.1.7 ► Scroll the report down and you will see that the field **RECEIVED-FROM-CALLED** used on the division has "0". So the abend is because of a divide by zero.

► Also notice that there are other tabs on this panel (like Event Details, Abend Information, etc.). You may try those tabs to get more information about the abend.

The screenshot shows the IBM System z Main Report view for a specific event. The top part displays the COBOL source code:

```

12
13 The cause of the failure was program DB2REGI in module DB2REGI. The COBOL
14 source code that immediately preceded the failure was:
15
16 Source
17 Line #
18 -----
19 000365 DIVIDE VALUE1 BY RECEIVED-FROM-CALLED GIVING RESULT
20
21 The COBOL source code for data fields involved in the failure:
22
23 Source
24 Line #
25 -----
26 000200      03 RECEIVED-FROM-CALLED      PIC 99.
27 000201      03 VALUE1                  PIC 99.
28 000206      03 RESULT                  PIC 99.
29
30 Data field values at time of abend:
31
32 RECEIVED-FROM-CALLED = 0
33 RESULT              = X'0000'
34 VALUE1              = 66
35
36

```

Red annotations include:

- A red circle labeled '1' with a red arrow pointing to the 'RECEIVED-FROM-CALLED' field in the source code.
- A red circle labeled '2' with a red arrow pointing to the 'Main Report' tab in the bottom navigation bar.
- A red circle labeled '3' with a red arrow pointing to the 'Line #' column header in the source code table.

The bottom navigation bar includes tabs: Main Report, Event Details, Abend Information, System-Wide Information, and Miscellaneous.

3.1.8 ► Using the *Main Report* view Click on the link **000365** This will show the COBOL statement that caused the *OCB abend*

The screenshot shows the IBM System z Systems Information view. On the left is a tree view of system resources, and on the right is the COBOL source code for line 000365:

```

28 source code that immediately preceded the failure was:
29
30   Source
31   Line #
32 -----
33 000365 DIVIDE VALUE1 BY RECEIVED-FROM-CALLED GIVING RESULT
34
35 The COBOL source code for data fields involved in the failure:
36
37   Source
38   Line #
39 -----
40 000200      03 RECEIVED-FROM-CALLED      PIC 99.
41 000201      03 VALUE1                  PIC 99.
42 000206      03 RESULT                  PIC 99.
43
44 Data field values at time of abend:
45
46 RECEIVED-FROM-CALLED = 0
47 RESULT              = X'0000'
48 VALUE1              = 66

```

A red arrow points from the 'Line #' column header in the main report to the 'Line #' column header in the systems information view. Another red arrow points from the '000365' link in the main report to the same line in the systems information view.

3.1.9 The program editor shows the line with the statement that is causing the abend.
 Notice that this is NOT the COBOL source code. This what is on the "minidump" downloaded by Fault Analyzer.
 There is no sense to make changes here. But you will see what caused the abend.

ZOS.DEV:2800/IDID10.HIST(F00327)-Report DB2REGI.COB

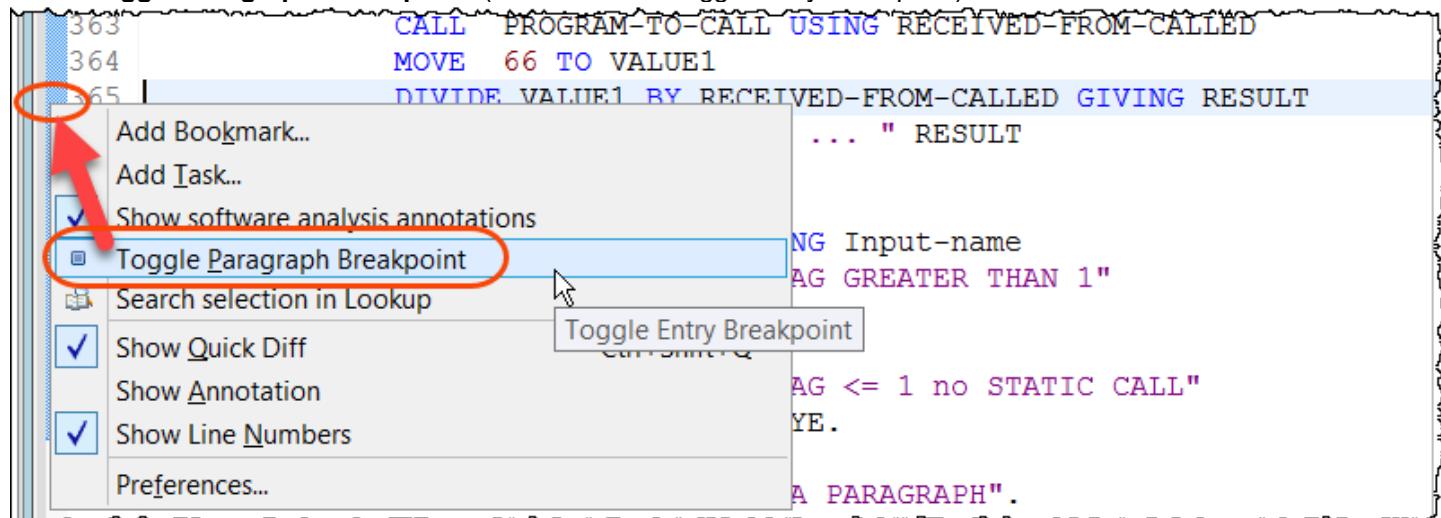
```

-----+--A-1-B---+---2---+---3---+---4---+---5---+---6---+---7
      MOVE "LAB2" to WHICH-LAB.
359   520-LOGIC.
360     IF WHICH-LAB = 'LAB2'
361       * If is LAB2 lets do a dynamic CALL.. and force a divide by ZERO
362       MOVE "REGI0B" TO PROGRAM-TO-CALL
363       CALL PROGRAM-TO-CALL USING RECEIVED-FROM-CALLED
364       MOVE 66 TO VALUE1
365       DIVIDE VALUE1 BY RECEIVED-FROM-CALLED GIVING RESULT
366       DISPLAY "The result is ... " RESULT
367     END-IF
368     IF BRANCHFLAG > 1
369       CALL 'REGI0C' USING Input-name
370       DISPLAY "BRANCHFLAG GREATER THAN 1"
371       PERFORM 530-SEEEYA
372     ELSE
373       DISPLAY "BRANCHFLAG <= 1 no STATIC CALL"
374       PERFORM 540-GOODBYE.
375   530-SEEEYA.
376     DISPLAY "EXECUTED SEEYA PARAGRAPH".

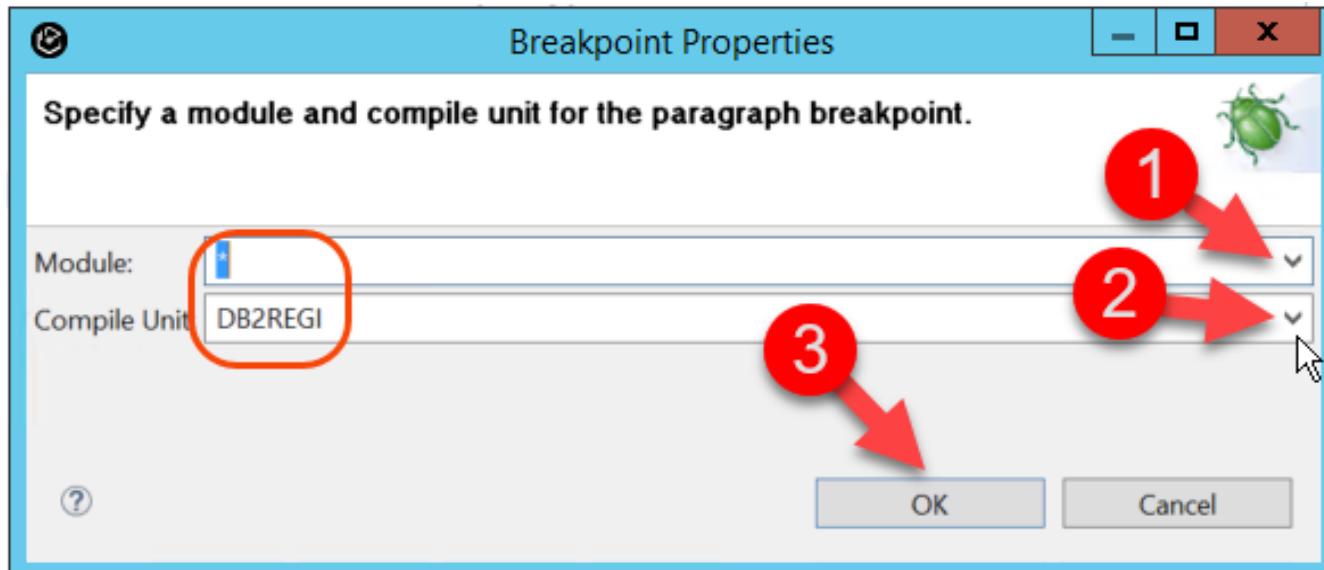
```

Tip: If the font is not clear on the editor, it is because we increased the font to 150%. To make this smaller start the Control Panel > Display > set a custom scaling level > reduce to 125% (we did set to 150%).

- 3.1.10 ► On line 365, right-click in the ruler area (the blue line on left) and select **Toggle Paragraph Breakpoint**. (also known as Toggle Entry Breakpoint)



- 3.1.11 ► On the Breakpoint Properties dialog use drop down to select *, and select or type **DB2REGI** as Compile Unit and click **OK**



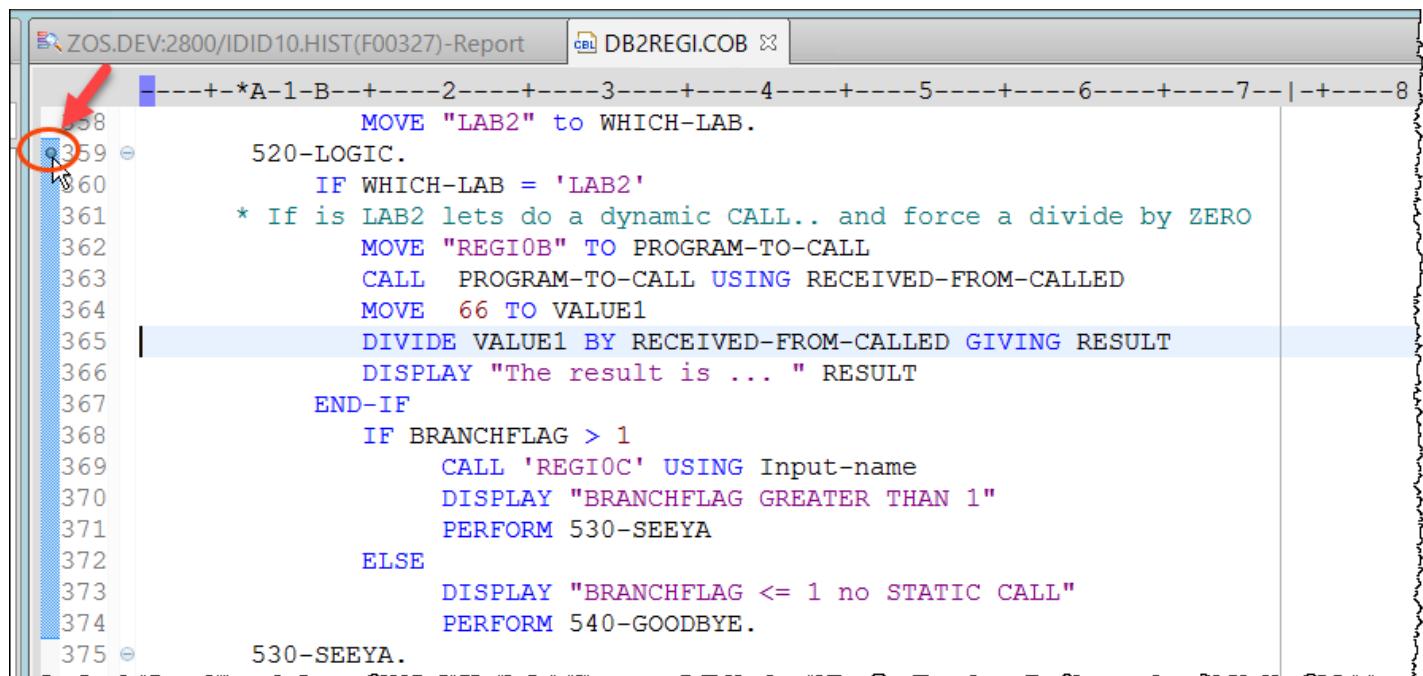
What is *Toggle Paragraph Breakpoint*?

 Toggle Paragraph Breakpoint provides the ability to set paragraph breakpoints prior to starting a debug session. Because these breakpoints are set using the original source files, they persist between debug sessions.

On previous versions, breakpoints could only be set at the level of the generated program listing files.

This allows a more natural edit, compile and debug workflow on the original source files.

- 3.1.12 ► Scroll up a bit. The *Toggle Paragraph Breakpoint* is set at the **520-LOGIC** paragraph. This break point may optionally be used when debugging the COBOL code.



```
--+--*A-1-B---+---2---+---3---+---4---+---5---+---6---+---7---+---8
      MOVE "LAB2" to WHICH-LAB.
359  520-LOGIC.
360      IF WHICH-LAB = 'LAB2'
361          * If is LAB2 lets do a dynamic CALL.. and force a divide by ZERO
362              MOVE "REGI0B" TO PROGRAM-TO-CALL
363                  CALL PROGRAM-TO-CALL USING RECEIVED-FROM-CALLED
364                  MOVE 66 TO VALUE1
365                      DIVIDE VALUE1 BY RECEIVED-FROM-CALLED GIVING RESULT
366                      DISPLAY "The result is ... " RESULT
367                  END-IF
368                  IF BRANCHFLAG > 1
369                      CALL 'REGI0C' USING Input-name
370                      DISPLAY "BRANCHFLAG GREATER THAN 1"
371                      PERFORM 530-SEYYA
372                  ELSE
373                      DISPLAY "BRANCHFLAG <= 1 no STATIC CALL"
374                      PERFORM 540-GOODBYE.
375  530-SEYYA.
```

3.1.13 ► Close all opened editors using **CTRL+ Shift + F4**.

Or just click on the  of each opened editor.

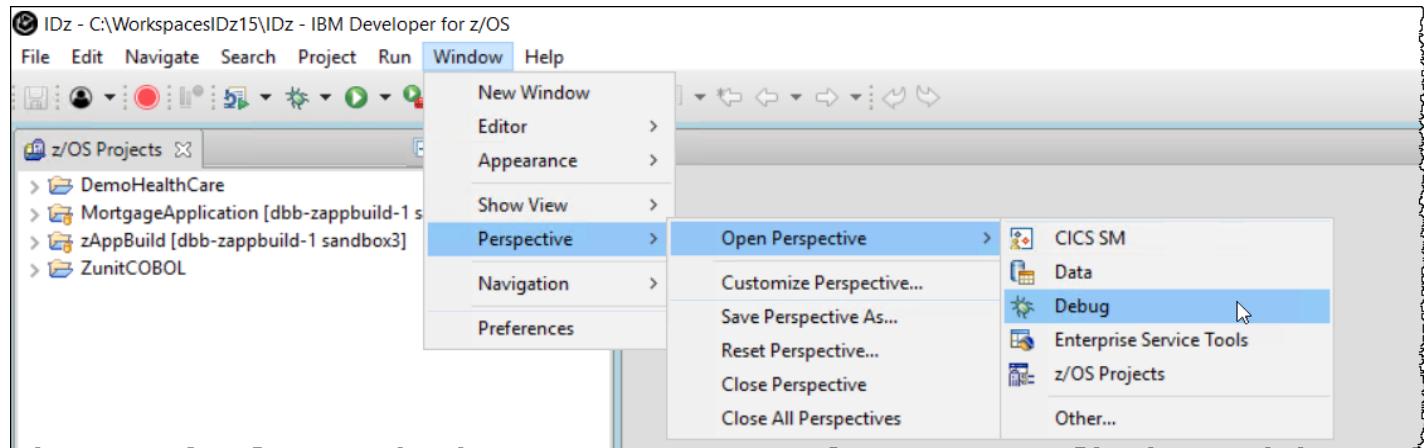
Section 4. Using the IBM z/OS Debugger for a temporary fix

You will use the Debug to verify the ABEND and make a runtime fix.

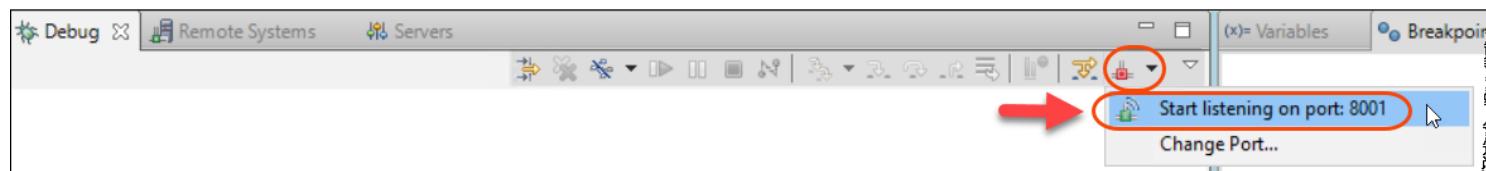
4.0 Be sure that IDz client is listening on port 8001

Usually this step is NOT required, but in our cloud environment we have timeouts that force ports to be closed.

4.0.1 ► Go to the *Debug Perspective* by selecting **Windows > Perspective > Open perspective > Debug**



4.0.2 ► If the icon is red, click and select **Start listening on port 8001**



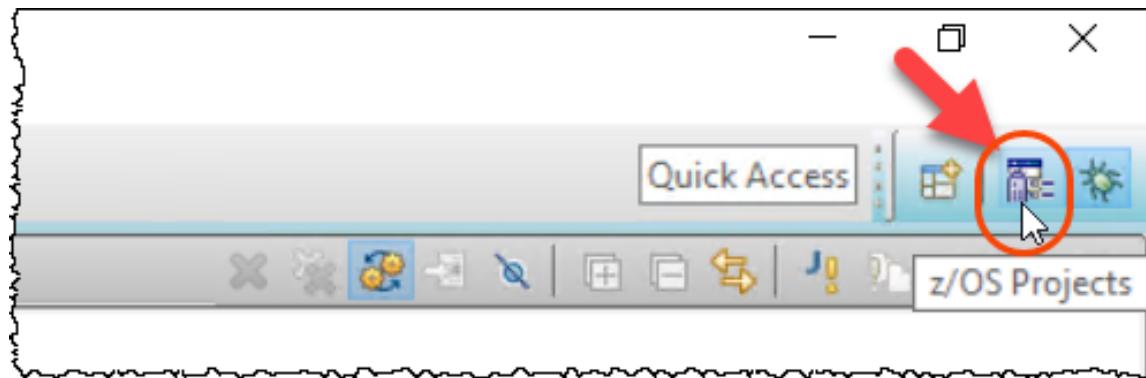
4.0.3 The listening icon will turn green and the IDz client is listening on port 8001.



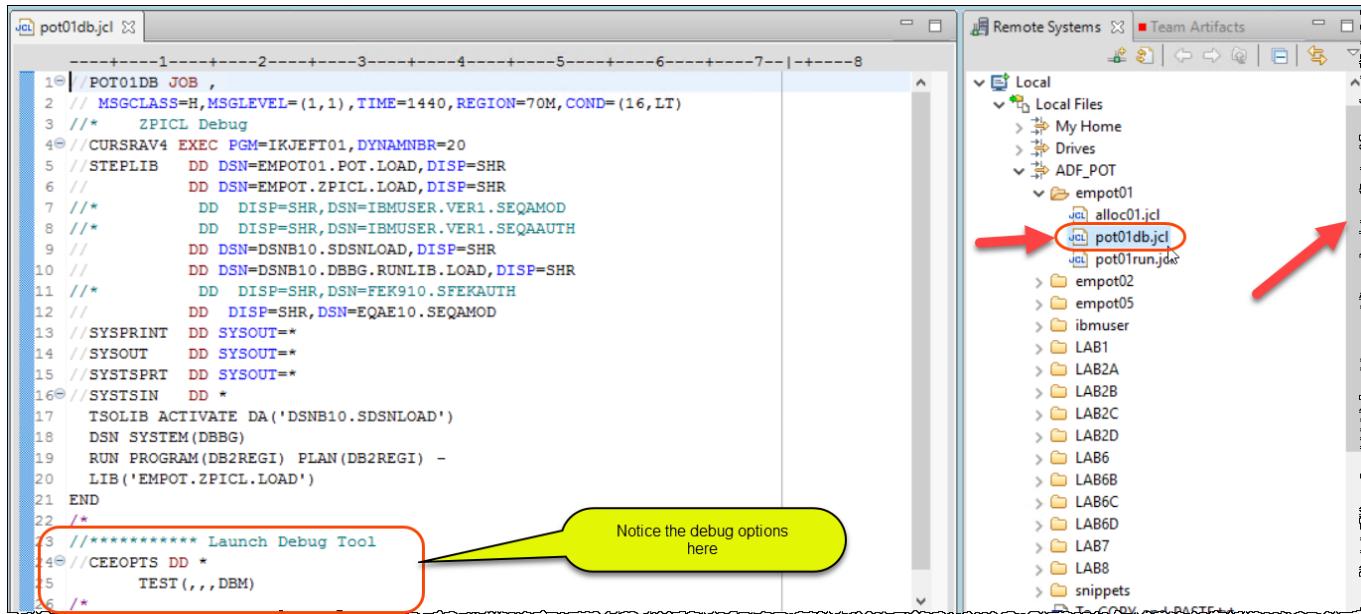
4.1 Submit the JCL to invoke the Debug

You can now submit the JCL to execute again with the Debug option.

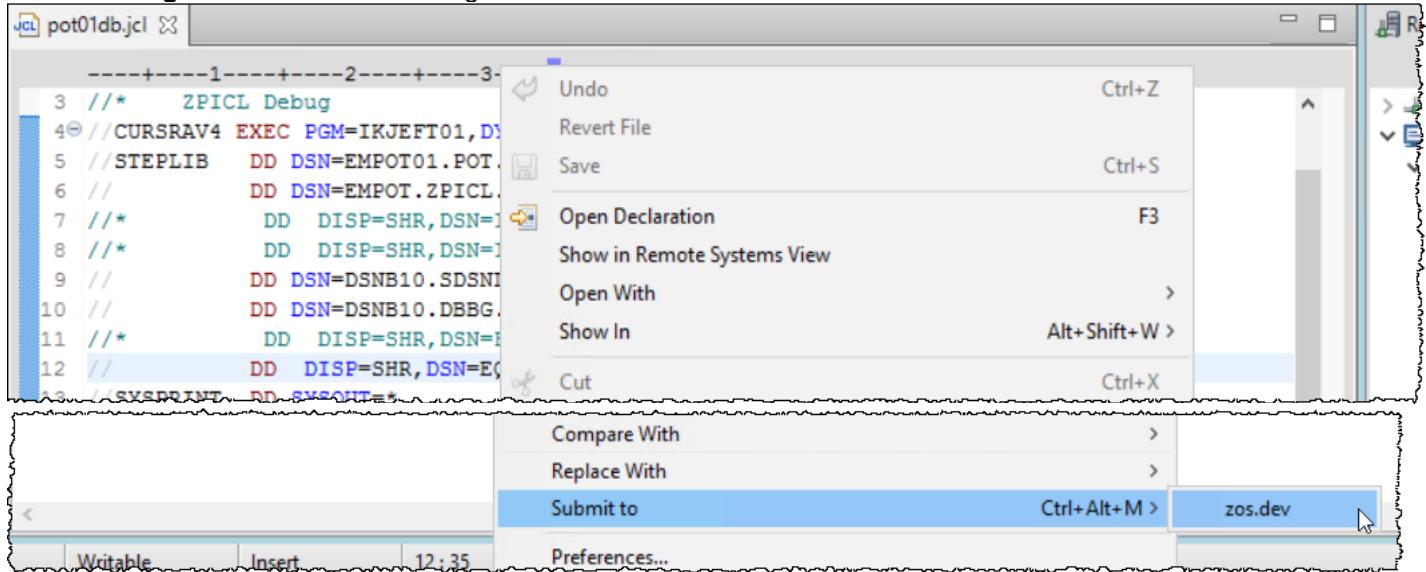
4.1.1 ► Using the top right corner Go to the **z/OS Projects** Perspective by clicking on the icon below.



4.1.2 ► Using *Remote System View*, scroll up to locate the file **pot01db.jcl** under **Local/Local Files/ADF_POT/empot01** and double click to edit..



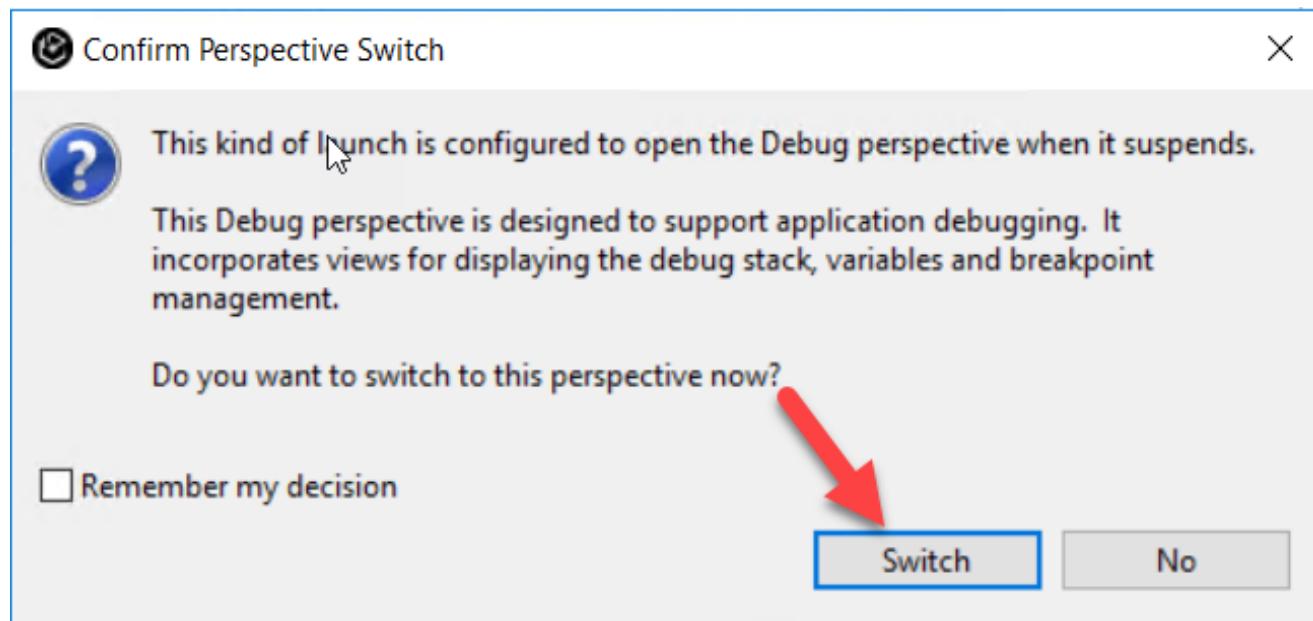
4.1.3 ► Right click on the JCL being edited and select **Submit to > zos.dev**



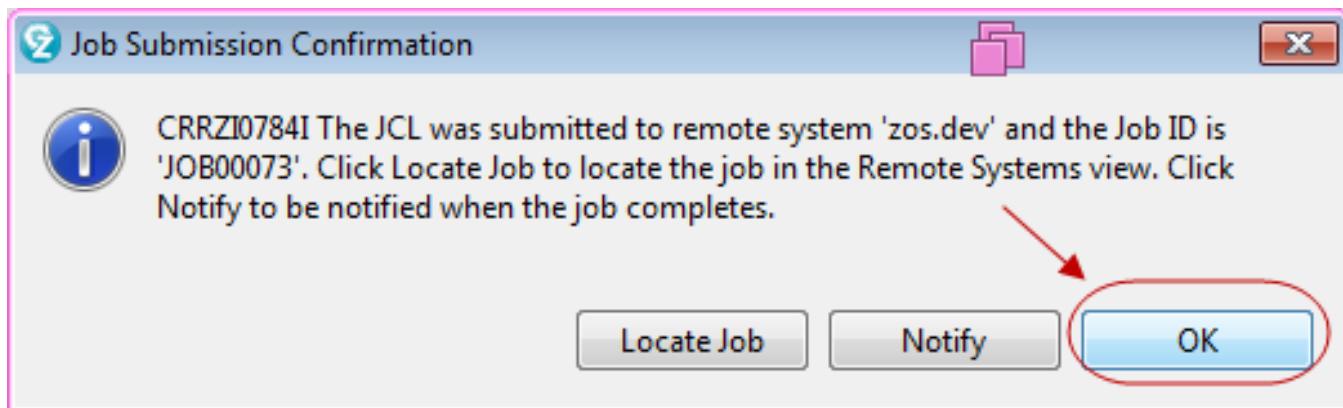
4.1.4 Once the job starts the z/OS debug will "talk" with you. Notice that the communication will be via the IDz connection (RSE), you don't need to specify IP addresses. This may work even when firewalls are in place.

► Click **Switch** to open the *Debug Perspective*

Notice: Depending how fast are you the dialogs order here could be different.

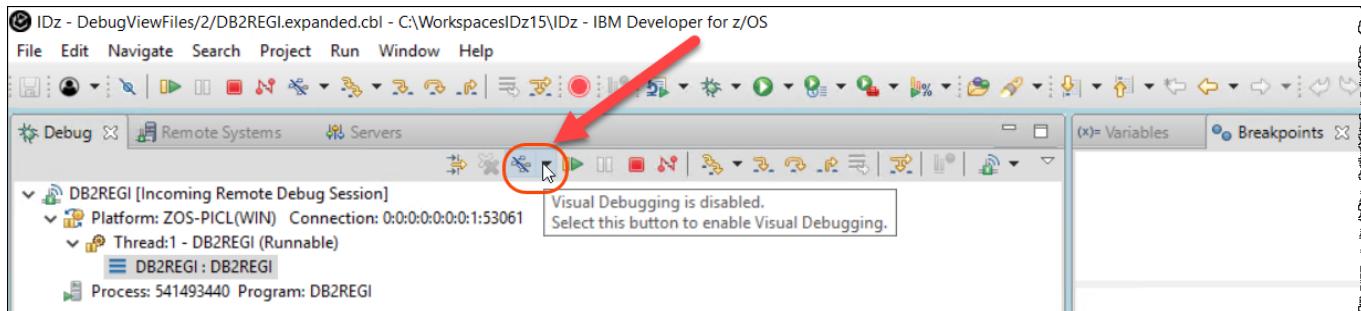


4.1.5 ►| Also click **OK** for this dialog below



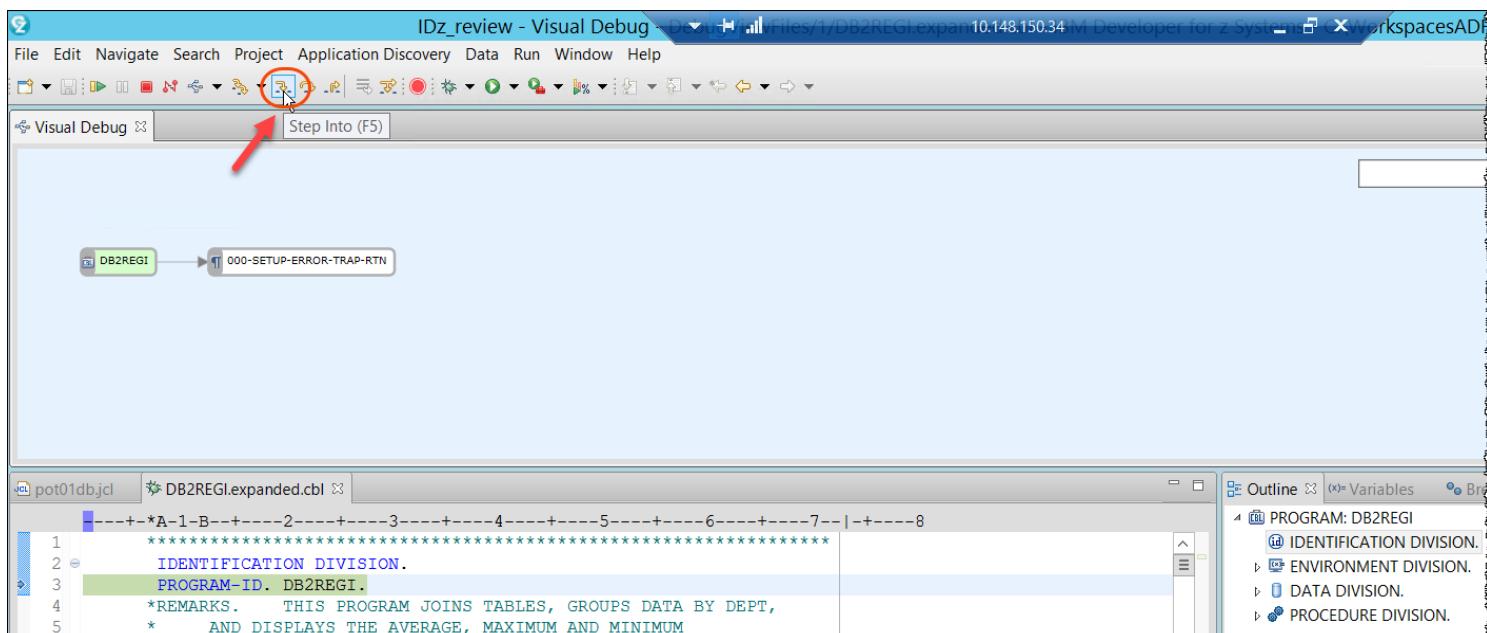
4.1.6 Using the *Debug* perspective:

►| Click the icon to enable Visual Debugging which shows the **Visual Debug** view. If a dialog pops up click **YES**.



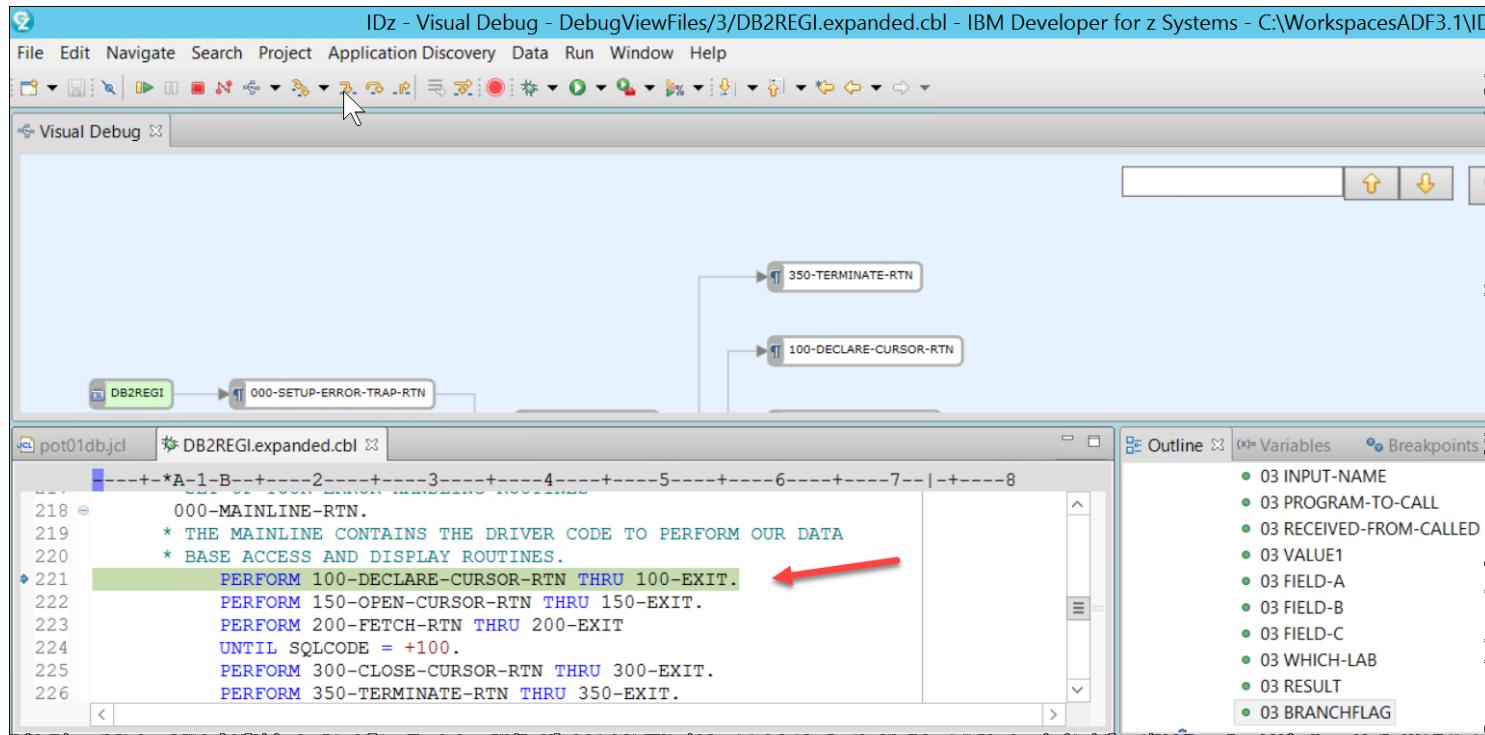
4.1.7 The *Visual Debug* view shows the paragraphs being executed (in the top)

►| Click the icon or **F5** (*Step into*) to execute first line of code.



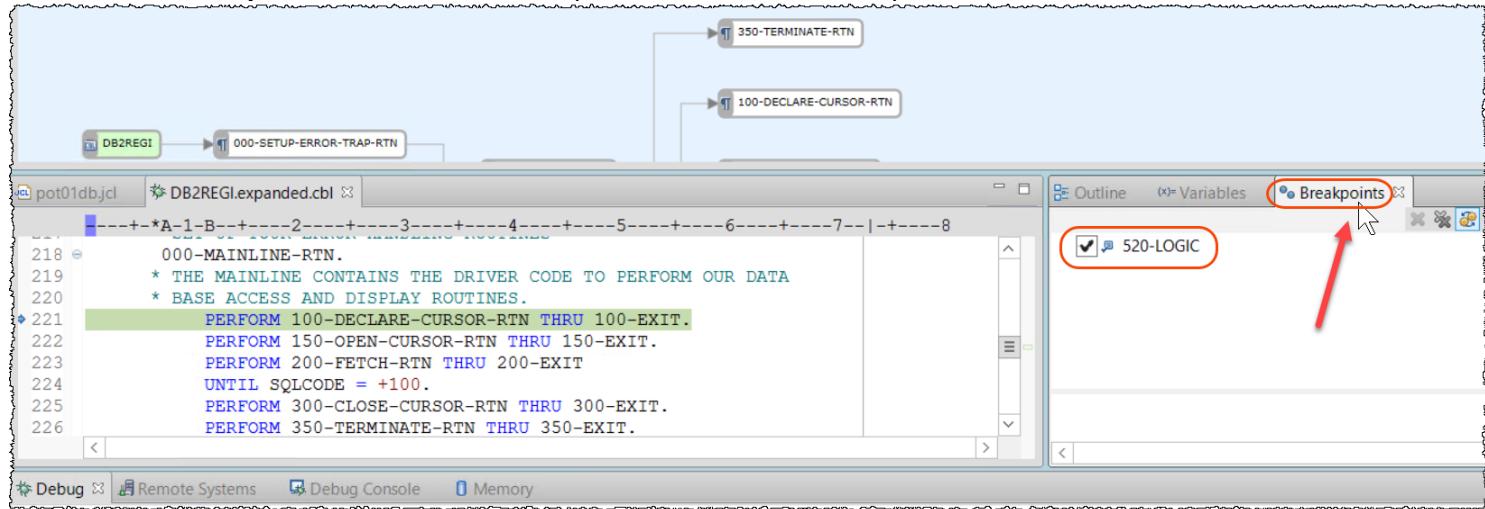
4.1.8 The execution will stop at the statement that will execute

```
PERFORM 100-DECLARE-CURSOR-RTN THRU 100-EXIT
```



4.1.9 On the step 3.1.10 when using the *Fault Analyzer*, you created a *paragraph breakpoint* even without having the execution started. This is handy since it will show the area that needs to be debugged..

► Click on **Breakpoints** tab to see this breakpoint created before on step 3.1.10.



What is the Visual Debugging?

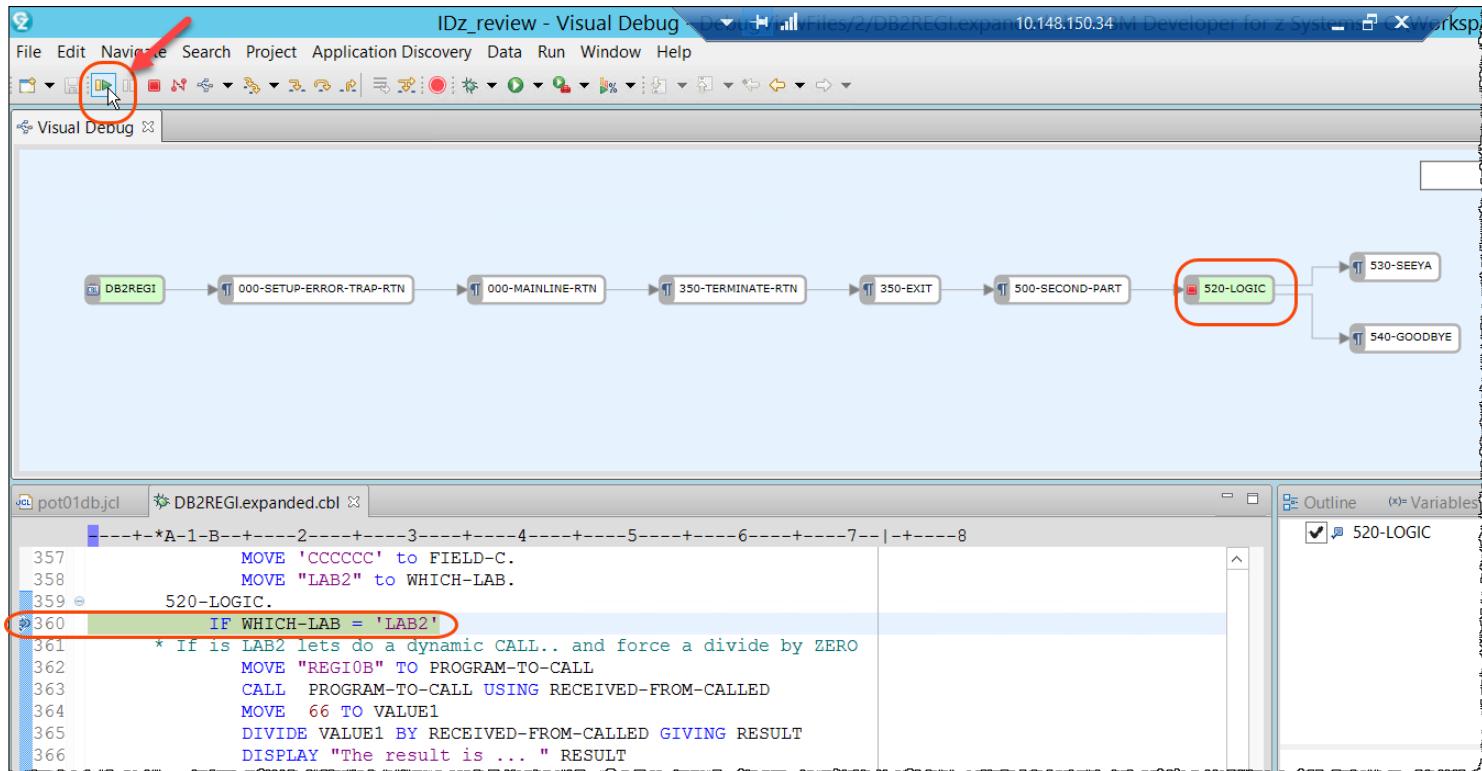


Visual debugging allows you to interact with your COBOL or PL/I debug session using the program control flow diagram.

With this diagram, you can visualize the stack trace, set breakpoints, and run to a selected

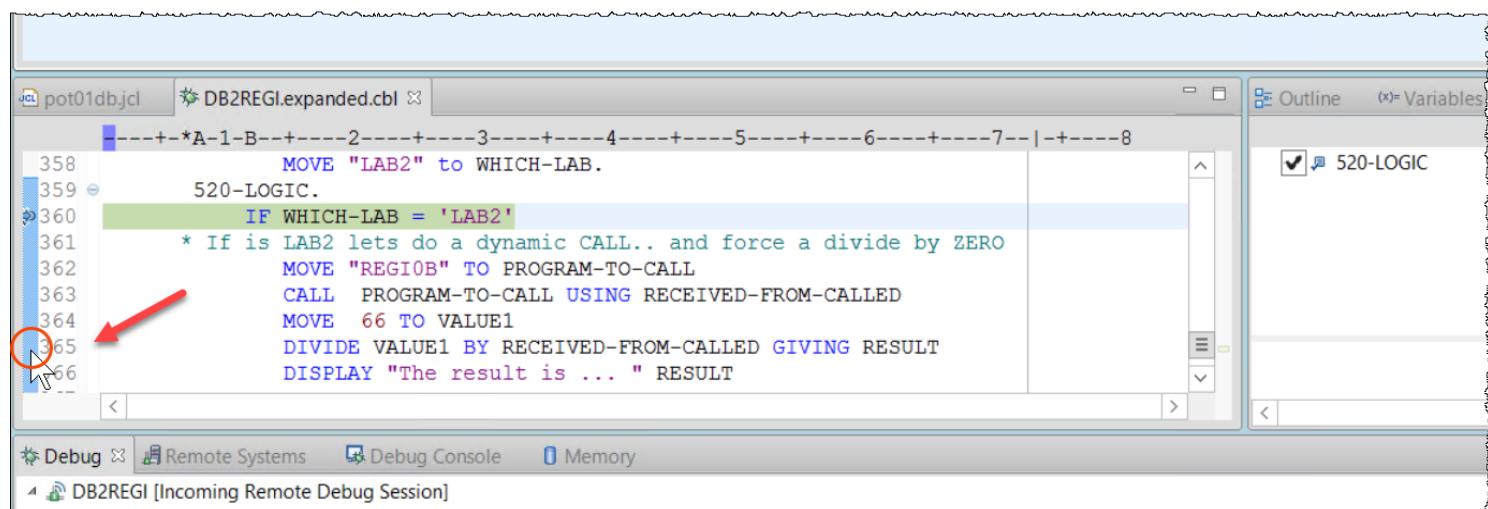
call path. In COBOL, the stack trace represents the paragraph call chain.
In PL/I, the stack trace represents the procedure call chain.

- 4.1.10 ► Click on or press **F8** and notice that the execution will stop on line 360 since a *Paragraph Entry Breakpoint* was created before. This line is about to be executed.

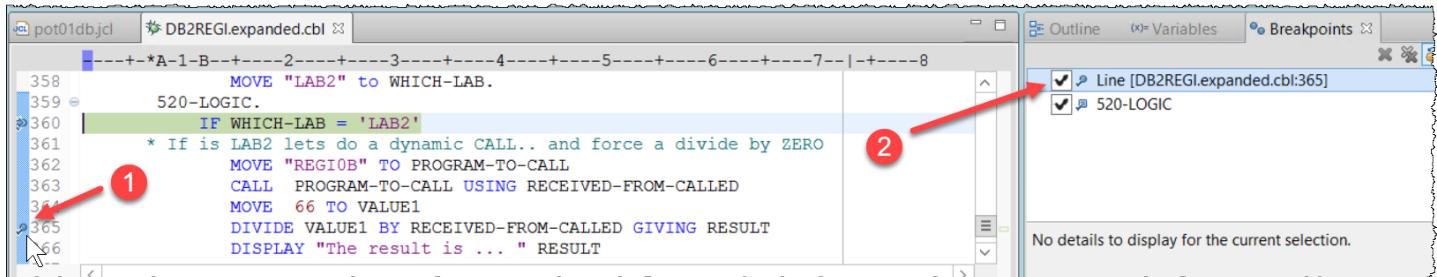


- 4.1.11. As you verified using Fault Analyzer, the abend occurred on line 365 where you had a divide by zero (see step 3.1.10). Now you will add a breakpoint on this line and change the values to avoid the abend.

- On the COBOL editor move the mouse to the **blue column** on left and **double click** on the line 365 `DIVIDE VALUE1 BY RECEIVED-FROM-CALLED GIVING RESULT` to create a breakpoint.

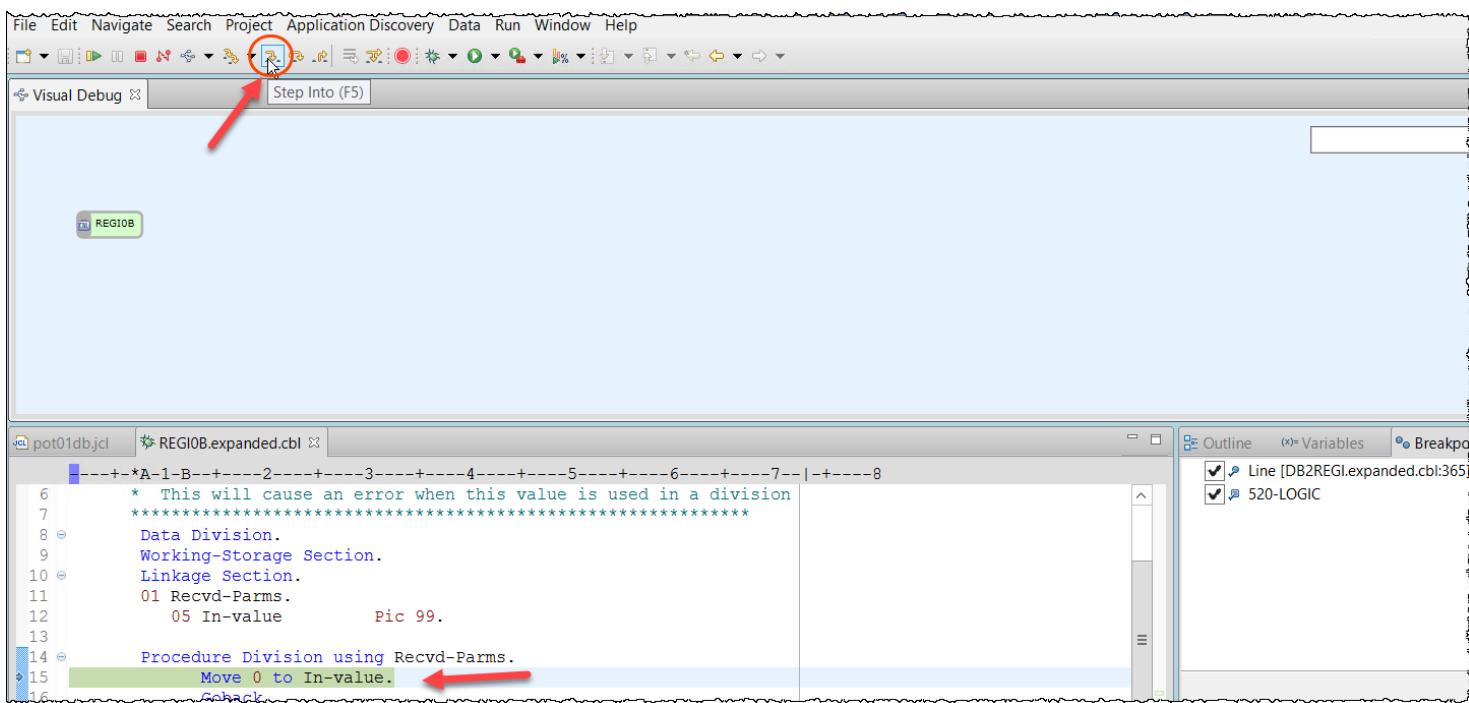


4.1.12 Notice that a small circle  is shown on the left of line 365 and also a breakpoint is displayed on the Breakpoint view



The screenshot shows the IBM Rational Application Developer interface. On the left, the COBOL source code for 'DB2REGI.expanded.cbl' is displayed. Line 365 contains a breakpoint, indicated by a small blue circle with a white dot on the left margin. A red arrow labeled '1' points to this circle. On the right, the 'Breakpoints' view is open, showing two breakpoints: 'Line [DB2REGI.expanded.cbl:365]' and '520-LOGIC'. Both have checkmarks and are highlighted with a red circle labeled '2'. The status bar at the bottom right of the Breakpoints view says 'No details to display for the current selection.'

4.1.13  Continue by clicking on  or using F5 until you will see that a program named REGI0B is called and this program is returning a value of 0.



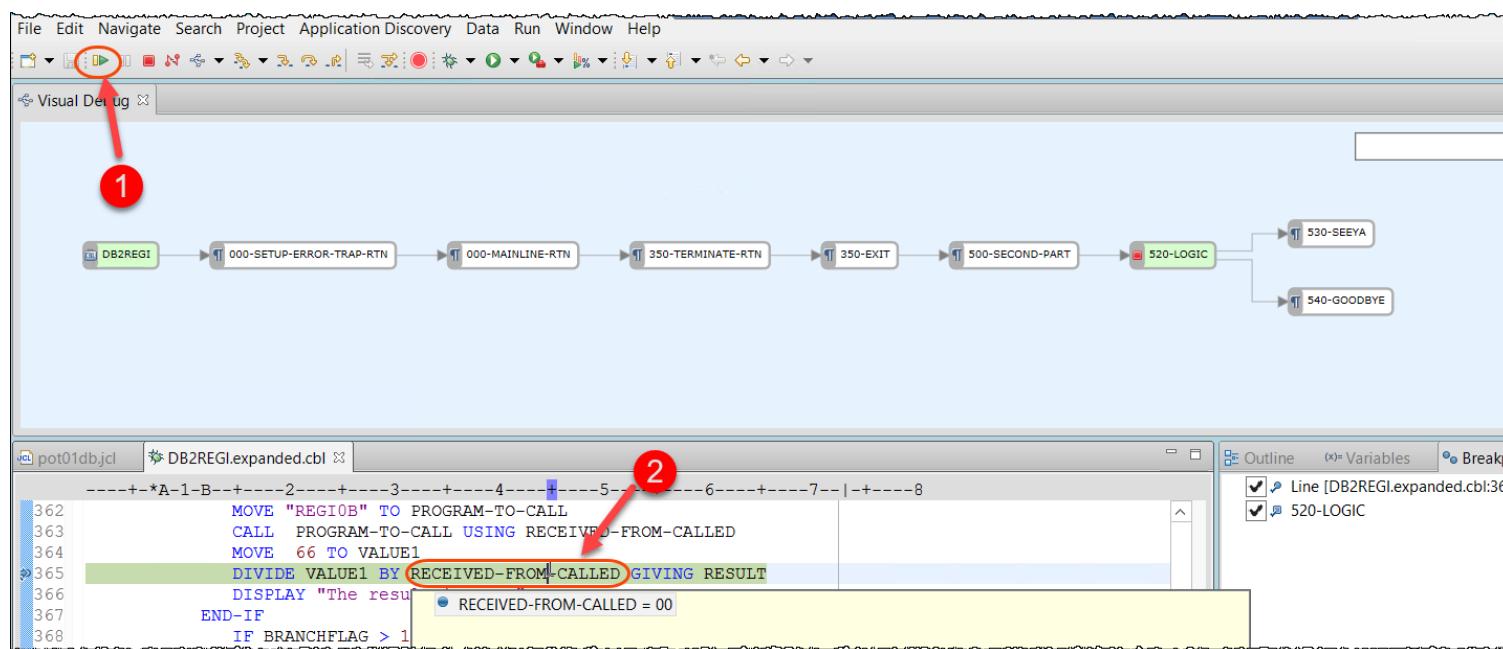
The screenshot shows the IBM Rational Application Developer interface. At the top, the toolbar has a 'Step Into (F5)' button highlighted with a red circle and a red arrow pointing to it. Below the toolbar, the 'Visual Debug' window is open, showing a stack frame for 'REGI0B'. In the COBOL editor window below, the source code for 'REGI0B.expanded.cbl' is shown. Line 15 contains the instruction 'Move 0 to In-value.', which is highlighted with a red arrow. The status bar at the bottom right of the editor window says 'No details to display for the current selection.'

4.1.14 1 Click on or press F8.

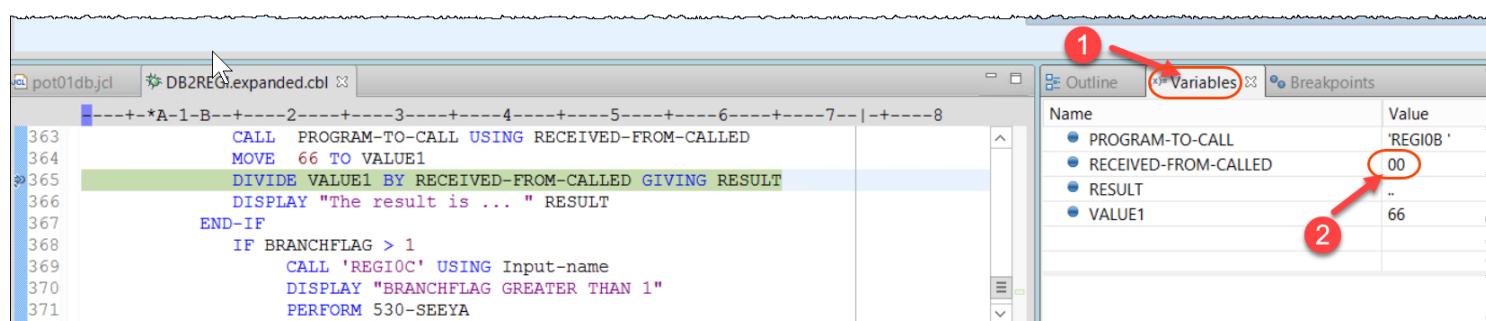
The execution will stop at the breakpoint that you created (line 365). .

2 Move the cursor to RECEIVED-FROM-CALLED variable and click to see the 00 value..

(Tip: under the cloud instance the behavior may act different and you may need to click on the editor area before moving the mouse to that field).



4.1.15 1 Click the Variables tab (right) and 2 verify that RECEIVED-FROM-CALLED is 00.



4.1.16 This is the abend cause. You must change the value to something different than 0.

Click on RECEIVED-FROM-CALLED value of 00 and modify to 2, just overtyping and press enter.

The screenshot shows a COBOL program named DB2REGI.expanded.cbl being run in a debugger. The code is as follows:

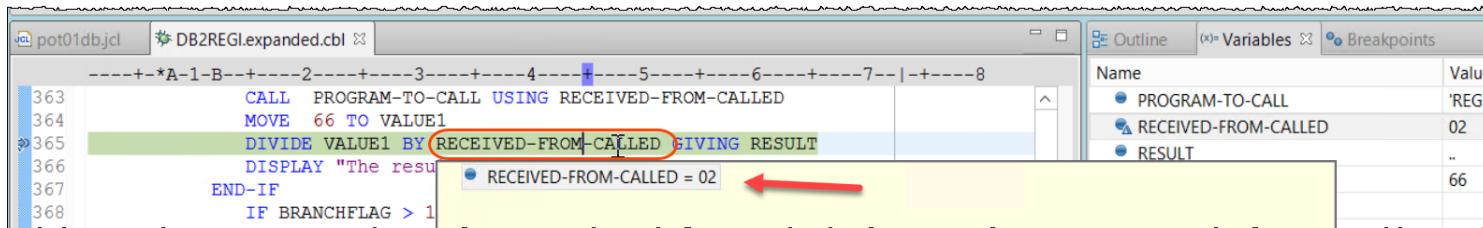
```
363      CALL PROGRAM-TO-CALL USING RECEIVED-FROM-CALLED
364      MOVE 66 TO VALUE1
365      DIVIDE VALUE1 BY RECEIVED-FROM-CALLED GIVING RESULT
366      DISPLAY "The result is ... " RESULT
367      END-IF
368      IF BRANCHFLAG > 1
369          CALL 'REGI0C' USING Input-name
370          DISPLAY "BRANCHFLAG GREATER THAN 1"
371          PERFORM 530-SEEEYA
```

The Variables view on the right shows the current values of the variables:

Name	Value
PROGRAM-TO-CALL	'REGI0B'
RECEIVED-FROM-CALLED	02
RESULT	..
VALUE1	66

A red arrow points to the value '02' in the RECEIVED-FROM-CALLED row, which is highlighted in green.

- 4.1.17 ► Again, move the cursor to **RECEIVED-FROM-CALLED** variable and now see the value **02**.
 (Tip: under the cloud instance the behavior may act different and you may need to click on the editor area before moving the mouse to that field).



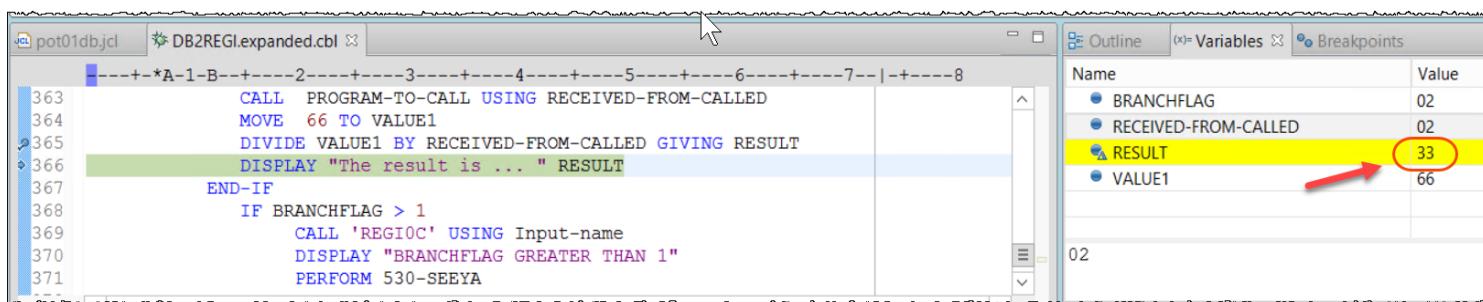
```

pot01dbjcl DB2REGI.expanded.cbl
-----+-----+-----+-----+-----+-----+-----+-----+
363      CALL PROGRAM-TO-CALL USING RECEIVED-FROM-CALLED
364      MOVE 66 TO VALUE1
365      DIVIDE VALUE1 BY RECEIVED-FROM-CALLED GIVING RESULT
366      DISPLAY "The result is ... " RESULT
367      END-IF
368      IF BRANCHFLAG > 1
-----+-----+-----+-----+-----+-----+-----+-----+

```

Name	Value
PROGRAM-TO-CALL	REGI00B
RECEIVED-FROM-CALLED	02
RESULT	..
VALUE1	66

- 4.1.18 ► Click the icon (Step into) or press **F5** to see the next line being executed
 This time the divide by 2 give you a result of **33**.



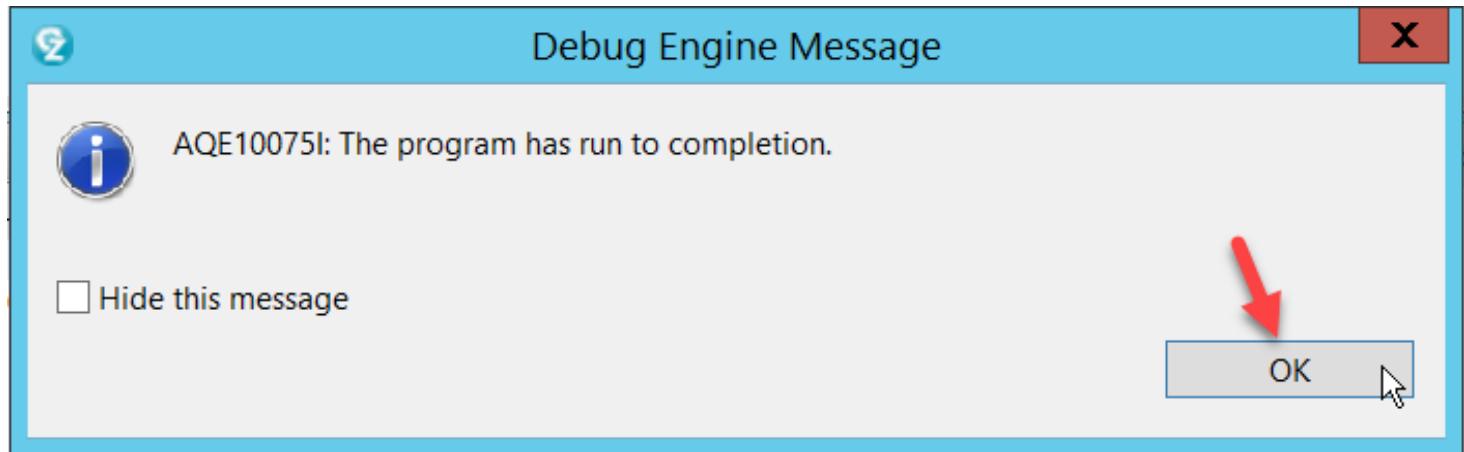
```

pot01dbjcl DB2REGI.expanded.cbl
-----+-----+-----+-----+-----+-----+-----+-----+
363      CALL PROGRAM-TO-CALL USING RECEIVED-FROM-CALLED
364      MOVE 66 TO VALUE1
365      DIVIDE VALUE1 BY RECEIVED-FROM-CALLED GIVING RESULT
366      DISPLAY "The result is ... " RESULT
367      END-IF
368      IF BRANCHFLAG > 1
369          CALL 'REGI0C' USING Input-name
370          DISPLAY "BRANCHFLAG GREATER THAN 1"
371          PERFORM 530-SEEYA
-----+-----+-----+-----+-----+-----+-----+-----+

```

Name	Value
BRANCHFLAG	02
RECEIVED-FROM-CALLED	02
RESULT	33
VALUE1	66

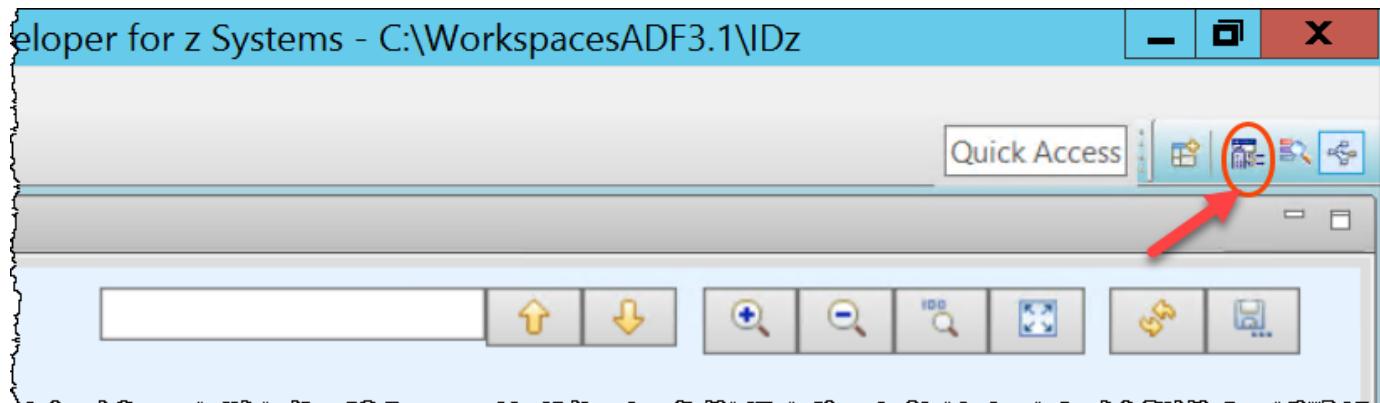
- 4.1.19 ► Click on (or F5) few times
 ► and **Resume** (F8) when you are satisfied so the program will execute until the end.



- 4.1.20 ► Click **OK** to close the dialog above.

To fix this bug definitely you will modify the program **REGI00B** that is returning 0 to be used in a division.
 You will do that later. For now, optionally you can play again with the Debugger and use the Visual Debugger.
 Or continue after the optional steps.

4.1.21 ► Go back to the **z/OS Projects Perspective** by clicking on the icon  in the top right corner.



4.2 (OPTIONAL) Using the Visual Debugger for Stack pattern breakpoints

If you are not running late and interested on this subject, execute the steps below, otherwise jump to 4.3

Stack pattern breakpoint



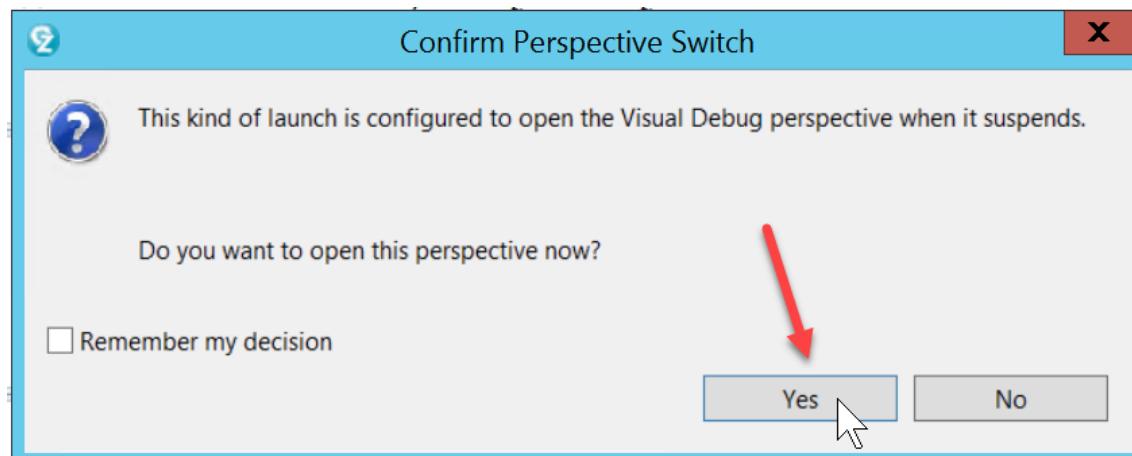
A stack pattern breakpoint is a special type of conditional breakpoint. With this feature, you can specify that you only want to stop at a location when the current stack trace contains a predefined stack pattern.

Visual debugging supports a stack pattern integration feature, which allows you to select a connected path from the program control flow diagram and set a stack pattern breakpoint using the selected path as a stack pattern.

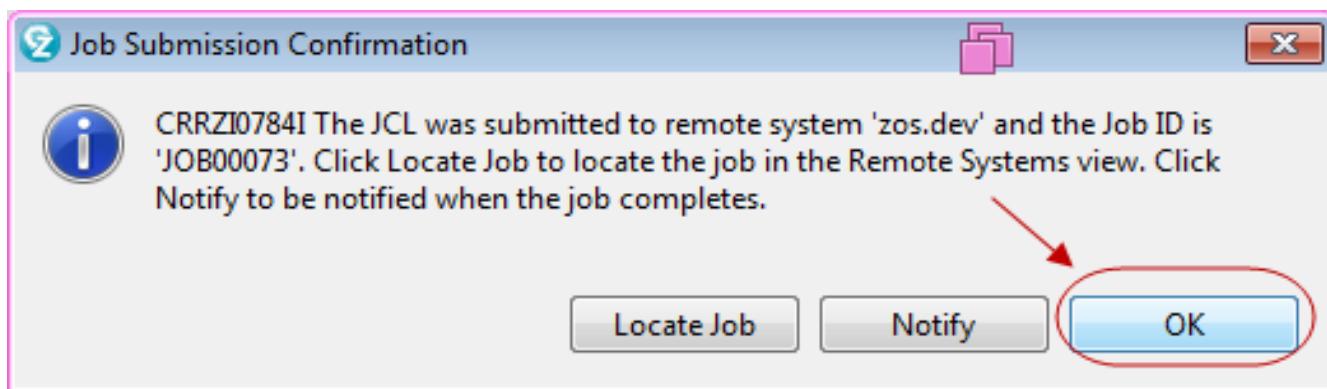
4.2.1 ► (Optional) Right click on the JCL being edited and select **Submit > zos.dev**

4.2.2 (Optional) Once the job starts the z/OS debug will “talk” with you.

▶ Click **Yes** to open the *Visual Debug Perspective*
(depending on the z/OS the order here could be the next step first).

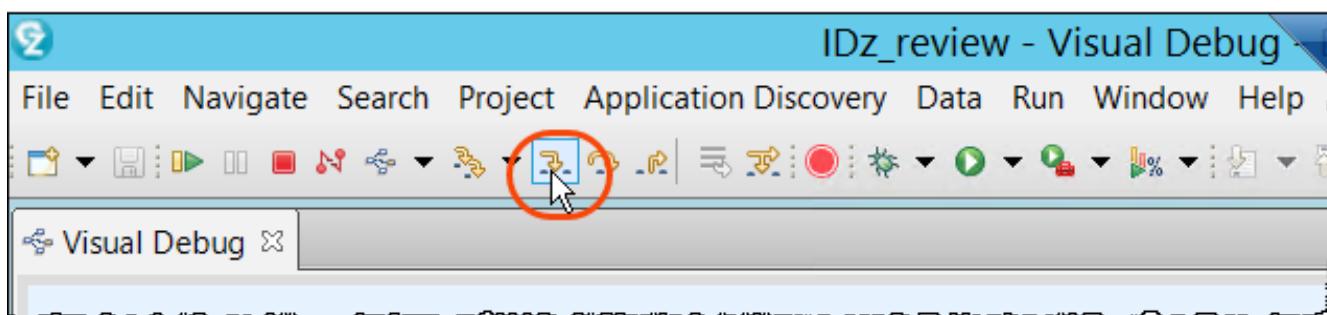


4.2.3 ► (Optional) Also click **OK** for this dialog below

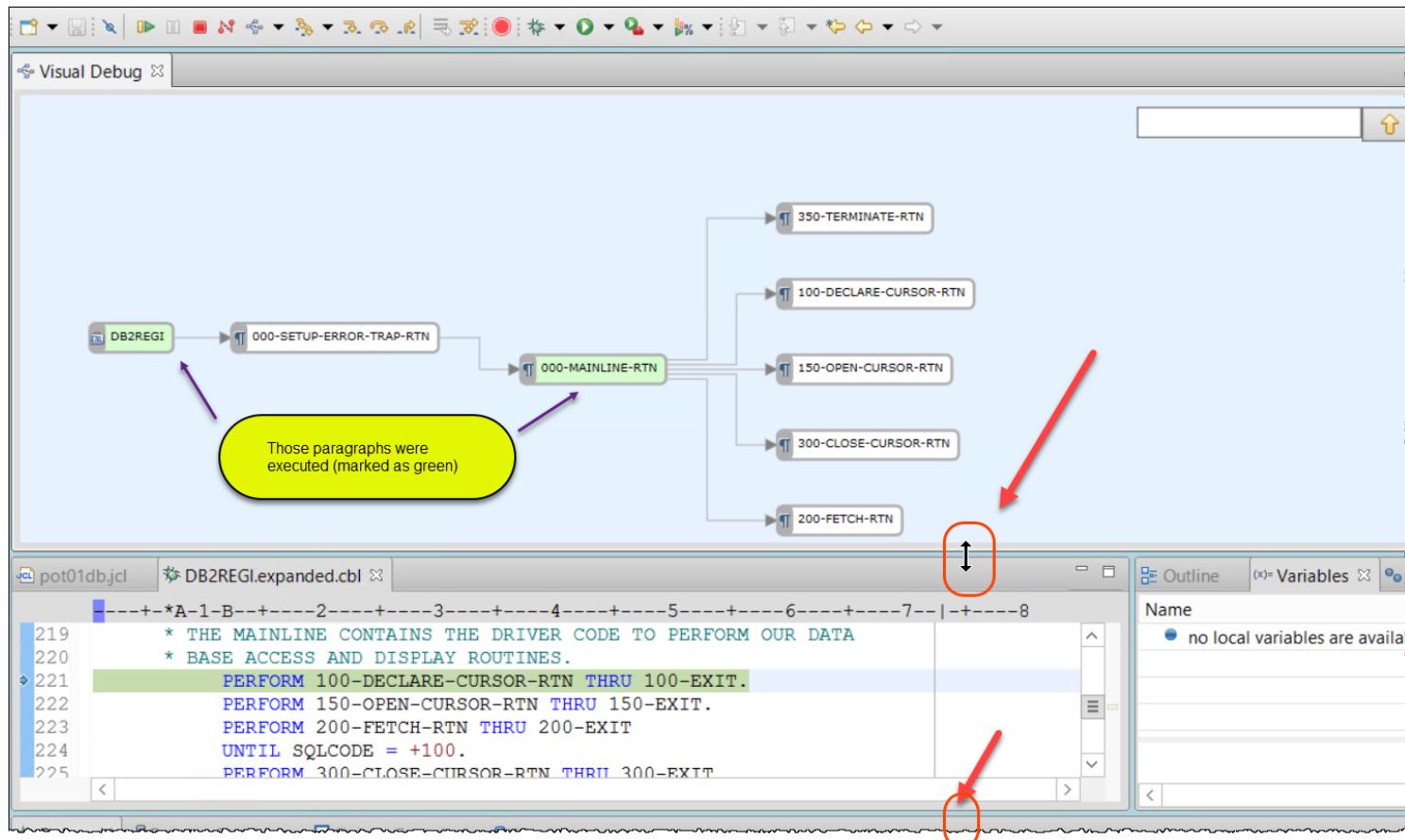


4.2.4 (Optional) Using the *Visual Debug* perspective:

► Click on icon (or Press **F5**)



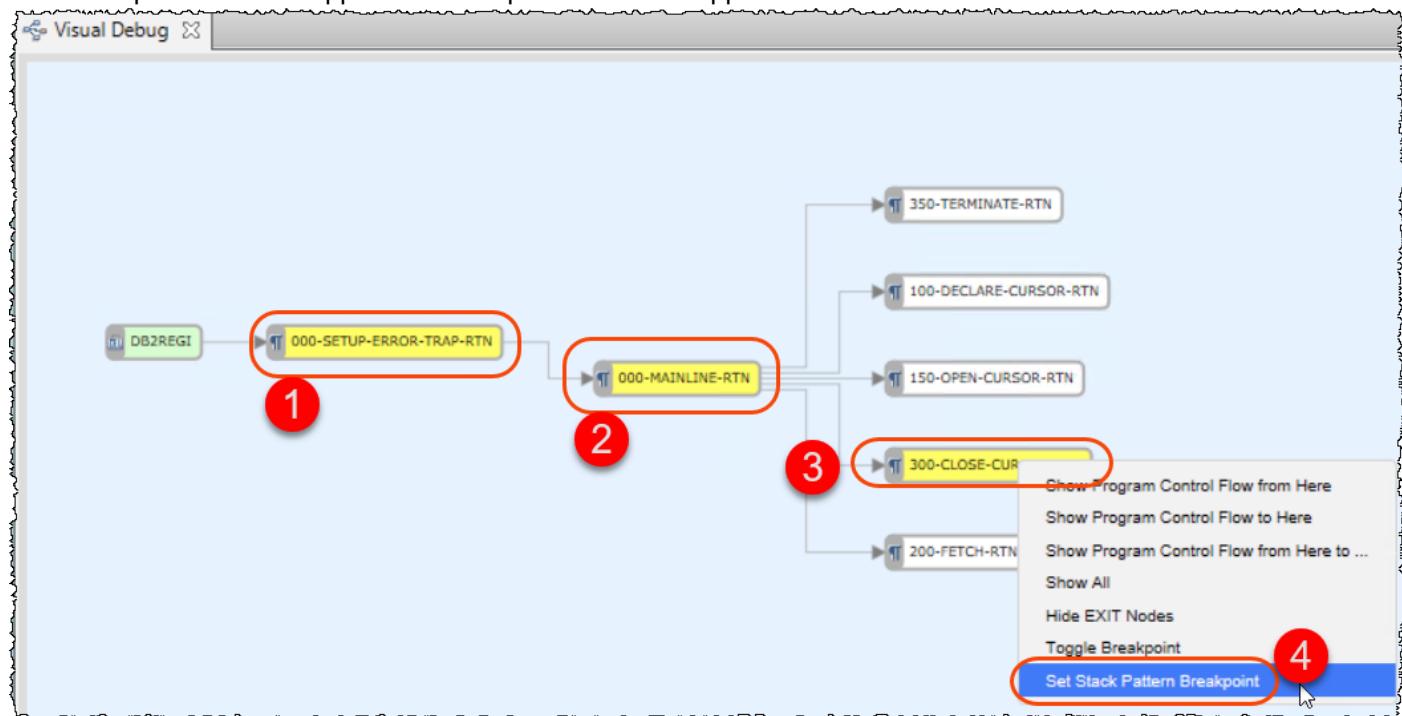
4.2.5 ► (Optional) Resize the **Visual Debug** view and you will notice that the paragraphs executed are marked as green



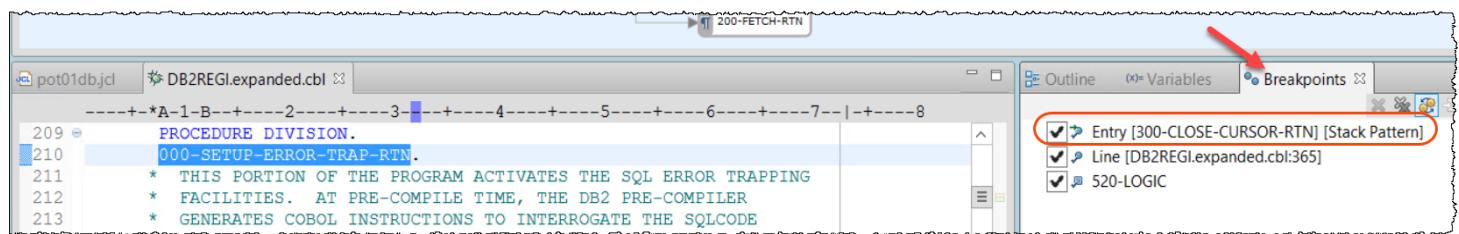
4.2.6 ► (Optional) Using the *Visual Debug* view, scroll down

► Press **CTRL** Key and select the 3 paragraphs as shown below, right click and choose **Set Stack Pattern Breakpoint**

This breakpoint that will happen when the path selected happens.

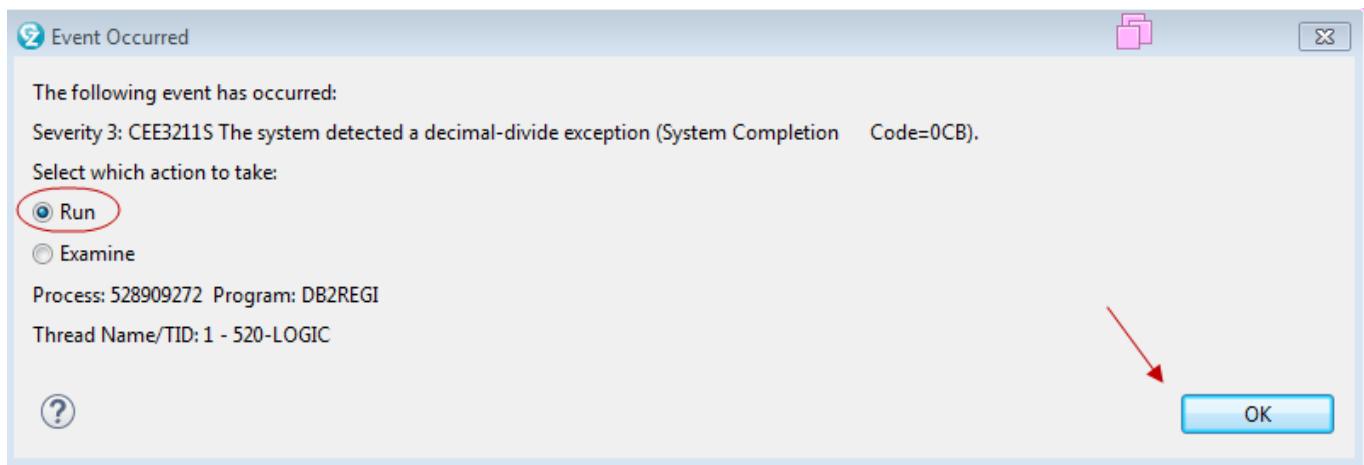


4.2.7 ► (Optional) Click on **Breakpoints** view and you will see this breakpoint created.



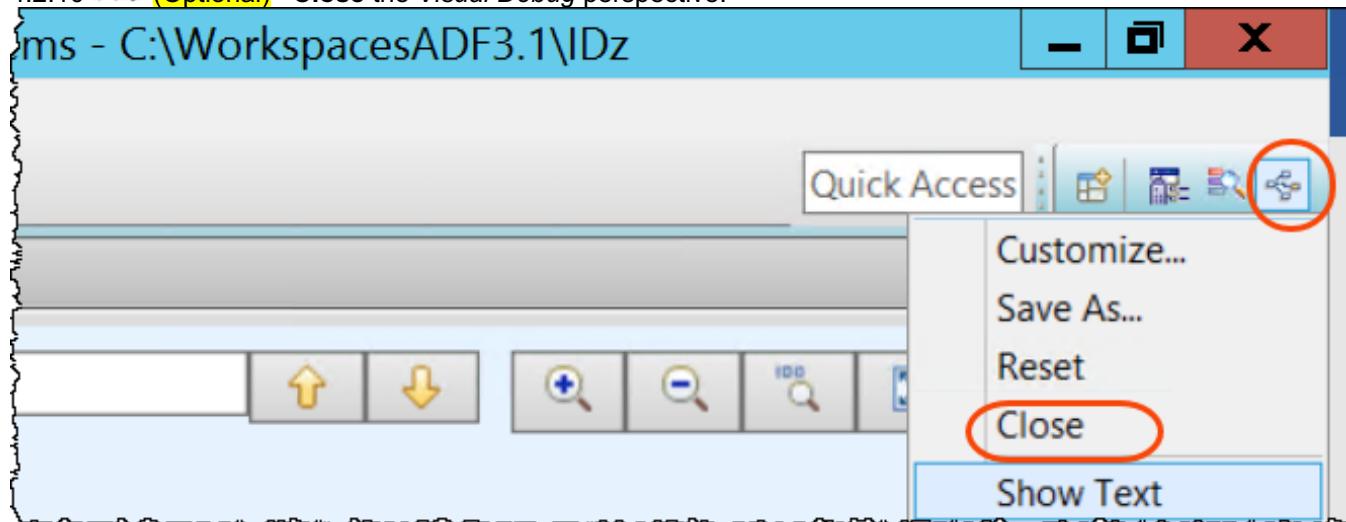
4.2.8 ► (Optional) Click on (or press **F8**) few times to resume the execution

4.2.9 ➤ (Optional) Select **Run** and click **OK** to continue

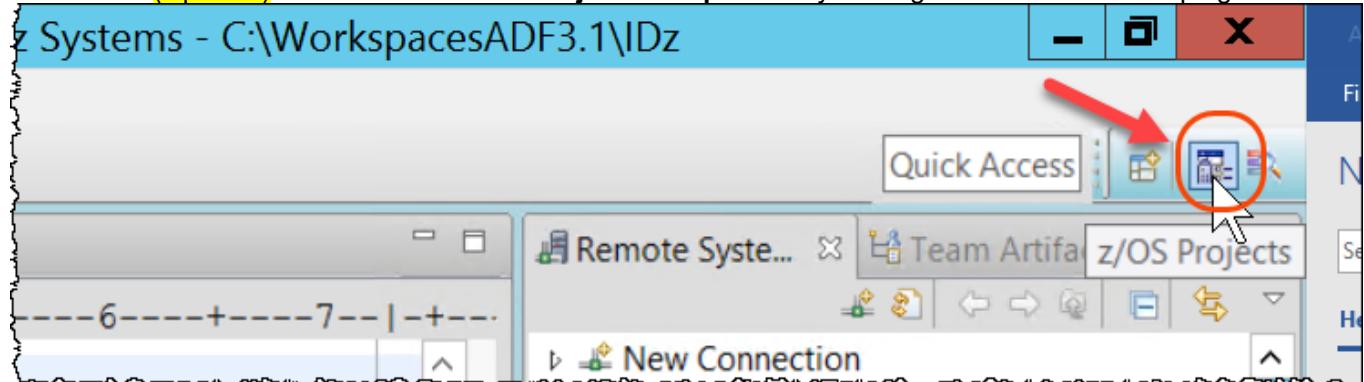


Notice – This breakpoint did not happen since the execution did not execute the 3 selected paragraphs.

4.2.10 ► (Optional) Close the *Visual Debug* perspective.



4.2.11 ► (Optional) Go back to the **z/OS Projects Perspective** by clicking on the icon the top right corner.



4.3 Accessing the z/OS JES

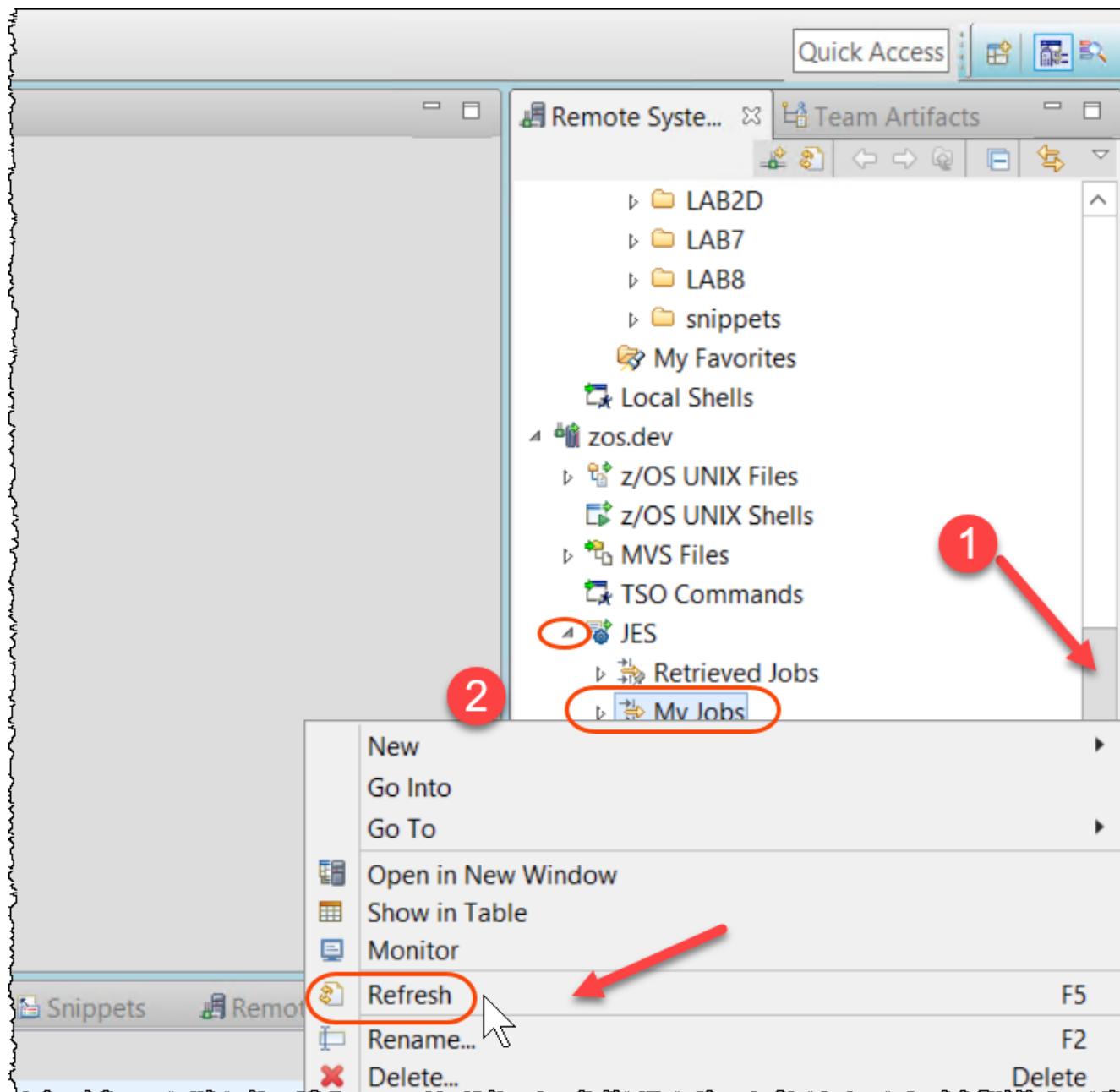
On ISPF many people use *SDSF* for this activity.

You will use the Job Monitor subsystem part of the host components.

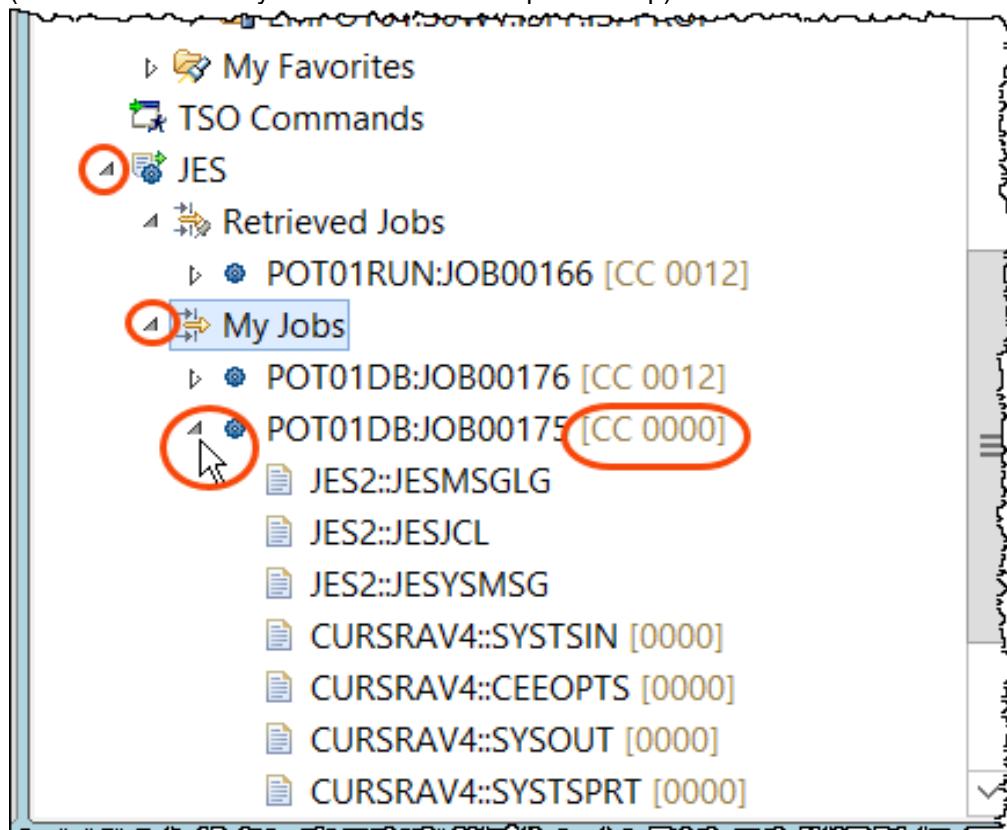
4.3.1 ► Use **Ctrl + Shift + F4** to close all opened editors. Do not save any changes.

Or just click on the of each opened editor.

4.3.2 ► Using the *Remote Systems* view scroll down, expand JES right click on **My Jobs** and select **Refresh**



- 4.3.3 ► Expand **My Jobs** and the Job **POT01DB:JOB00xxx** that has the CC 0000
(it will be the latest if you did not make the optional step)



- 4.3.4 ► On Remote Systems view, double click on the **POT01DB:JOB00xxx** that has the CC 0000

POT01DB:JOB00175.spool

```

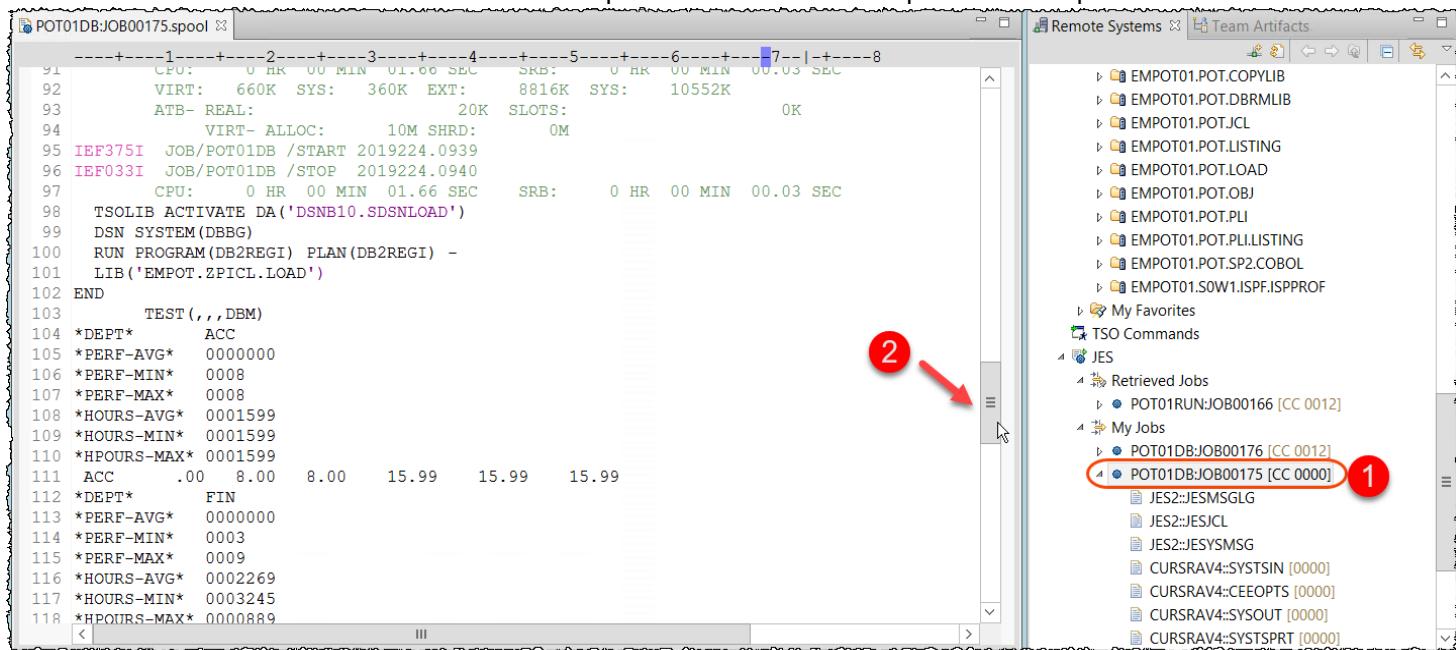
1 J E S 2   B   O   L   O   G   --   S   Y   S   T   E   M   S   O   W   1   --   N   O   D   E   S   O   W
2
3 09.39.41 JOB00175 ---- MONDAY, 12 AUG 2019 ----
4 09.39.41 JOB00175 IRR0101 USERID EMPOT01 IS ASSIGNED TO THIS JOB.
5 09.39.42 JOB00175 ICH70001I EMPOT01 LAST ACCESS AT 09:37:51 ON MONDAY, AUGUST 12, 2019
6 09.39.42 JOB00175 $HASP373 POT01DB STARTED - INIT 1 - CLASS A - SYS S0W1
7 09.39.43 JOB00175 +EQAP0018I Using DBM port 5336; timeout=300 seconds
8 09.40.24 JOB00175 --TIMINGS (MINS.)--
9 09.40.24 JOB00175 -JOBNAME STEPNAME PROCSTEP RC EXCP TCB SRB CLOCK SERV E
10 09.40.24 JOB00175 -POT01DB 00 1240 ***** .00 .7 2251
11 09.40.24 JOB00175 -POT01DB ENDED. NAME- TOTAL TCB CPU TIME=.02
12 09.40.24 JOB00175 $HASP395 POT01DB ENDED - RC=0000
13 ----- JES2 JOB STATISTICS -----
14 12 AUG 2019 JOB EXECUTION DATE
15      27 CARDS READ
16      167 SYSOUT PRINT RECORDS
17      0 SYSOUT PUNCH RECORDS
18      12 SYSOUT SPOOL KBYTES
19      0.70 MINUTES EXECUTION TIME
20 1 //POT01DB JOB ,
21 // MSGCLASS=H,MSGLEVEL=(1,1),TIME=1440,REGION=70M,COND=(16,LT)
22 ///* ZPICL Debug
23 2 //CURSRAV4 EXEC PGM=IKJEFT01,DYNAMNBR=20
24 3 //STEPLIB DD DSN=EMPOT01.POT.LOAD,DISP=SHR
25 4 // DD DSN=EMPOT.ZPICL.LOAD,DISP=SHR
26 ///* DD DISP=SHR,DSN=IBMUSER.VER1.SEQAMOD
27 ///* DD DISP=SHR,DSN=IBMUSER.VER1.SEQAUTH

```

Remote Systems

- EMPOT01.POT.COPYLIB
- EMPOT01.POT.DBRMLIB
- EMPOT01.POT.JCL
- EMPOT01.POT.LISTING
- EMPOT01.POT.LOAD
- EMPOT01.POT.OBJ
- EMPOT01.POT.PLI
- EMPOT01.POT.PLLISTING
- EMPOT01.POT.SP2.COBOL
- EMPOT01.S0W1.ISPF.ISPPROF
- My Favorites
- TSO Commands
- JES
 - Retrieved Jobs
 - POT01RUN:JOB00166 [CC 0012]
 - My Jobs
 - POT01DB:JOB00176 [CC 0012]
 - POT01DB:JOB00175 [CC 0000] (highlighted)
 - JES2::JESMSGLG
 - JES2::JESJCL
 - JES2::JESYMSG
 - CURSRAV4::SYSTSIN [0000]
 - CURSRAV4::CEEOPTS [0000]
 - CURSRAV4::SYSOUT [0000]
 - CURSRAV4::SYSTSPRT [0000]

4.3.5  Scroll down to see the various JOB steps. Notice that a small report has been printed.



The screenshot shows a terminal window titled "POT01DB:JOB00175.spool" displaying a series of job steps and system messages. A red arrow labeled '2' points to the scroll bar on the right side of the terminal window, indicating where to scroll down to view more content. To the right of the terminal is a "Remote Systems" browser pane titled "Team Artifacts". A red circle labeled '1' highlights the entry "POT01DB:JOB00175 [CC 0000]" under the "My Jobs" section. Other entries in the list include "POT01RUN:JOB00166 [CC 0012]", "POT01DB:JOB00176 [CC 0012]", and several JES and CURSRAV4 log files.

```

91    CPU:      0 HR 00 MIN 01.66 SEC   SRB:      0 HR 00 MIN 00.03 SEC
92    VIRT:  660K  SYS: 360K  EXT: 8816K  SYS: 10552K
93    ATB- REAL:          20K  SLOTS:          0K
94    VIRT- ALLOC:       10M  SHRD:          0M
95 IEF375I  JOB/POT01DB /START 2019224.0939
96 IEF033I  JOB/POT01DB /STOP 2019224.0940
97    CPU:      0 HR 00 MIN 01.66 SEC   SRB:      0 HR 00 MIN 00.03 SEC
98  TSOLIB ACTIVATE DA('DSNB10.SDSNLOAD')
99  DSN SYSTEM(DBBG)
100 RUN PROGRAM(DB2REGI) PLAN(DB2REGI) -
101 LIB('EMPOT.ZPICL.LOAD')
102 END
103     TEST(,,,DBM)
104 *DEPT*      ACC
105 *PERF-AVG*  0000000
106 *PERF-MIN*  0008
107 *PERF-MAX*  0008
108 *HOURS-AVG* 0001599
109 *HOURS-MIN* 0001599
110 *HPOURS-MAX* 0001599
111 ACC      .00 8.00 8.00      15.99 15.99 15.99
112 *DEPT*      FIN
113 *PERF-AVG*  0000000
114 *PERF-MIN*  0003
115 *PERF-MAX*  0009
116 *HOURS-AVG* 0002269
117 *HOURS-MIN* 0003245
118 *HPOURS-MAX* 0000889
  
```

- 4.3.6 ➔ Close all opened editors. (**Ctrl + Shift + F4**). Click **No** if asked to save changes.
Or just click on the of each opened editor.

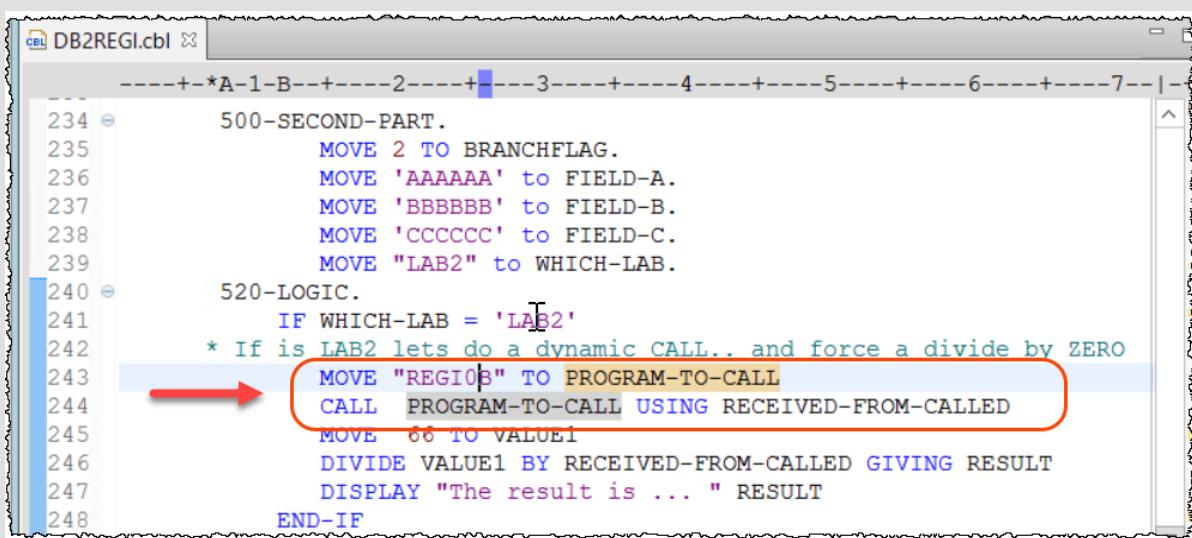
Section 5. Modify the COBOL code to fix the bug

You will work with a z/OS member using the editor and now we will be working with the assets located on the z/OS remote system.



What z/OS remote assets you will work with?

This is a batch program that reads DB2. In addition, this program does a Dynamic call to another COBOL program named **REGI0B**.



```

DB2REGI.cbl
-----+*A-1-B-----2-----+ 3-----+ 4-----5-----+ 6-----7-----+
234      500-SECOND-PART.
235      MOVE 2 TO BRANCHFLAG.
236      MOVE 'AAAAAAA' to FIELD-A.
237      MOVE 'BBBBBBB' to FIELD-B.
238      MOVE 'CCCCCC' to FIELD-C.
239      MOVE "LAB2" to WHICH-LAB.
240      520-LOGIC.
241      IF WHICH-LAB = 'LAB2'
242      * If is LAB2 lets do a dynamic CALL.. and force a divide by ZERO
243      MOVE "REGI0B" TO PROGRAM-TO-CALL
244      CALL PROGRAM-TO-CALL USING RECEIVED-FROM-CALLED
245      MOVE 66 TO VALUE1
246      DIVIDE VALUE1 BY RECEIVED-FROM-CALLED GIVING RESULT
247      DISPLAY "The result is ... " RESULT
248      END-IF

```

5.1 Using the Editor with z/OS COBOL programs

Before fixing the code let's explore some editor capabilities on the main DB2 COBOL program.

- 5.1.1 ➔ Using the z/OS Projects perspective and the z/OS Projects view, **double click** on **EMPOT01.POT.COBOL(DB2REGI)** to open the file using the editor.

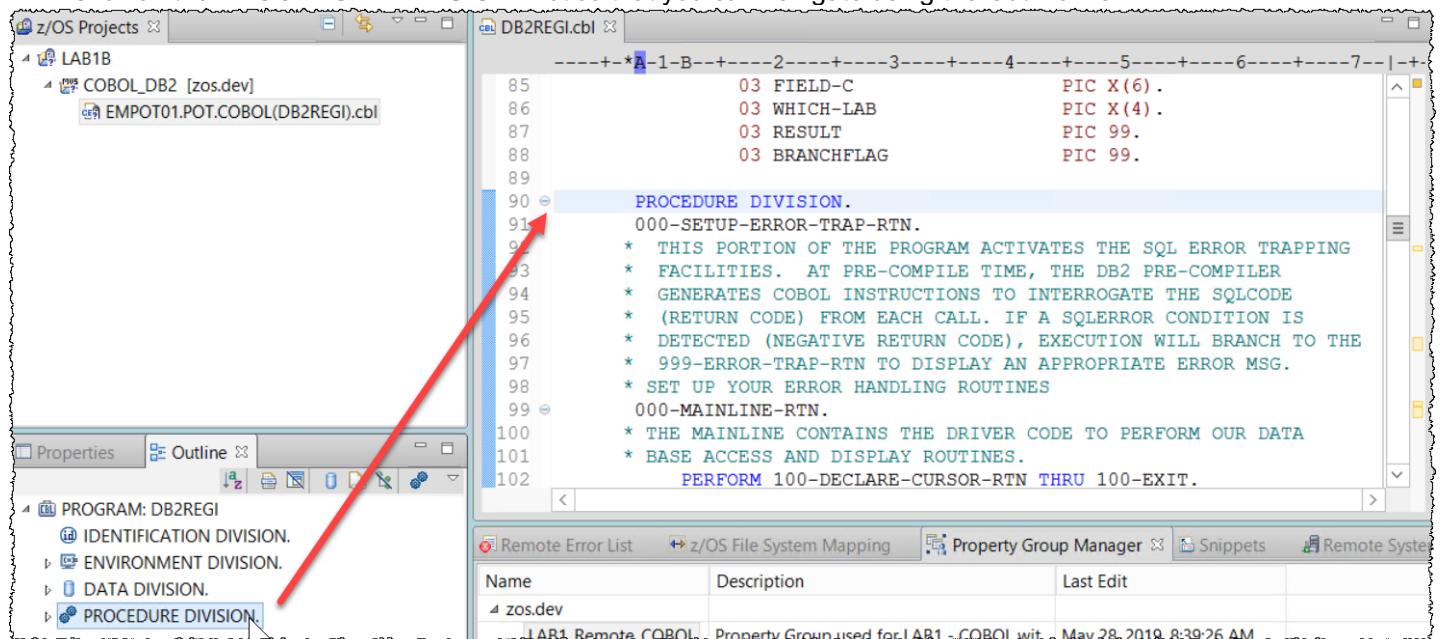
The screenshot shows the IBM Rational Application Developer interface. On the left, the 'z/OS Projects' view displays a project named 'LAB1B' which contains a 'COBOL_DB2 [zos.dev]' folder. Inside this folder, a file named 'EMPOT01.POT.COBOL(DB2REGI).cbl' is highlighted with a red oval and a red arrow points from it to the code editor on the right. The code editor window is titled 'DB2REGI.cbl' and contains the following COBOL code:

```
--+*A-1-B---+---2---+---3---+---4---+---5---+---6---+
IDENTIFICATION DIVISION.
PROGRAM-ID. DB2REGI.
*REMARKS. THIS PROGRAM JOINS TABLES, GROUPS DATA BY DEPT.
* AND DISPLAYS THE AVERAGE, MAXIMUM AND MINIMUM
* HOURS, AND PERFORMANCE EVALUATION BY DEPT.
* Modified by Regi to add DEAD CODE - Jan/2-14
* Modified by Regi to added test if -204 - Mar/2015
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-370.
OBJECT-COMPUTER. IBM-370.
DATA DIVISION.
WORKING-STORAGE SECTION.
* CODE THE NECESSARY DB2 INCLUDE STATEMENTS HERE
```

5.1.2 ► Click on the **Outline** tab (bottom and left) to see the *Outline* view.

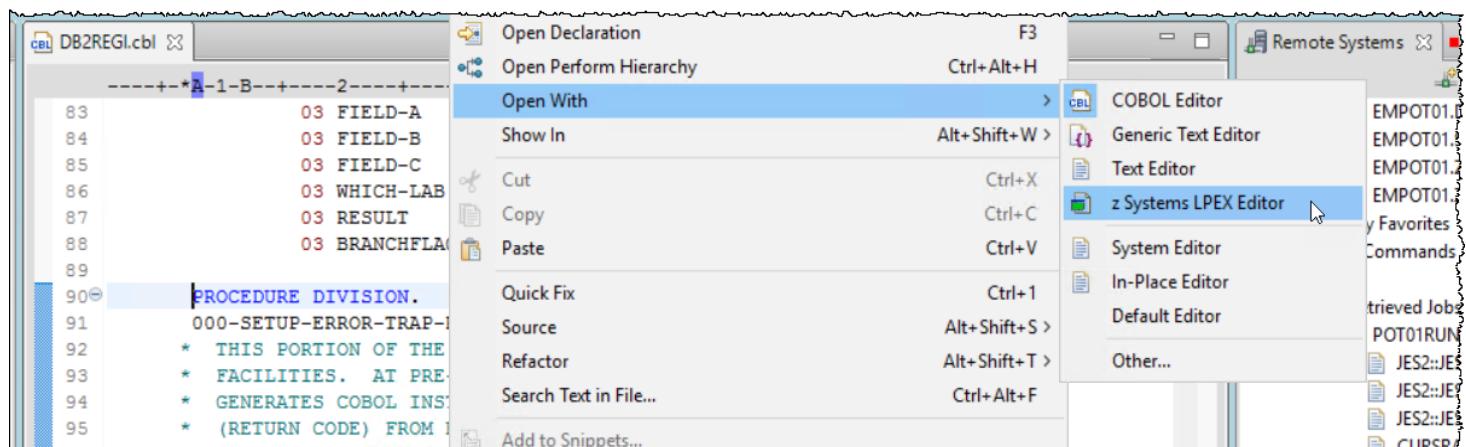
► Using the editor, browse the program and notice that the contents of the *Outline* view are synchronized with the COBOL source code and vice versa.

► Click on the **PROCEDURE DIVISION**. Notice that you can navigate using the *Outline* view.



5.1.3 If you prefer the ISPF editor, you might use another IDz editor called "LPEX editor".

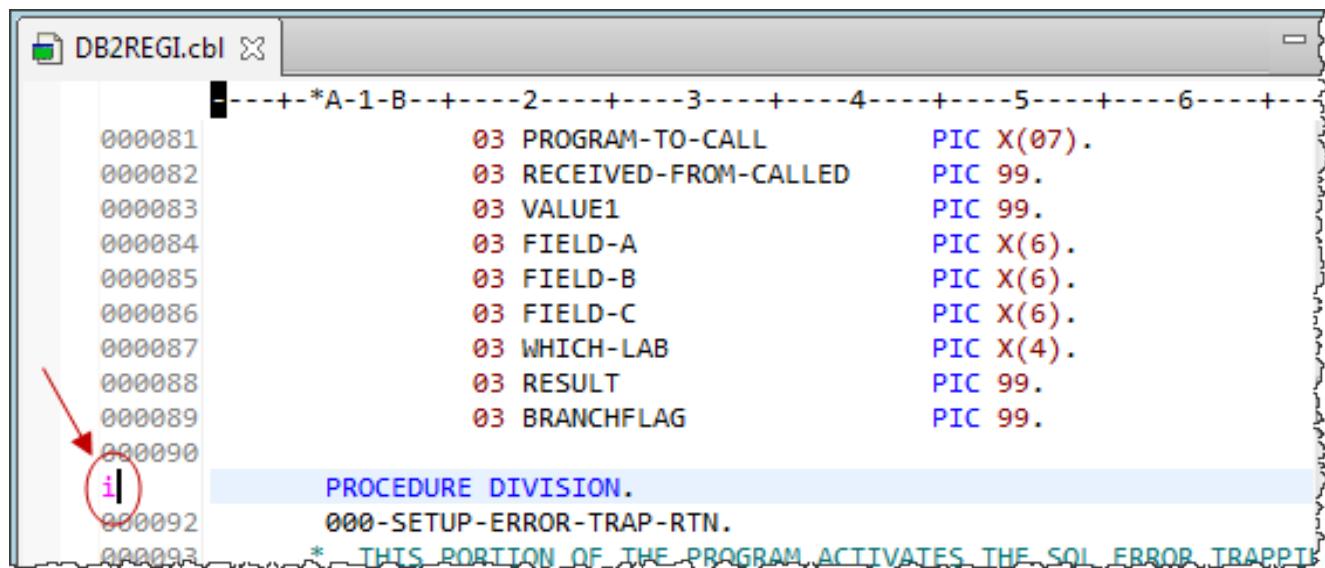
► To switch the editor, right click on the program and select **Open with > z Systems LPEX Editor**



5.1.4 ► Note that the column on left are similar as the prefix area of the ISPF editor..

You may type commands to add lines etc..

Like **I** to insert a line, **d** to delete, **m** to move, etc. **Try it.**



```
DB2REGI.cbl
-----+-----+-----+-----+-----+-----+
000081      03 PROGRAM-TO-CALL      PIC X(07).
000082      03 RECEIVED-FROM-CALLED  PIC 99.
000083      03 VALUE1                PIC 99.
000084      03 FIELD-A               PIC X(6).
000085      03 FIELD-B               PIC X(6).
000086      03 FIELD-C               PIC X(6).
000087      03 WHICH-LAB             PIC X(4).
000088      03 RESULT                PIC 99.
000089      03 BRANCHFLAG            PIC 99.
000090      PROCEDURE DIVISION.
000091      000-SETUP-ERROR-TRAP-RTN.
* THIS PORTION OF THE PROGRAM ACTIVATES THE SQL ERROR TRAPPI
```

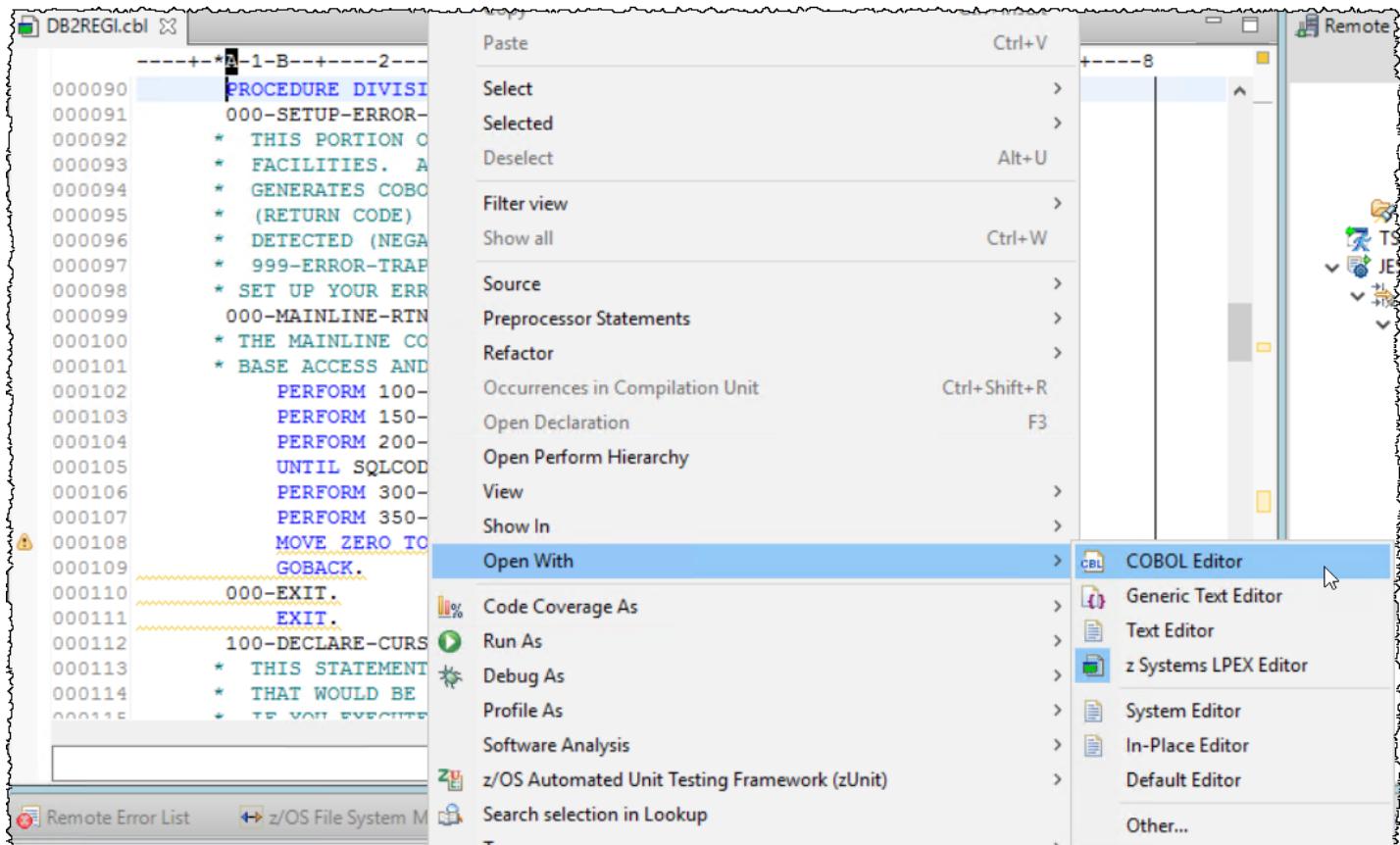
5.1.5 Switch back to **COBOL editor** (since the screen captures uses the COBOL editor)

► Right click on the program and select **Open with > COBOL Editor**

Click **No** for saving changes.

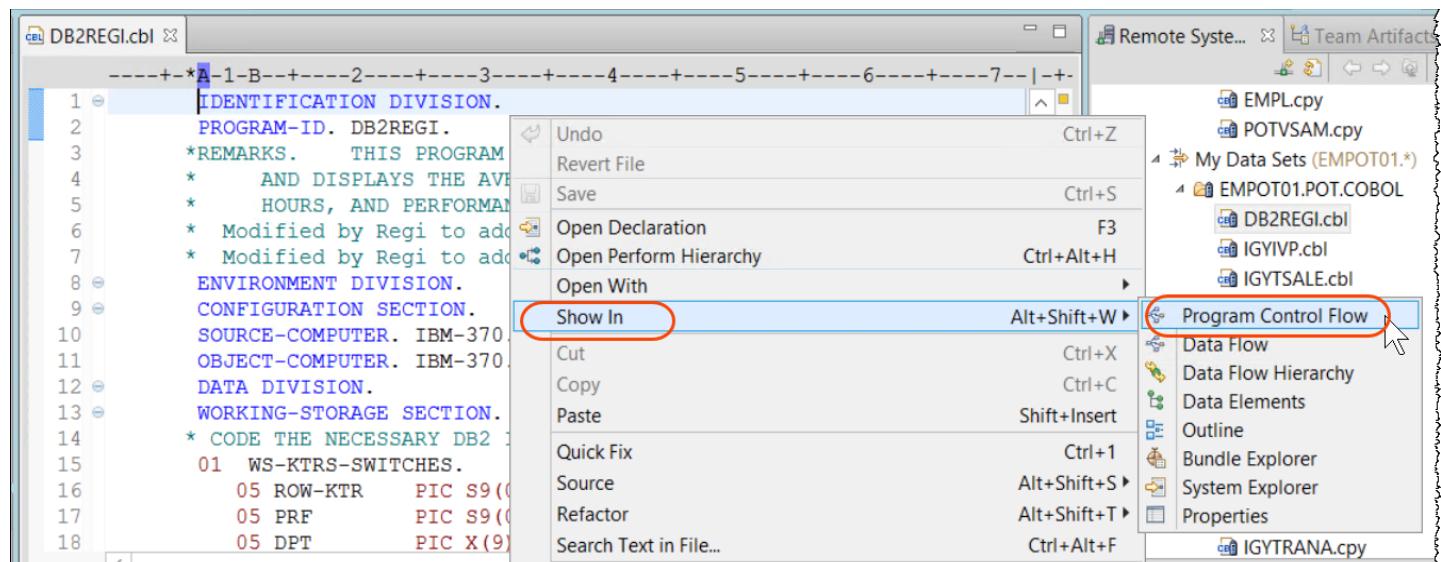
This editor is more like the Java Editor and gives you more capabilities. When you see all advantages, probably you will prefer this editor than the ISPF like editor (LPEX)..

Please keep this program opened as we will work on it later.



5.1.6 Let's see the program in more details..

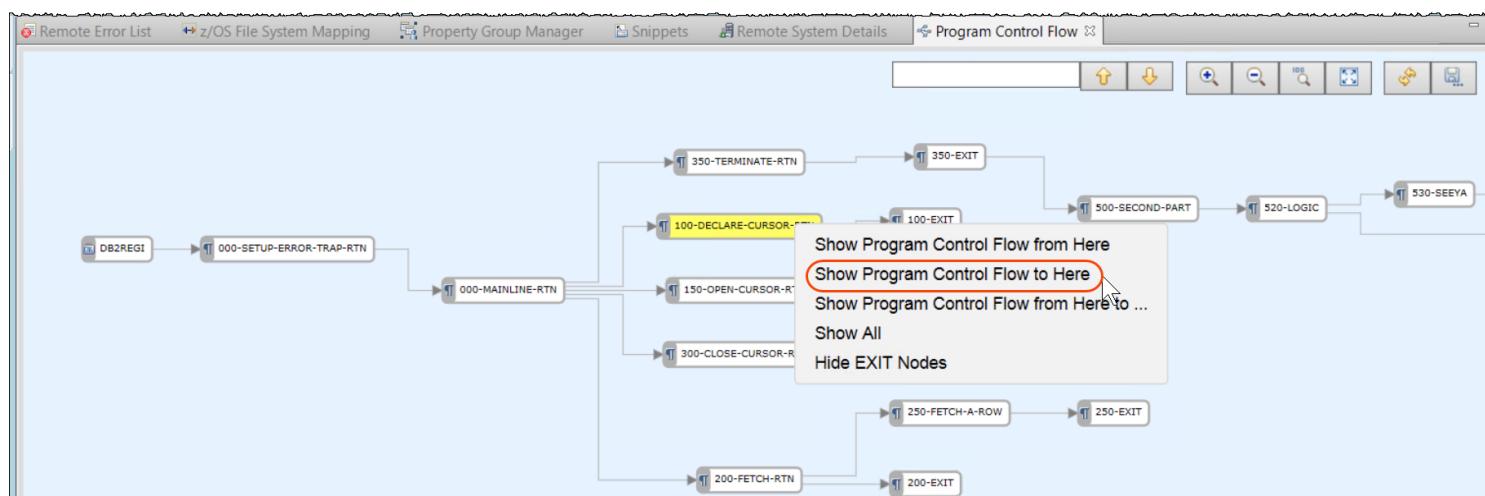
► Right click on the editor and select **Show In → Program Control Flow**



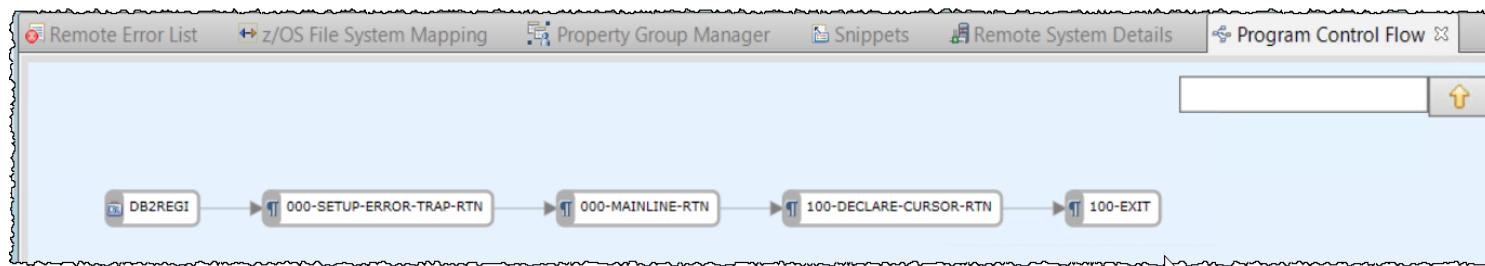
5.1.7 See the results under *Program Control Flow* view.

▶ Double click “**Program Control Flow**” title to have a bigger diagram (full page).

▶ Right-click on **100-DECLARE-CURSOR-RTN** box and select **Show Program Flow Control to here**.



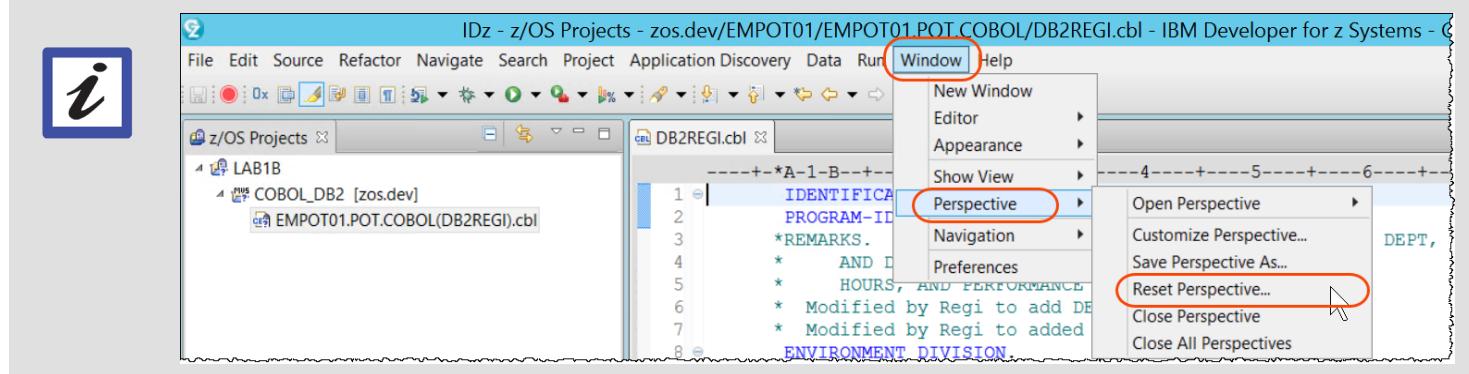
5.1.8 It shows the path to reach this paragraph.. Nice uh? That helps with extremely complex diagrams



The perspectives are not the same as here? Got confused with the opened views?

At any time, you may restore the perspective to the default. You may need that when you did mistakes and closed views you should not.

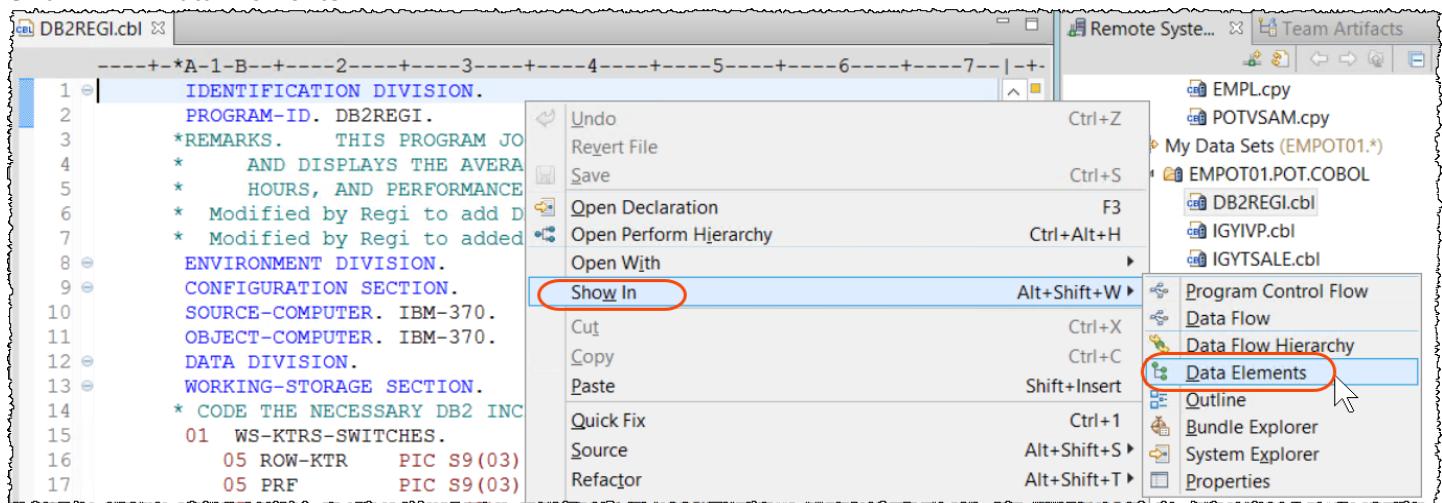
Just select **Windows > Perspective > Reset Perspective...** as below:



5.1.9 ► Double click "Program Control Flow" title to have the diagram smaller.

► Right click again on the editor and select

Show In → Data Elements

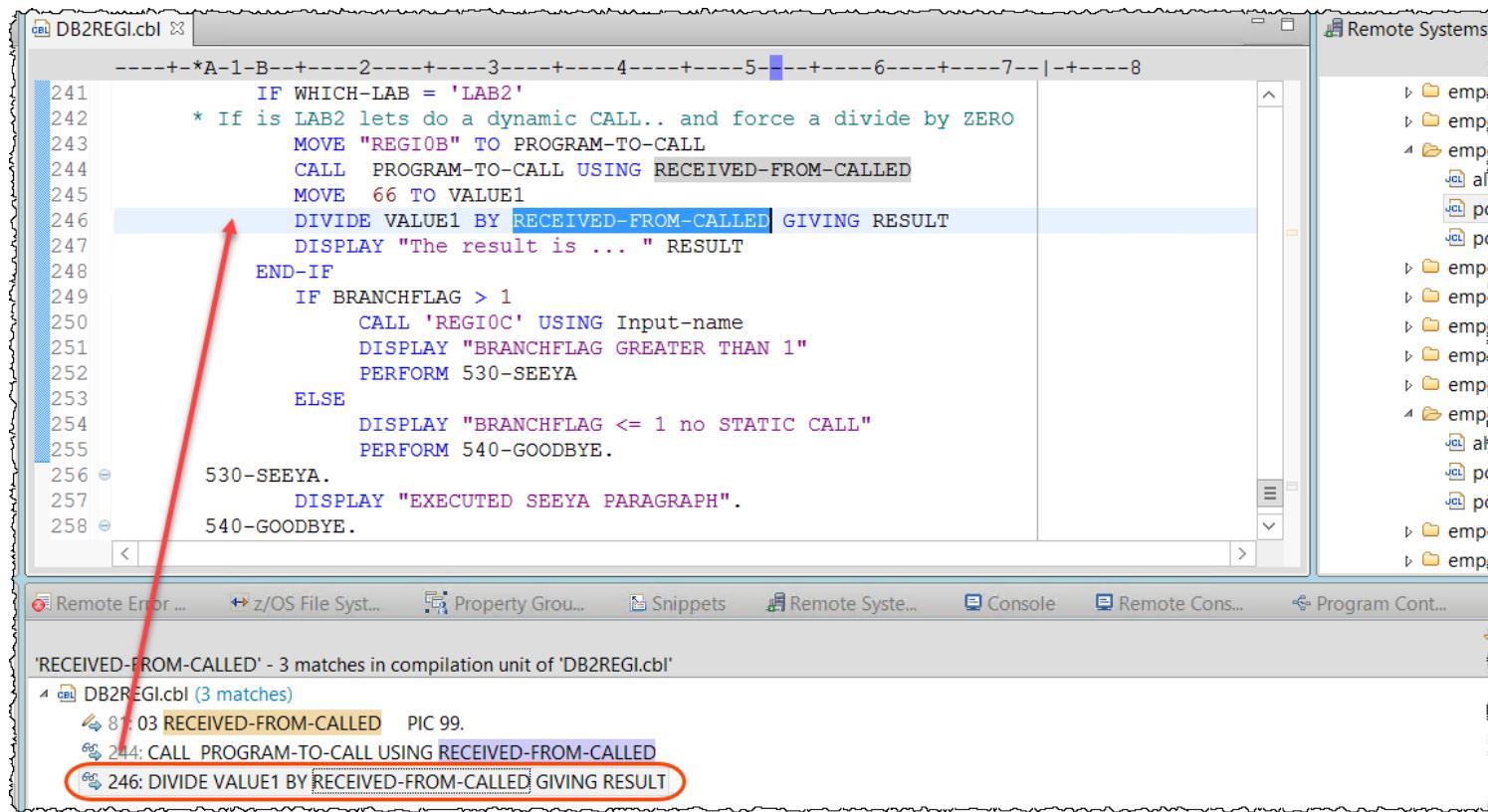


5.1.10 ► On Data Elements view, resize the view, scroll down and right click on RECEIVED-FROM-CALLED and select Occurrences in Compilation Unit

Name	Level	Top-level Item	Declaration	Declared In	Line Number	References	Item Type
PRF	5	WS-KTRS-SWITCHES	PIC S9(03)	DB2REGI.cbl	17	0	Data
PROGRAM-TO-CALL	3	WORK-FIELDS	PIC X(07)	DB2REGI.cbl	80	2	Data
PROJ	10	EMPL	PIC X(2)	EMPLcpy	46	0	Data
RECEIVED-FROM-CALLED	3	WORK-FIELDS	PIC 99	DB2REGI.cbl	81	2	Data
RESULT			PIC 99	DB2REGI.cbl	87	2	Data
ROW-KTR			PIC S9(03)	DB2REGI.cbl	16	3	Data
ROW-MSG				DB2REGI.cbl	61	1	Data
ROW-STAT			PIC Z99	DB2REGI.cbl	64	1	Data

5.1.11 ► Double click on line 246. It will show in the line that is abending.

Notice the colors. When the variable is referenced, it is blue and when it is modified is brown.



The screenshot shows the IBM Rational Developer for z/OS interface. The main window displays a COBOL program named DB2REGI.cbl. The code editor shows several lines of COBOL code, including an IF block that calls a program named REGI0B. The bottom panel shows search results for the keyword 'RECEIVED-FROM-CALLED', which appears three times in the program. The result for line 246 is circled in red, indicating it is the current focus.

```

241      IF WHICH-LAB = 'LAB2'
242          * If is LAB2 lets do a dynamic CALL.. and force a divide by ZERO
243          MOVE "REGI0B" TO PROGRAM-TO-CALL
244          CALL PROGRAM-TO-CALL USING RECEIVED-FROM-CALLED
245          MOVE 66 TO VALUE1
246          DIVIDE VALUE1 BY RECEIVED-FROM-CALLED GIVING RESULT
247          DISPLAY "The result is ... " RESULT
248      END-IF
249      IF BRANCHFLAG > 1
250          CALL 'REGI0C' USING Input-name
251          DISPLAY "BRANCHFLAG GREATER THAN 1"
252          PERFORM 530-SEYYA
253      ELSE
254          DISPLAY "BRANCHFLAG <= 1 no STATIC CALL"
255          PERFORM 540-GOODBYE.
256 530-SEYYA.
257          DISPLAY "EXECUTED SEYYA PARAGRAPH".
258 540-GOODBYE.

```

'RECEIVED-FROM-CALLED' - 3 matches in compilation unit of 'DB2REGI.cbl'

- 4 DB2REGI.cbl (3 matches)
 - 81:03 RECEIVED-FROM-CALLED PIC 99.
 - 244: CALL PROGRAM-TO-CALL USING RECEIVED-FROM-CALLED
 - 246: DIVIDE VALUE1 BY RECEIVED-FROM-CALLED GIVING RESULT

You need to fix the called program named **REGI0B** that is returning 0 on this variable as seen before with Fault Analyzer.

Exploring Data Flow

This capability is cool. *Program data flow* provides a graphical and hierarchical view of the data flow within a COBOL program. You can use this feature to examine how a data element is populated, modified, or written elsewhere.

5.1.12 Double click on line 81. It will show where RECEIVED-FROM-CALLED is defined.

DB2REGI.cbl

```

-----+*A-1-B---+---2---+---3---+---4---+---5---+---6---+---7---+---8
      01 WORK-FIELDS.
      03 INPUT-NAME          Pic x(30).
      03 PROGRAM-TO-CALL    PIC X(07).
      03 RECEIVED-FROM-CALLED PIC 99.
      03 VALUE1              PIC 99.
      03 FIELD-A              PIC X(6).
      03 FIELD-B              PIC X(6).
      03 FIELD-C              PIC X(6).
      03 WHICH-LAB             PIC X(4).
      03 RESULT                PIC 99.
      03 BRANCHFLAG            PIC 99.

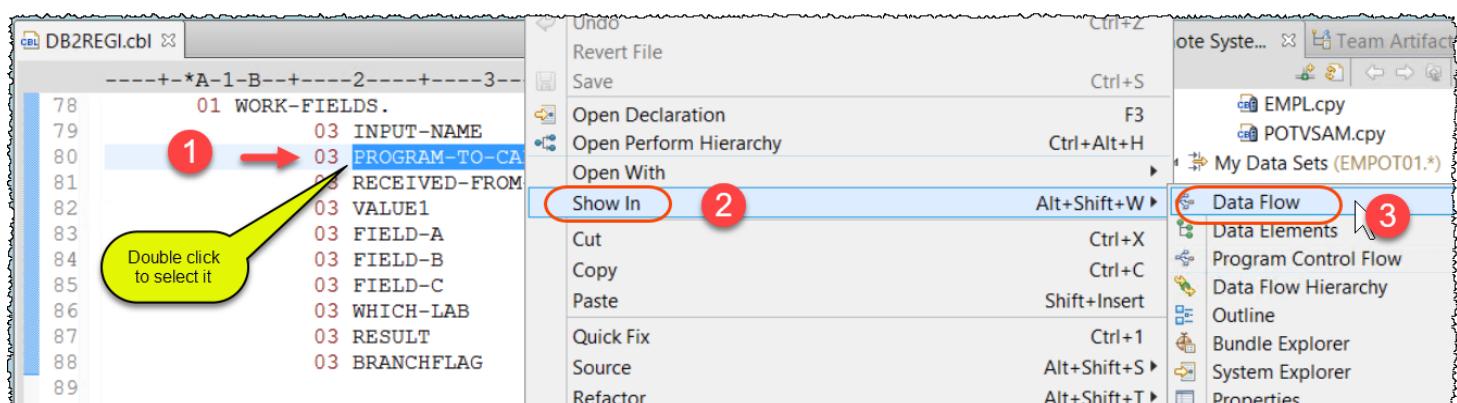
      PROCEDURE DIVISION.
      000-SETUP-ERROR-TRAP-RTN.
      * THIS PORTION OF THE PROGRAM ACTIVATES THE SQL ERROR TRAPPING
      * FACILITIES. AT PRE-COMPILATION TIME, THE DB2 PRE-COMPILER
      * GENERATES COBOL INSTRUCTIONS TO INTERROGATE THE SQLCODE
      * (RETURN CODE) FROM EACH CALL. IF A SQLERROR CONDITION IS
      * DETECTED (NEGATIVE RETURN CODE), EXECUTION WILL BRANCH TO THE
      * 999-ERROR-TRAP-RTN TO DISPLAY AN APPROPRIATE ERROR MSG.
      * SET UP YOUR ERROR HANDLING ROUTINES
      000-MAINLINE-RTN.
      * THE MAINLINE CONTAINS THE DRIVER CODE TO PERFORM OUR DATA
      * BASE ACCESS AND DISPLAY ROUTINES.
      000-DEclare CURSOR PTVSAM_100 EXIT.

```

RECEIVED-FROM-CALLED - 3 matches in compilation unit of 'DB2REGI.cbl'

- 81: 03 RECEIVED-FROM-CALLED PIC 99.
- 244: CALL PROGRAM-TO-CALL USING RECEIVED-FROM-CALLED
- 246: DIVIDE VALUE1 BY RECEIVED-FROM-CALLED GIVING RESULT

5.1.13 ➔ Select **PROGRAM-TO-CALL** (on line 80) double clicking on it, **right click** and select **Show In > Data Flow**. (the options order can differ from the screen capture here).



5.1.14 ►| Move the cursor to the line between **REGI0B** and **03 PROGRAM-TO-CALL**.

►| Clicking in this line shows the program statement where “**REGI0B**” is moved to **PROGRAM-TO-CALL**.

The screenshot shows the Rational Developer for System z interface. The top window is titled "DB2REGI.cbl" and displays COBOL source code. Line 243 contains the statement "MOVE \"REGI0B\" TO PROGRAM-TO-CALL". A red arrow points from this line down to the Navigator view below. The Navigator view shows a node for "REGI0B" with a circled "x" icon, indicating it is selected. A tooltip above the node reads "DB2REGI.cbl:243: MOVE \"REGI0B\" TO PROGRAM-TO-CALL". The bottom navigation bar includes tabs for "Remote Error List", "z/OS File Syste...", "Property Group ...", "Snippets", "Remote System ...", and "Data".

```

-----+--A-1-B--+---2---+---3---+---4---+---5---+---6---+---7---+---8---+
240      520-LOGIC.
241          IF WHICH-LAB = 'LAB2'
242              * If is LAB2 lets do a dynamic CALL.. and force a divide by ZERO
243                  MOVE "REGI0B" TO PROGRAM-TO-CALL
244                      CALL PROGRAM-TO-CALL USING RECEIVED-FROM-CALLED
245                      MOVE 66 TO VALUE1
246                      DIVIDE VALUE1 BY RECEIVED-FROM-CALLED GIVING RESULT
247                      DISPLAY "The result is ... " RESULT
248          END-IF
249          IF BRANCHFLAG > 1
250              CALL 'REGI0C' USING Input-name
251              DISPLAY "BRANCHFLAG GREATER THAN 1"

```

5.1.15 ►| Close all opened editors if still opened. (**CTRL + Shift + F4**)

Or just click on the of each opened editor.

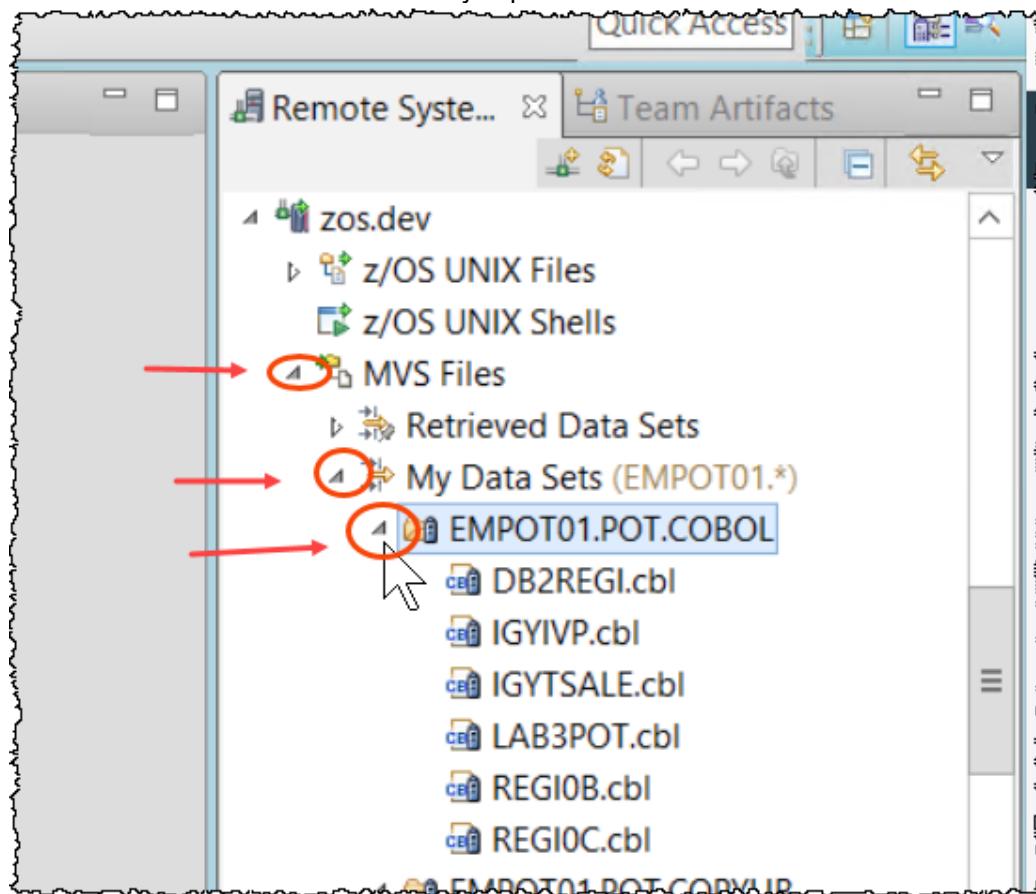
5.2 Fixing the program REGI0B

You need to fix the called program named **REGI0B** that is returning 0 on this variable as seen before with Fault Analyzer and the debugger.

This program is on your **PDS EMPOT01.POT.COBOL** .

5.2.1 ►| Using **Remote Systems** view **scroll back**, expand **MVS Files**, **My Data Sets** and

EMPOT01.POT.COBOL. if not already expanded.



5.2.4 ► Double click on the program **REGI0B.cbl** to edit it.

The screenshot shows the Rational Application Developer interface. On the left is the COBOL editor window titled "REGI0B.cbl" containing the following code:

```

1  * Identification Division.
2  Program-ID. "REGI0B".
3  ****
4  * This program is called.
5  * It returns a value (0)
6  * This will cause an error when this value is used in a division
7  ****
8  Data Division.
9  Working-Storage Section.
10 Linkage Section.
11 01 Recvd_Parms.
12      05 In-value      Pic 99.
13
14 Procedure Division using Recvd_Parms.
15     Move 0 to In-value.
16     Goback.
17

```

On the right is the "Remote System..." browser pane, which lists the following files under "zos.dev":

- z/OS UNIX Files
- z/OS UNIX Shells
- MVS Files
 - Retrieved Data Sets
 - My Data Sets (EMPOT01.*)
 - EMPOT01.POT.COBOL
 - DB2REGI.cbl
 - IGYIVP.cbl
 - IGYSALE.cbl
 - LAB3POT.cbl
 - REGI0B.cbl** (highlighted with a red circle)
 - REGI0C.cbl
- EMPOT01.POT.COPYLIB

5.2.5 ► Move the cursor after the **In-value** and press **enter** to add a blank line

The screenshot shows the COBOL editor window titled "*REGI0B.cbl" with the same code as before. A yellow callout bubble points to the cursor position at the end of line 12, just after the word "In-value". The callout contains the text: "This is the COBOL editor. Just press enter to add a new line." The "Remote System..." browser pane on the right is identical to the previous screenshot.

5.2.6 You can now take advantage of the **editor content assist...**

► On the column 12 as shown below, type **m** and press **Ctrl + Space**

5.2.7 ► Select the statement **MOVE** (you can use the mouse double click or select and press enter)

```
Linkage Section.  
01 Recvd-Parms.  
    05 In-value      Pic 99.  
  
Procedure Division using Recvd-Parms.  
    Move 0 to In-value.  
    1  m  
    G  
        RBC MERGE  
        RBC MOVE  
        RBC MULTIPLY  
        RBC MULTIPLY - NOT ON SIZE ERROR - END-MULTIPLY  
        RBC MULTIPLY - ON SIZE ERROR - END-MULTIPLY  
        RBC MULTIPLY - ON SIZE ERROR - NOT ON SIZE ERROR - END-M
```

5.2.8 ► Type "XXX to"

```
Procedure Division using Recvd-Parms.  
    Move 0 to In-value.  
    MOVE XXX to  
    Goback.
```

5.2.9 ► Press **Ctrl + Space** to list the variables and select **In-value**.

Notice that variables could be defined in copybooks.

The screenshot shows the COBOL editor with the following code in the Procedure Division:

```
Procedure Division using Recvd-Parms.
  Move 0 to In-value.
  MOVE XXX to |
```

A code completion window is open at the cursor position, showing the following suggestions:

- In-value (highlighted with a red oval)
- In-value IN Recvd-Parms
- ABC JSON-CODE
- ABC LENGTH OF
- ABC Recvd-Parms
- ABC RETURN-CODE
- ABC XML-CODE
- ABC XML-EVENT
- ABC XML-NTEXT
- ABC XML-TEXT

The status bar at the bottom of the editor window displays the message: "Press 'CTRL+SPACE' to view Template Proposals".

A yellow mark shows that something is wrong.

The screenshot shows the COBOL editor with the following code in the Procedure Division:

```
Procedure Division using Recvd-Parms.
  Move 0 to In-value.
  MOVE XXX to In-value
  Goback.
```

A yellow warning icon is circled in red on the left margin, indicating an error.

5.2.10 ► Move the cursor to the yellow mark to see what the error is.

You are finding this error without using the z/OS compilation. If you were using ISPF you would see that only after submitting this JOB to the z/OS COBOL compiler.

Productivity and z/OS CPU saving. You are using the power of the smart editor.

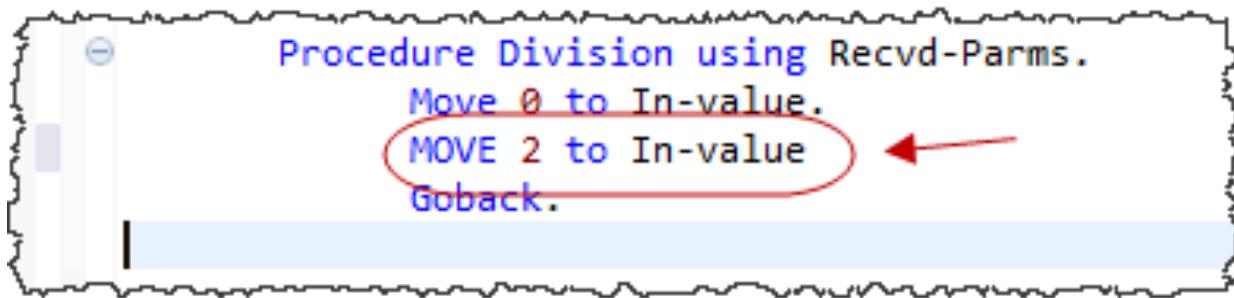
The screenshot shows the COBOL editor with the following code in the Procedure Division:

```
Procedure Division using Recvd-Parms.
  Move 0 to In-value.
```

A yellow tooltip box is displayed over the yellow warning icon on the left margin, containing the following error message:

Multiple markers at this line
 - Unable to resolve reference to XXX
 - 1 added line

5.2.11 ►| Change from **XXX** to **2** as below. Note that the icon  will go away.

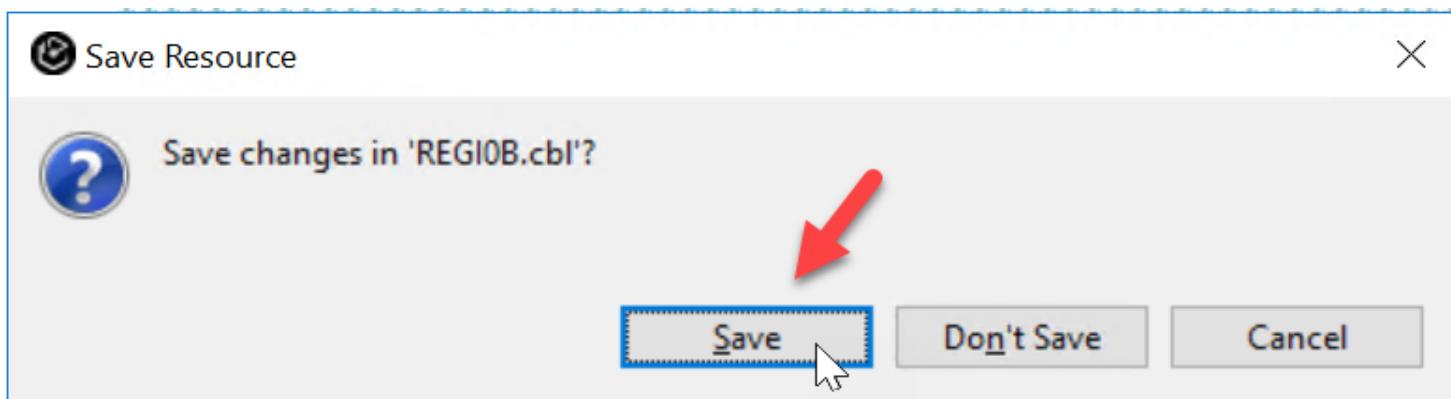


5.3 Compile and link REGI0B.

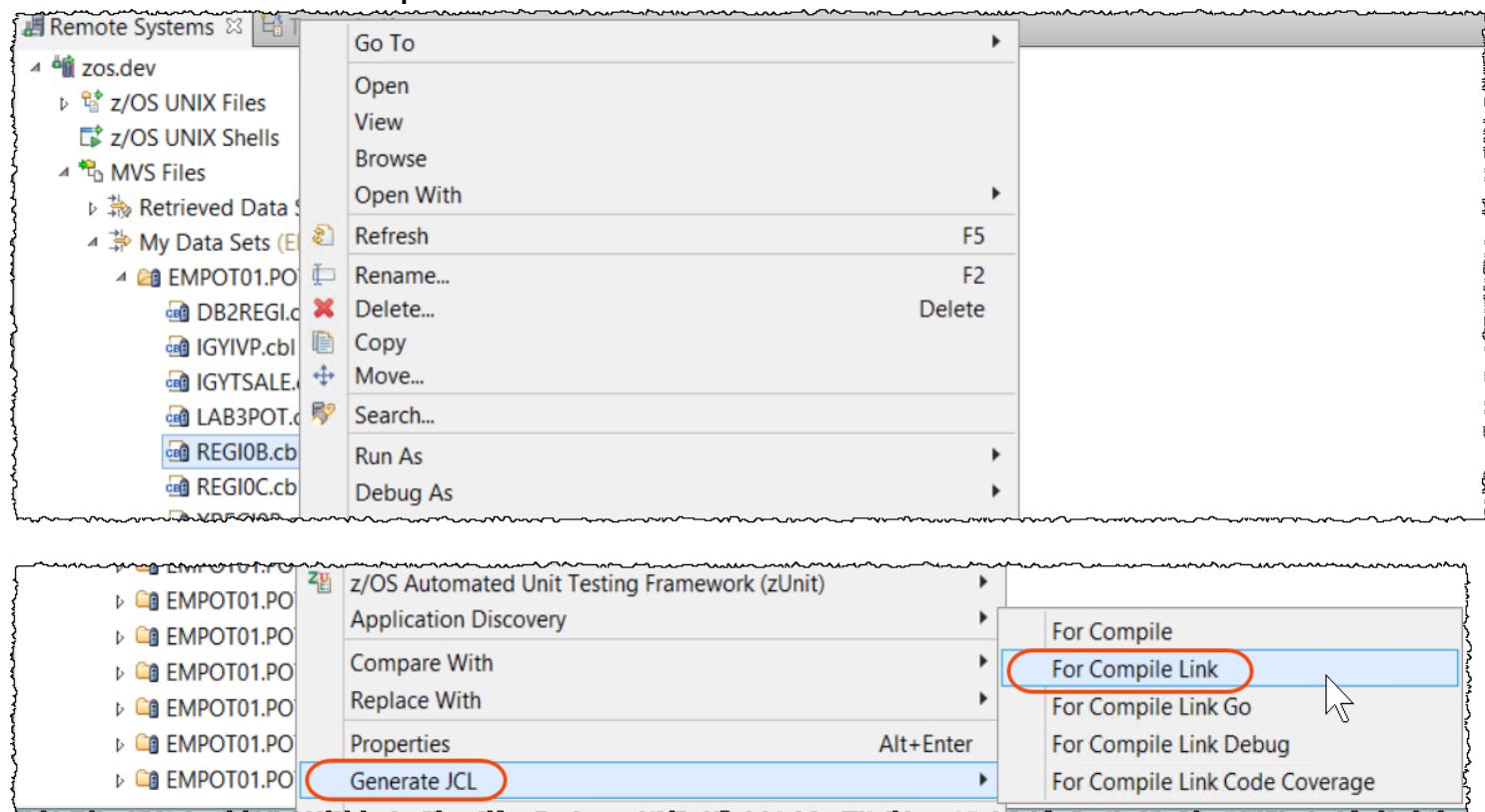
You need to compile and link the program that you just changed. You could use a JCL or let IDz generate a JCL for you. Once you have a Property Group associated to the COBOL code a JCL file could be generated on the fly when you need it. You have associated the property Group at step 7.2.1 above.

5.3.1 ►| Click **Ctrl + Shift + F4** to close the editors. Or just left click on the  of each opened editor.

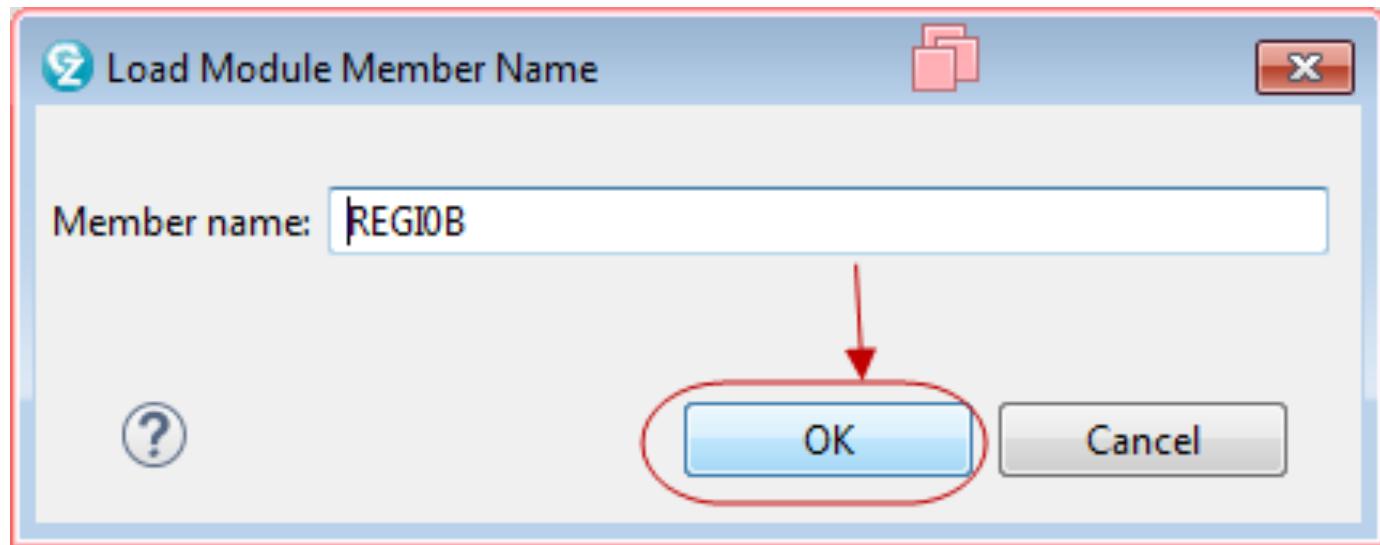
►| Click **Save** to save the changes.



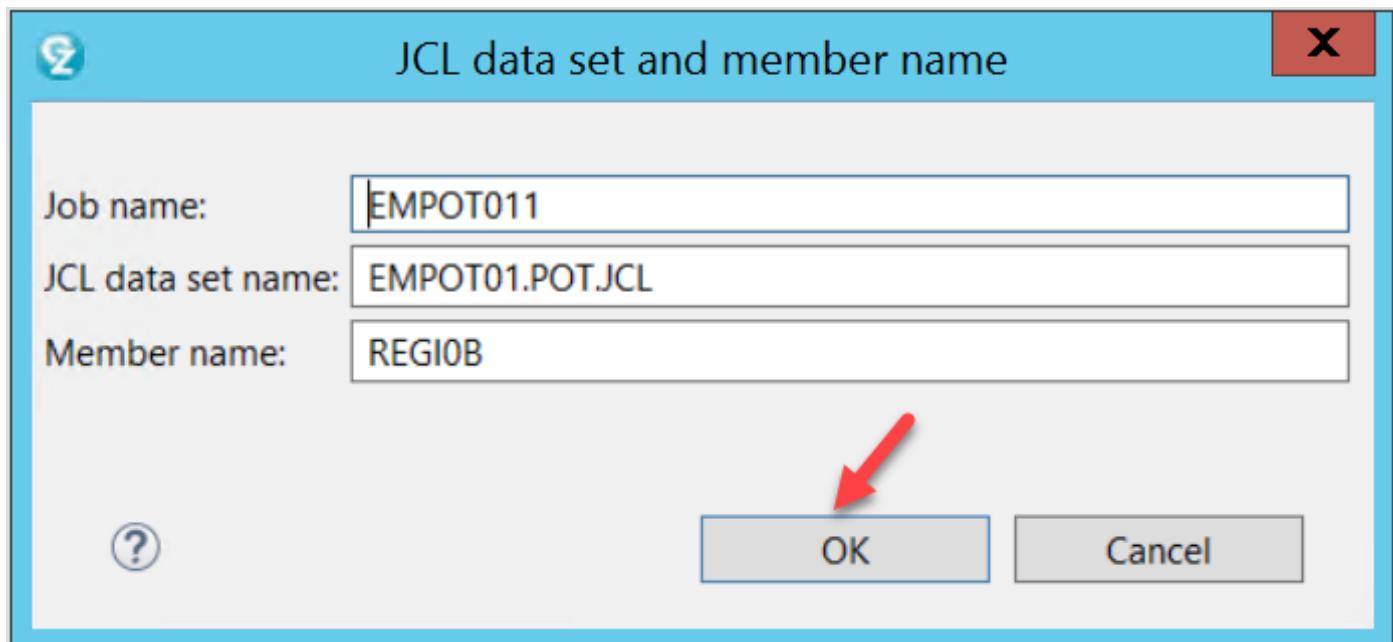
5.3.2 ► Using the Remote Systems view right click on **REGI0B.cbl** and select **Generate JCL > For Compile Link**.



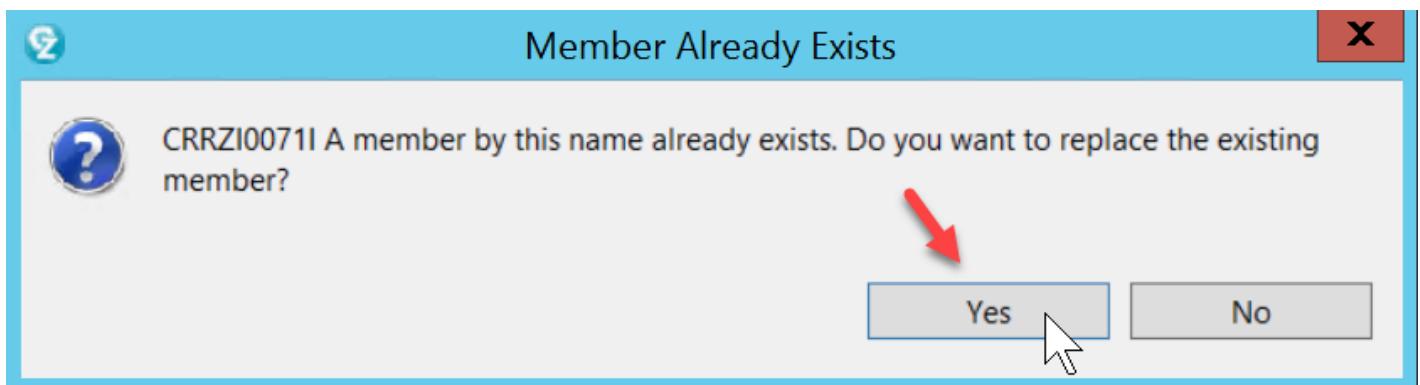
5.3.3 ► Accept the default member name and click **OK**



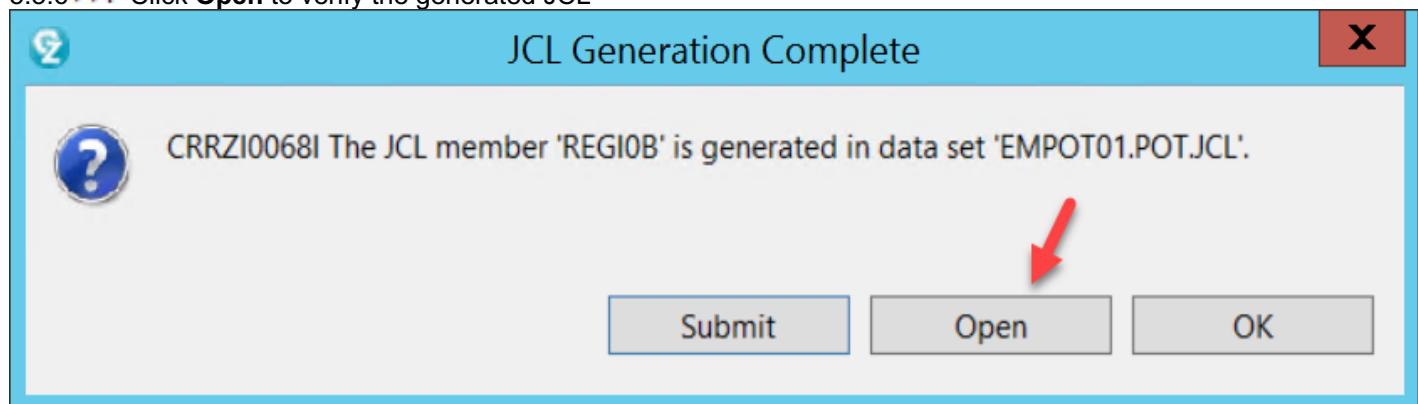
5.3.4 ► Accept the default and click **OK**



5.3.5 ► Click **Yes** if the member already exists



5.3.6 ► Click **Open** to verify the generated JCL

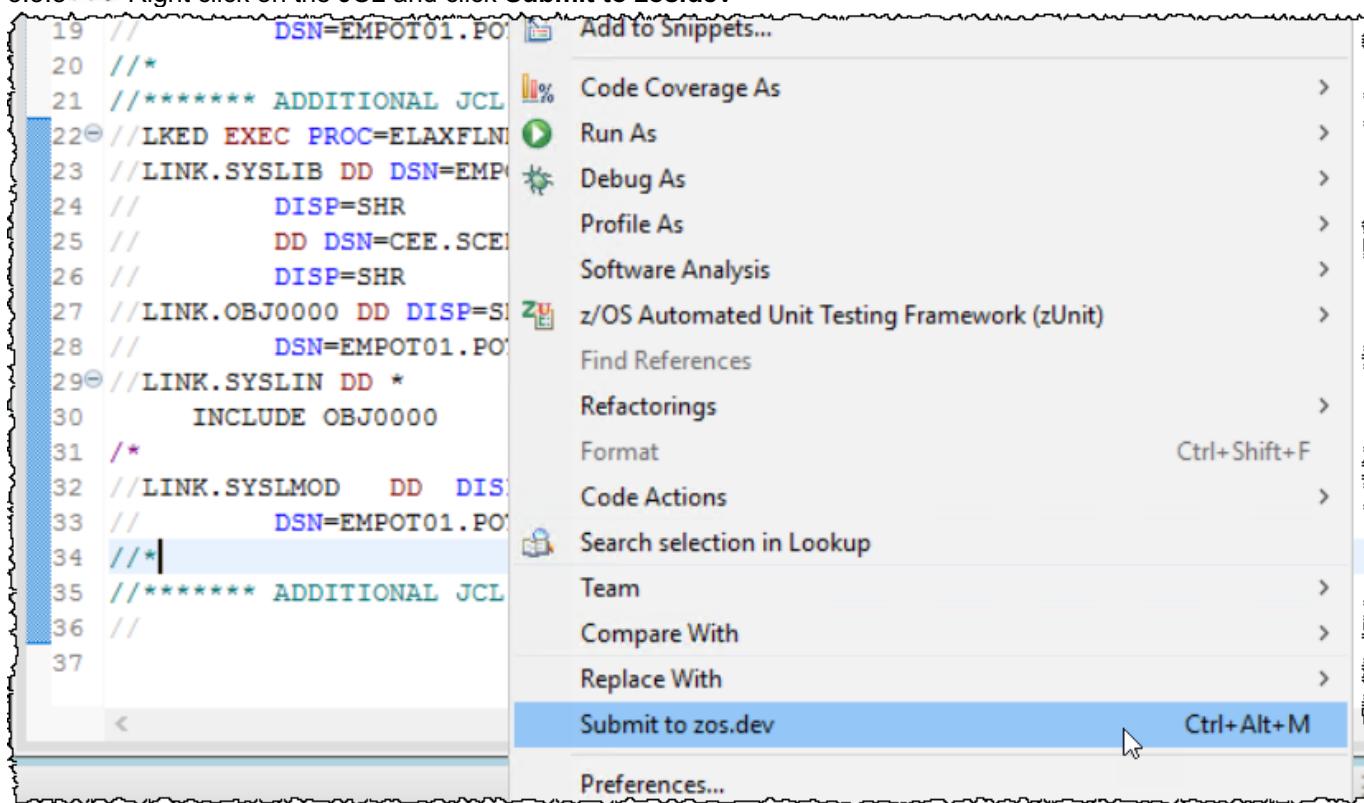


5.3.7 ► Scroll down and notice that the *Load module* will be created on the PDS: **EMPOT01.POT.LOAD**.

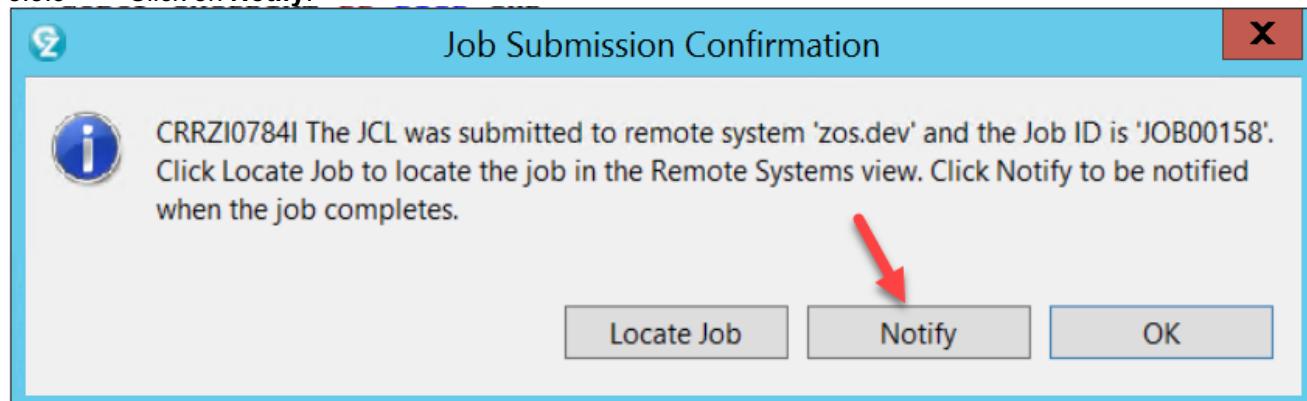
```
JCL REGI0B.jcl

15 //      DSN=EMPOT.ZPICL.COPYLIB
16 //COBOL.SYSXMLSD DD DUMMY
17 //COBOL.SYSIN DD DISP=SHR,
18 //      DSN=EMPOT01.POT.COBOL(REGI0B)
19 /**
20 //***** ADDITIONAL JCL FOR COMPILE HERE *****
21 //LKED EXEC PROC=ELAXFLNK
22 //LINK.SYSLIB DD DSN=EMPOT01.POT.OBJ,
23 //      DISP=SHR
24 //      DD DSN=CEE.SCEELKED,
25 //      DISP=SHR
26 //LINK.OBJ0000 DD DISP=SHR,
27 //      DSN=EMPOT01.POT.OBJ(REGI0B)
28 //LINK.SYSLIN DD *
29     INCLUDE OBJ0000
30 /*
31 //LINK.SYSLMOD DD DISP=SHR,
32 //      DSN=EMPOT01.POT.LOAD(REGI0B)
33 /**
34 //***** ADDITIONAL JCL FOR LINK HERE *****
35 //
```

5.3.8 ► Right click on the JCL and click **Submit to zos.dev**



5.3.9 ► Click on **Notify**.



5.3.10 You MUST have 000 as return code.

▶ Click on **EMPOT011:JOB00xxx**

```

20 //***** ADDITIONAL JCL FOR COMPILE HERE *****
21 //LKED EXEC PROC=ELAXFLNK
22 //LINK.SYSLIB DD DSN=EMPOT01.POT.OBJ,
23 //      DTCP=SPD
[5/28/19, 10:34 AM] Job JOB00158 is being submitted
[5/28/19, 10:34 AM] Job EMPOT011:JOB00158 ended with completion code CC 0000

```

5.3.11 ▶ Expand **EMPOT01:JOB00xxx** and verify the results double clicking on **LKED:LINK:SYSPRINT**.

▶ Scroll down and verify that the load module **REGI0B** invoked by your DB2 COBOL program is now created in **EMPOT01.POT.LOAD**.

```

298 SAVE OPERATION SUMMARY:
299
300 MEMBER NAME          REGI0B
301 LOAD LIBRARY         EMPOT01.POT.LOAD
302 PROGRAM TYPE         PROGRAM OBJECT (FORMAT 3)
303 VOLUME SERIAL        C2DBAR
304 DISPOSITION          ADDED NEW
305 TIME OF SAVE         09.33.57 MAY 28, 2019
306
307
308 SAVE MODULE ATTRIBUTES:
309
310 AC                  000
311 AMODE               31
312 COMPRESSION          NONE
313 DC                  NO
314 EDITABLE             YES
315 EXCEEDS 16MB         NO
316 EXECUTABLE           YES
317 LONGPARAM            NO
318 MIGRATABLE           NO

```

5.3.10 ▶ Use **Ctrl + Shift + F4** to close all editors.

Or just click on the of each opened editor

5.4 Execute the COBOL/DB2 program

On this step, you will explore some capabilities of the JCL editor and modify the JCL for execution.

- 5.4.1 ► Using *Remote System View*, scroll back to locate the file **pot01run.jcl** under the folder **Local/Local Files/ADF_POT/empot01** and **double click** to edit..

```

JCL pot01run.jcl
-----+-----+-----+-----+-----+-----+-----+-----+
1 //POT01RUN JOB ,                                |-----8
2 // MSGCLASS=H,MSGLEVEL=(1,1),TIME=1440,REGION=70M,COND=(16,LT) |
3 /*      ZPICL Debug                            |
4 //CURSRAV4 EXEC PGM=IKJEFT01,DYNAMNBR=20    |
5 //STEPLIB   DD DSN=EMPOT01.POT.LOAD,DISP=SHR |
6 //          DD DSN=EMPOT.ZPICL.LOAD,DISP=SHR |
7 /*          DD DISP=SHR,DSN=IBMUUSER.VER1.SEQAMOD |
8 /*          DD DISP=SHR,DSN=IBMUUSER.VER1.SEQAAUTH |
9 //          DD DSN=DSNB10.SDSNLOAD,DISP=SHR     |
10 //         DD DSN=DSNB10.DB2G.RUNLIB.LOAD,DISP=SHR |
11 //          DD DISP=SHR,DSN=FEK910.SFEKAUTH |
12 //SYSPRINT  DD SYSOUT=*                         |
13 //SYSOUT    DD SYSOUT=*                         |
14 //SYSTSPRT  DD SYSOUT=*                         |
15 //SYSTSIN   DD *                               |
16 TSOLIB ACTIVATE DA('DSNB10.SDSNLOAD')        |
17 DSN SYSTEM(DB2G) -                           |
18 RUN PROGRAM(DB2REGI) PLAN(DB2REGI) -          |
19 LIB('EMPOT.ZPICL.LOAD')                      |
20 END                                           |
21 /*                                           |
22 //                                          |

```

- 5.4.2 Notice on bottom left corner the same *Outline* view that you have for the COBOL program.
It will help to navigate on large JCL members.

- Expand **CURSRAV4 EXEC PGM=** and click on **STEPLIB**

5.4.3 ➔ Notice that your PDS where **REG10B** is created is already on the STEPLIB.

The screenshot shows the z/OS Studio interface with two main windows:

- z/OS Projects** window (left):
 - Project: LAB1B
 - Source: COBOL_DB2 [zos.dev]
 - File: EMPOT01.POT.COBOL(DB2REGI).cbl
- JCL pot01run.jcl** window (right):


```
--+---1---+---2---+---3---+---4---+---5---+---6
1 //POT01RUN JOB ,
2 // MSGCLASS=H,MSGLEVEL=(1,1),TIME=1440,REGION=70M,COND=(16,LT)
3 ///* ZPICL Debug
4 //CURSRAV4 EXEC PGM=IKJEFT01,DYNAMNBR=20
5 //STEPLIB DD DSN=EMPOT01.POT.LOAD,DISP=SHR
6 //          DD DSN=EMPOT.ZPICL.LOAD,DISP=SHR
7 //          DD DISP=SHR,DSN=IBMUSER.VER1.SEQAMOD
8 //          DD DISP=SHR,DSN=IBMUSER.VER1.SEQAAUTH
9 //          DD DSN=DSNB10.SDSNLOAD,DISP=SHR
10 //         DD DSN=DSNB10.DBBG.RUNLIB.LOAD,DISP=SHR
11 //         DD DISP=SHR,DSN=FEK910.SFEKAUTH
12 //SYSPRINT DD SYSOUT=*
13 //SYSOUT DD SYSOUT=*
14 //SYSTSPPRT DD SYSOUT=*
15 //SYSTSIN DD *
16 TSOLIB ACTIVATE DA('DSNB10.SDSNLOAD')
17 DSN SYSTEM(DBBG)
18 RUN PROGRAM(DB2REGI) PLAN(DB2REGI) -
19 LIB('EMPOT.ZPICL.LOAD')
20 END
21 /*
22 //
```

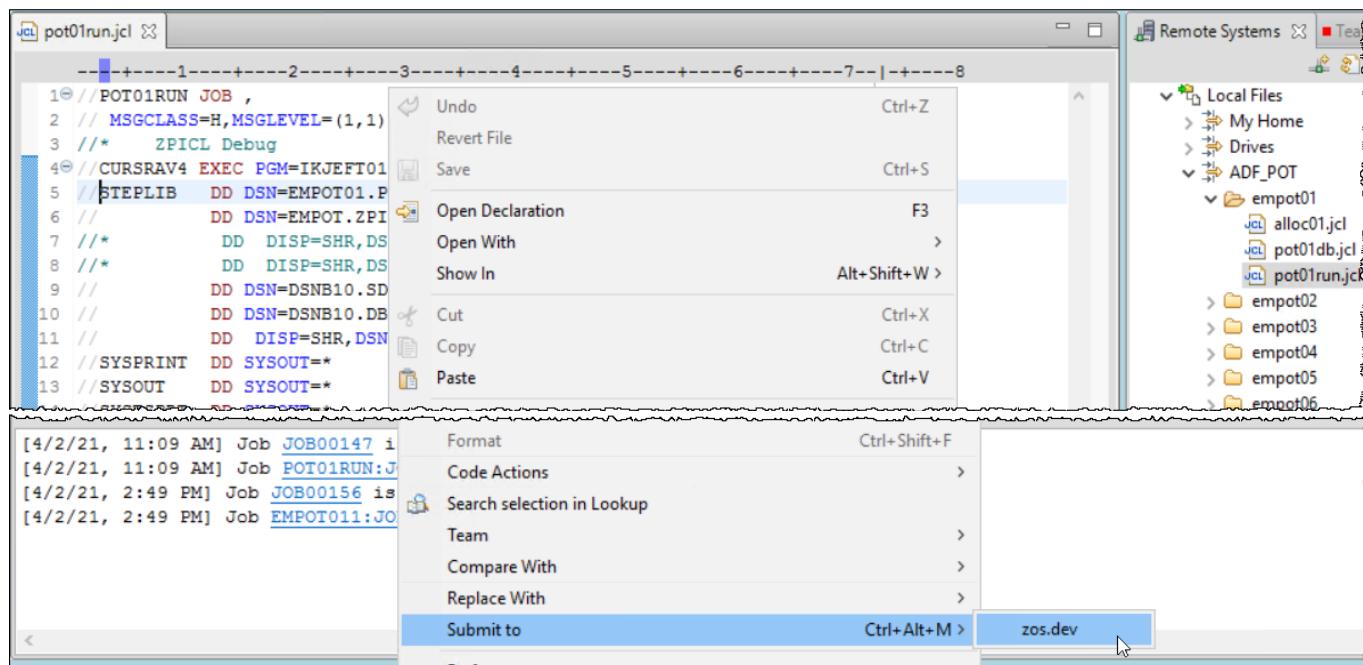
Annotations in the screenshot:

- A red arrow labeled **1** points from the **Properties** tab in the left window to the **Properties** tab in the right window.
- A red circle labeled **2** highlights the **STEPLIB** entry in the JCL outline.
- A red oval highlights the **STEPLIB DD DSN=EMPOT01.POT.LOAD,DISP=SHR** line in the JCL code.
- A red arrow points from the highlighted line in the JCL code to the corresponding entry in the Properties view.

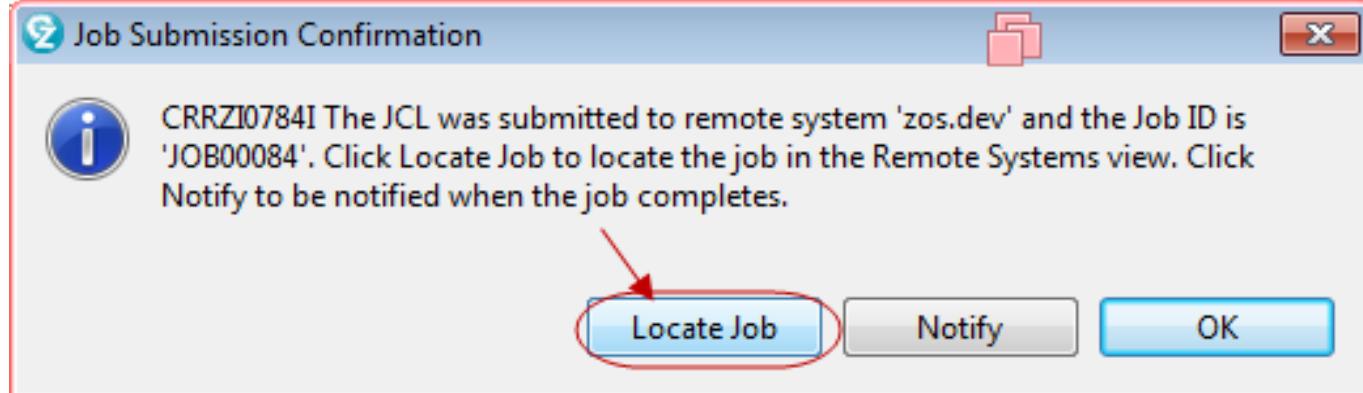
Output window (bottom right):

```
[5/28/19, 10:34 AM] Job JOB00158 is being submitted
[5/28/19, 10:34 AM] Job EMPOT011:JOB00158 ended with completion code 0
```

5.4.4 ► Right click and select **Submit to > zos.dev**



5.4.5 ► Click on **Locate Job**. The job submitted will be shown under JES folder under Remote Systems view



5.4.6 You MUST have the return code 0000 and the bug will be corrected.

► Using *Remote Systems* view, verify the results double expanding **POT01RUN:JOB00xxx** (where **00xxx** can be any number) and clicking on **CURSRAV4:SYSOUT**. Now you **MUST** have 000 as return code
Tip: You may need to refresh Retrieved Jobs to see it go from active to finished

The screenshot shows the Rational Application Developer interface. On the left is a terminal window titled 'pot01run.jcl' with the command 'EMPOT01.POT01RUN.JOB00113.D0000103.?spool'. The output of the job is displayed, including statistics for departments and hours, and a series of messages at the end indicating the result was 33, thanks to a user, and executing various paragraphs. A red box highlights the last six lines of this message block.

The right side of the interface is a 'Remote Systems' view showing a file tree. It includes sections for 'My Favorites', 'Local Shells', and 'zos.dev'. Under 'zos.dev', there are 'z/OS UNIX Files', 'z/OS UNIX Shells', 'MVS Files', 'TSO Commands', and 'JES'. The 'Retrieved Jobs' section contains several entries, each with a circled number above it:

- 1: POT01RUN:JOB00113 [CC 0000] (highlighted with a red circle)
- 2: CURSRAV4::SYSTSIN [0000] (highlighted with a red circle)
- 3: CURSRAV4::SYSOUT [0000] (highlighted with a red circle)
- 4: CURSRAV4::SYSTSPRT [0000]
- 5: POT01CC:JOB00092 [CC 0000]

5.4.7 ►| Use **Ctrl + Shift + F4** to close all editors. Or just click on the of each opened editor

What have you done so far?

You used **Fault Analyzer** to identify why the program was abending.

You used the **Debugger** to temporarily fix the error caused by the called program REGI0B.

You used IDz to change the program REGI0B to return other value than zero.

You compiled and linked REGI0B creating another version on your load library.

You executed again the JOB but now using the REGI0B that you created and verified that the abend is gone.



Section 6. Using the Code coverage

Code coverage analyzes a running program and generates a report of statements that were executed, compared to the total number of executable lines.

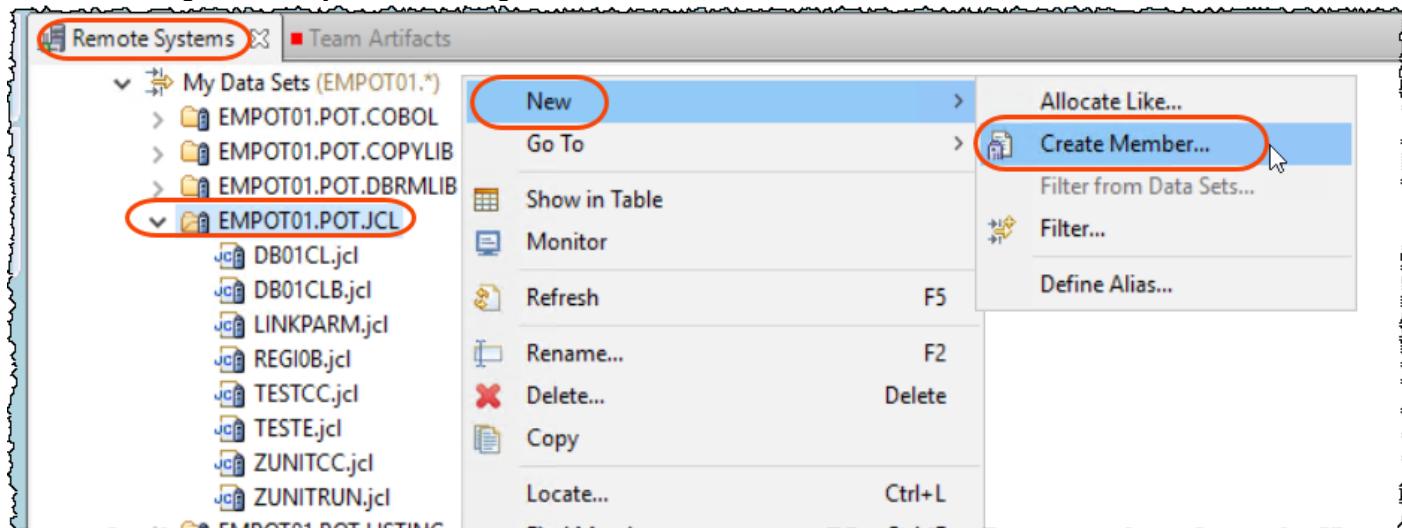
Developer for z Systems Host Utilities provides two ways to invoke Code coverage in batch mode, A sample JCL procedure, to process a single program run, and a set of scripts to start and stop a permanently active Code coverage collector that can process multiple program runs.

You can run code coverage for any application you can debug. You can generate code coverage reports that you can view in the workbench or save the results to your file system for future analysis. We will show a simple batch mode invocation.

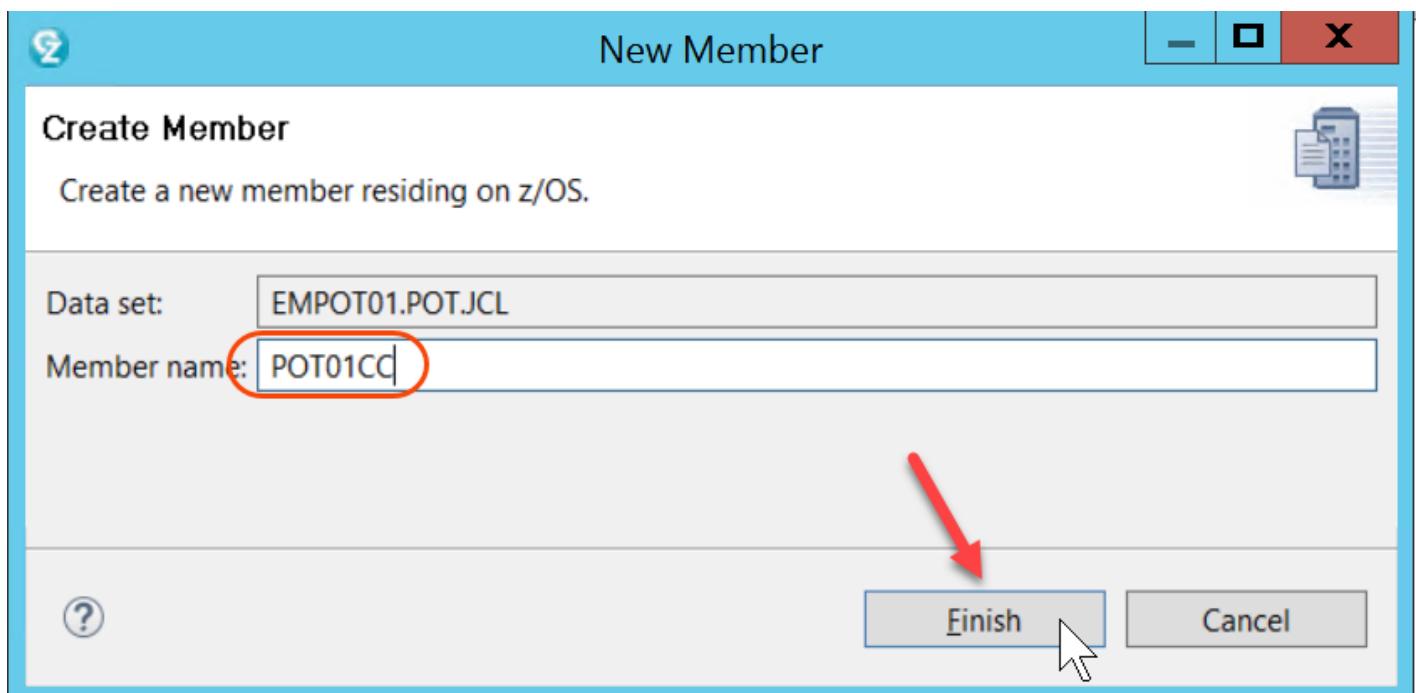
6.1 Creating a JCL to run the code coverage

You will create a JCL file to run Code Coverage

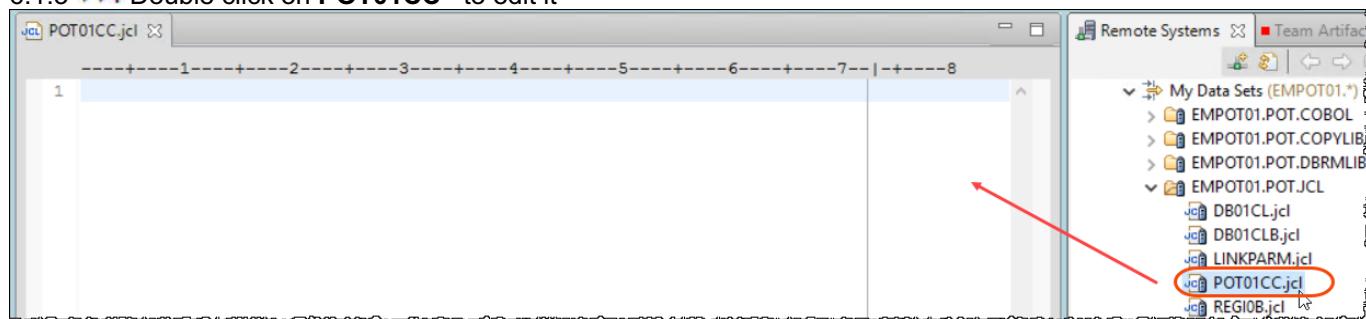
6.1.1 ►| Using Remote Systems view, right click on **EMPOT01.POT.JCL** and select **New > Create Member...**



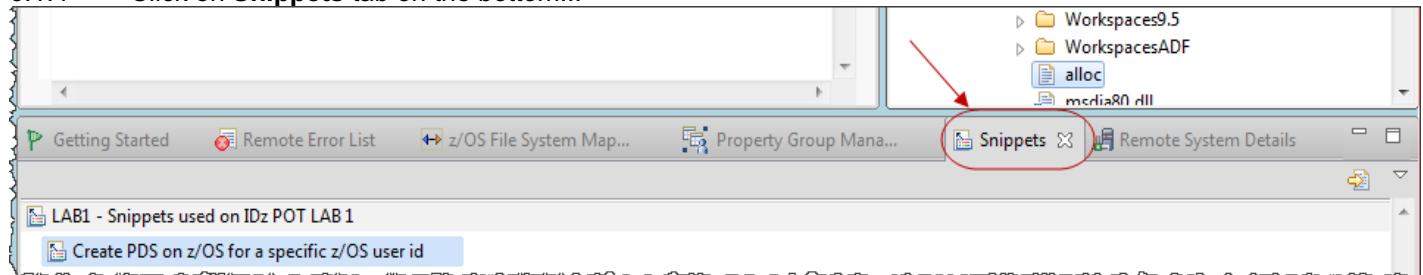
6.1.2 ►| Name it as **POT01CC** and click **Finish**



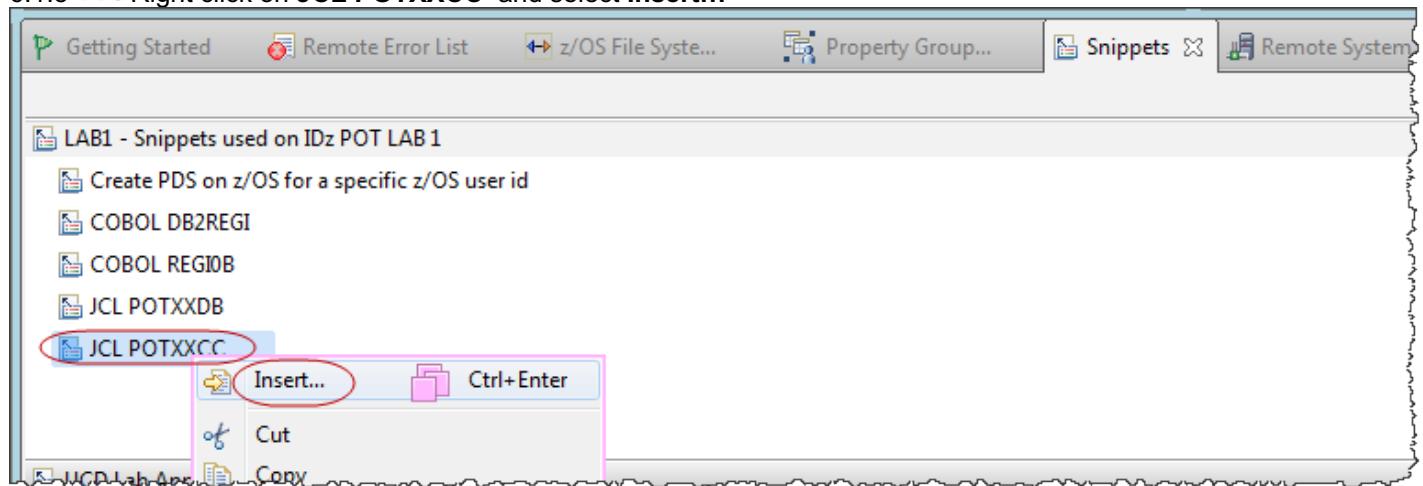
6.1.3 ► Double click on **POT01CC** to edit it



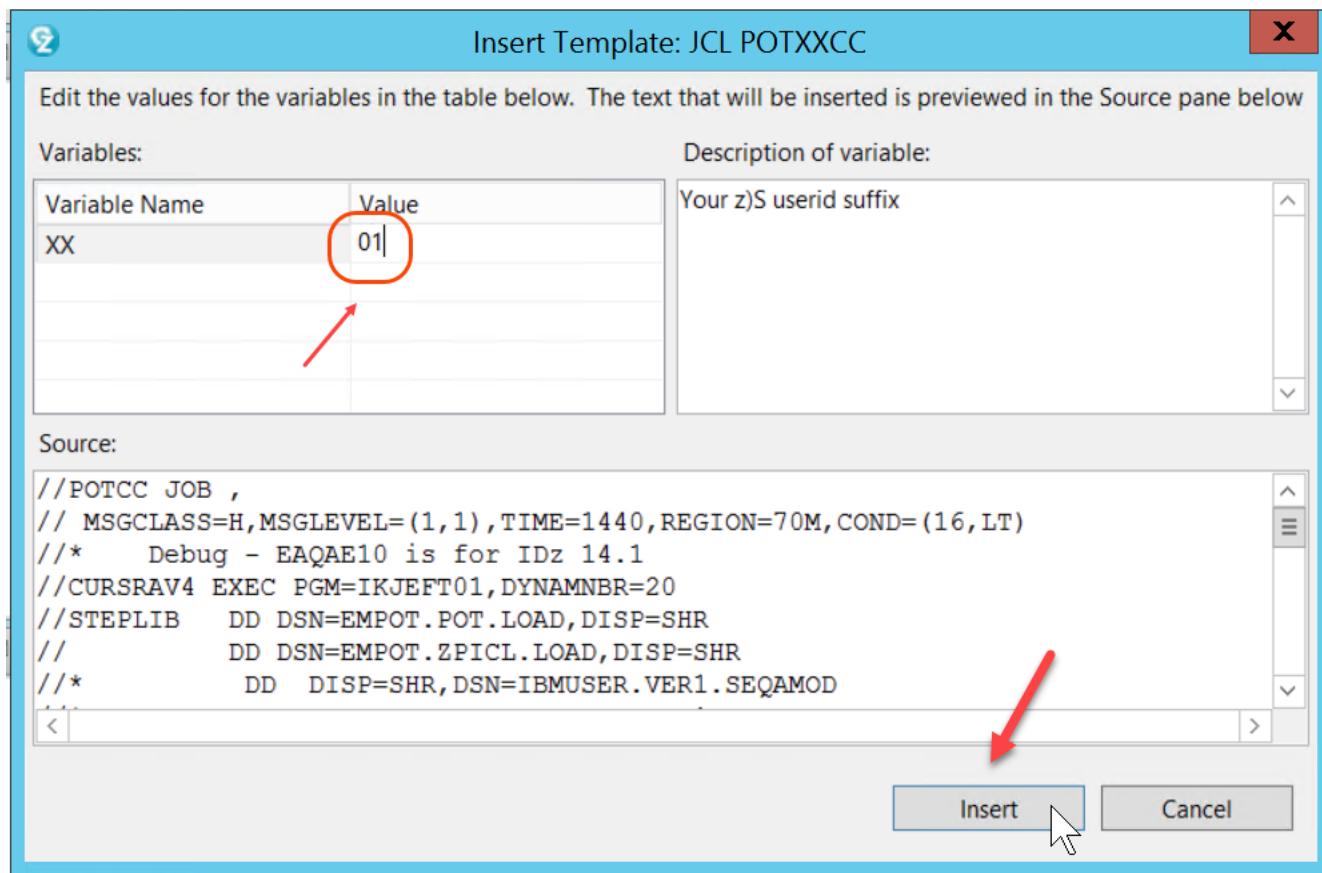
6.1.4 ► Click on **Snippets** tab on the bottom...



6.1.5 ► Right click on **JCL POTXXCC** and select **Insert...**



6.1.6 ► When the *Insert* dialog opens, type **01** in the *Value* and click **Insert**



6.1.7 . Notice that the Snippets variables will be replaced.

►| Use **Ctrl + S** to save the changes.

```

JCL *POT01CC.jcl

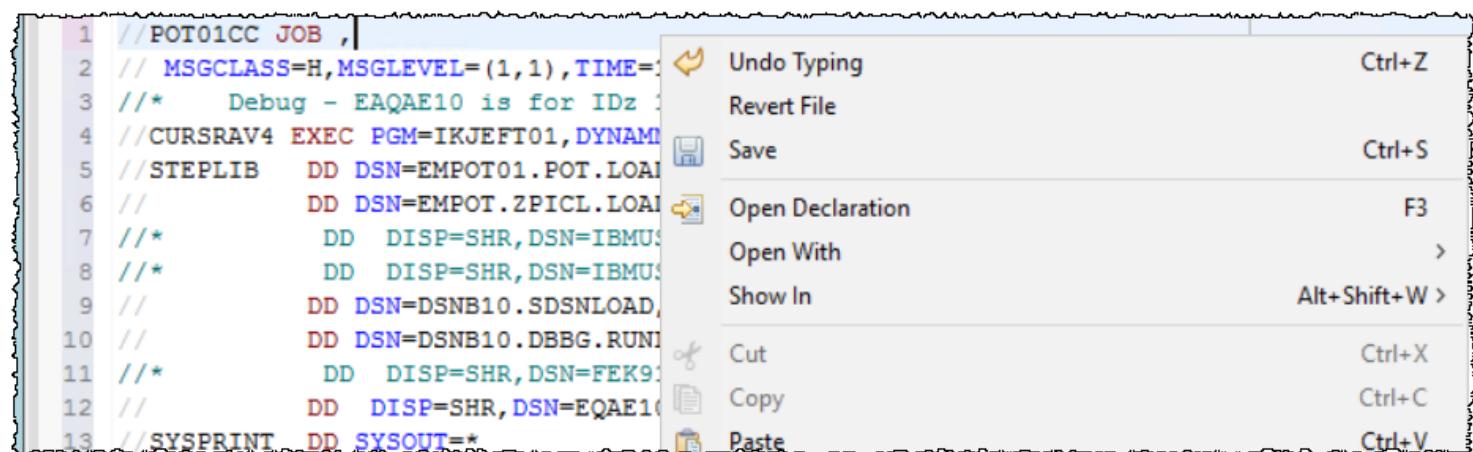
1 //POT01CC JOB ,
2 // MSGCLASS=H,MSGLEVEL=(1,1),TIME=1440,REGION=70M,COND=(16,LT)
3 /* Debug - EAQAE10 is for IDz 14.1
4 //CURSRAV4 EXEC PGM=IKJEFT01,DYNAMNBR=20
5 //STEPLIB DD DSN=EMPOT01.POT.LOAD,DISP=SHR
6 // DD DSN=EMPOT.ZPICL.LOAD,DISP=SHR
7 /* DD DISP=SHR,DSN=IBMUSER.VER1.SEQAMOD
8 /* DD DISP=SHR,DSN=IBMUSER.VER1.SEQAAUTH
9 //
10 // DD DSN=DSNB10.SDSNLOAD,DISP=SHR
11 // DD DISP=SHR,DSN=DSNB10.DBDBG.RUNLIB.LOAD,DISP=SHR
12 // DD DISP=SHR,DSN=FEK910.SFEKAUTH
13 //SYSPRINT DD SYSOUT=*
14 //SYSOUT DD SYSOUT=*
15 //SYSTSPRT DD SYSOUT=*
16 //SYSTSIN DD *
17 TSOLIB ACTIVATE DA('DSNB10.SDSNLOAD')
18 DSN SYSTEM(DBDBG)
19 RUN PROGRAM(DB2REGI) PLAN(DB2REGI) -

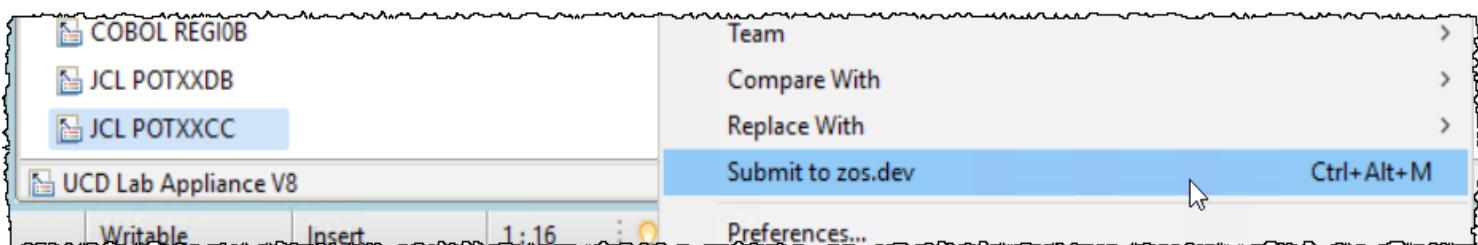
```

6.2 Submit the JCL for z/OS execution

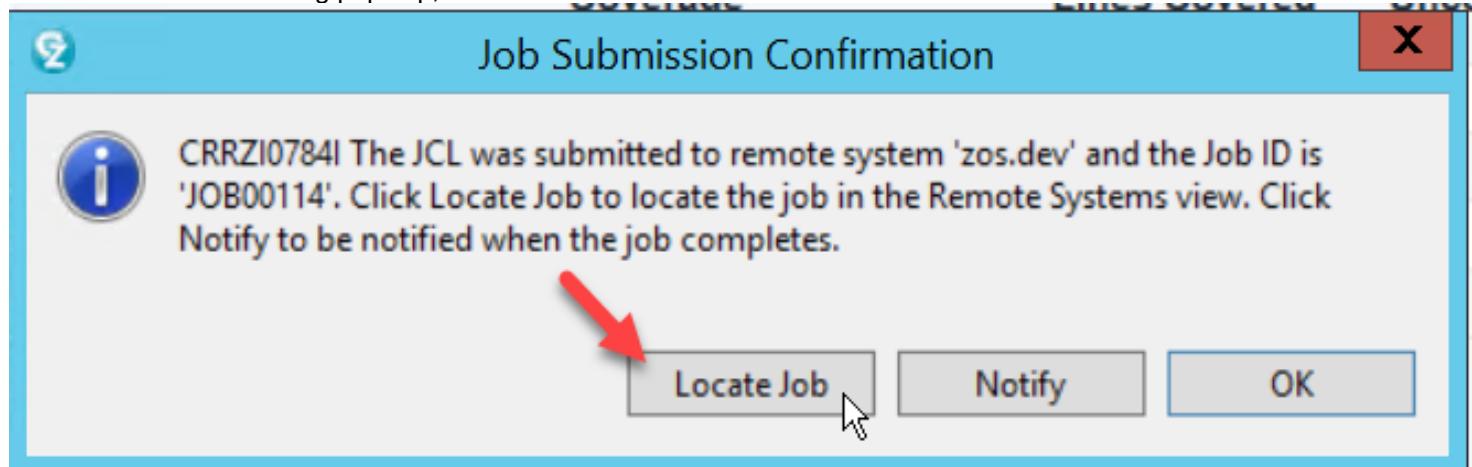
You will now execute the fixed batch COBOL/DB2 on z/OS

6.2.1 ►| Right click on the JCL being edited and select **Submit to zos.dev**





6.2.2 ► When the dialog pops up, select **Locate Job**



6.2.3 The code coverage report will be created. You can see that this test covered only **73%** of your program

► Click on **Code Coverage Results** tab

Code Coverage Report

Code coverage report for 'DB2REGI_2021_04_16_131035_0684', analyzed Apr 16, 2021 1:10:35 PM

Export

Off On Show below : 80 % Refresh

Files Modules

Name	Coverage	Lines Covered	Uncovered Lines	Total
DB2REGI.DB2REGI.cob	69%	61	27	
DB2REGI.REGI0C.cob	100%	9	0	
REGI0B.REGI0B.cob	100%	3	0	
Summary (Elapsed time: 0.718 sec)	73%	73	27	

Remote Error List z/OS File System Mapping Property Group Manager Snippets Remote System Details **Code Coverage Results**

Name Status Coverage Level Analyzed Date Additional Info

Compiled Code Coverage Workspace Results DB2REGI_2021_04_16_131035_0684 73% Line Apr 16, 2021 1:10:35 PM CRRDG7202

Java Code Coverage Workspace Results

6.2.4 ► Click on **DB2REGI.DB2REGI.cob**

Name	Coverage	Lines Covered	Uncovered Lines	Total
DB2REGI.DB2REGI.cob	69%	61	27	

6.2.5 ► Scroll down move the mouse to the colored green and red bars and you will see the lines executed in **green** and the lines not executed in **Red**.

```

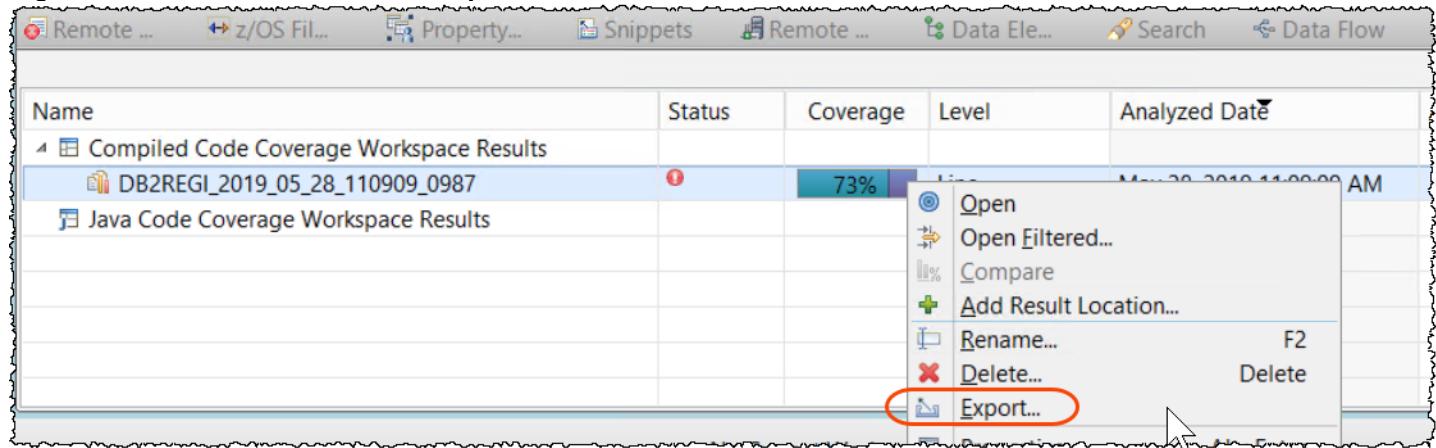
JCL POT02CC.jcl DB2REGI_2021_04_16_131035_0684 CBL DB2REGI.DB2REGI.cob
-----+-----+-----+-----+
332     EXIT.
333     350-TERM-RTN.
334     MOVE ROW-KTR TO ROW-STAT.
335     DISPLAY ROW-MSG.
336     350-EXIT.
337     * EXIT.
338     GO TO 500-SECOND-PART.
339     999-ERROR-TRAP-RTN.
340     **** ERROR TRAPPING ROUTINE FOR NEGATIVE SQLCODES ****
341     ****
342     ****
343 Lines 343-351 not covered. **** WE HAVE A SERIOUS PROBLEM HERE ****.
344     999-ERROR-TRAP-RTN'.
345     MULTIPLY SQLCODE BY -1 GIVING SQLCODE.
346     DISPLAY 'SQLCODE ==> ' SQLCODE.
347     DISPLAY SQLCA.
348     DISPLAY SQLERRM.
349     EXEC SQL WHENEVER SQLERROR CONTINUE END-EXEC.
350     EXEC SQL ROLLBACK WORK END-EXEC.
351     *
352     * -----
353     500-SECOND-PART.
354     MOVE 2 TO BRANCHFLAG.
355     MOVE 'AAAAAAA' to FIELD-A.
356     MOVE 'BBBBBBB' to FIELD-B.
357     MOVE 'CCCCCCC' to FIELD-C.

```

As you can see the error routine on the picture above is not tested..
Imagine how bad if you go to production and never test this condition

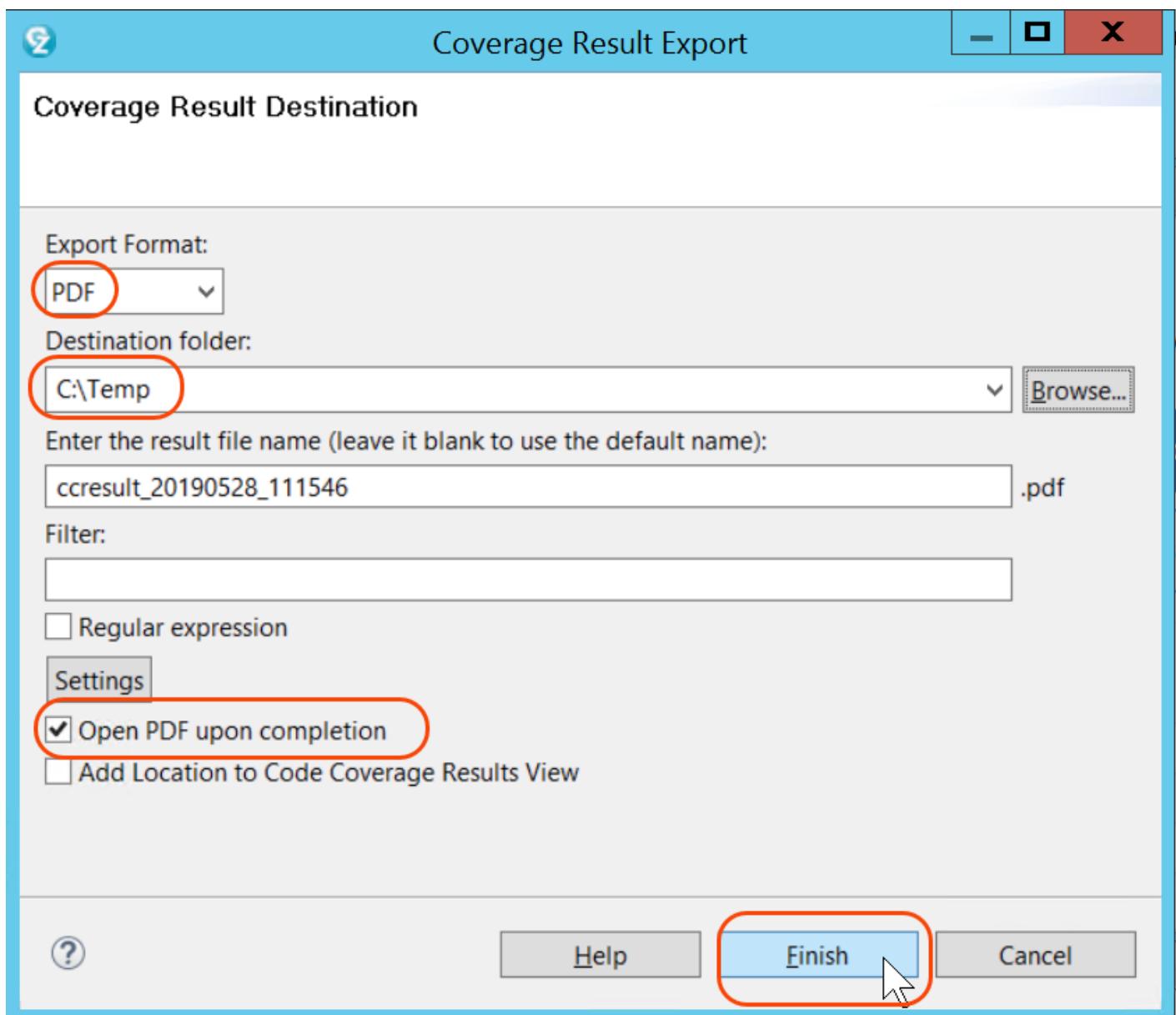
6.2.6 ► You could also create a PDF report as documentation or export this data to other products (like ADDI or SonarQube)

Right click on the results and select **Export ..**



6.2.7 ► Select **PDF**, **C:\Temp**, **Open PDF upon completion** and **Finish**.

Note that you can export to **Sonar Qube** if you have this software.



6.2.8 A PDF report will be generated.

File Overall Summary

Report Info

Report: ccresult_20190528_111546.pdf

Type: LINE

Generation Date: May 28, 2019 11:20:54 AM

Filter String:

Overall Summary

Total Number of Files: 3

Total Number of Functions: 24

Total Number of Lines: 100

Total Number of Statements: 100

Code Coverage Summary

File Coverage: 100%

Function Coverage: 88%

Line Coverage: 73%

Statement Coverage: 73%

6.2.8 ► Close all editors pressing **Ctrl + Shift + F4** Or just click on the  of each opened editor.

Section 7. (Optional) Executing SQL statements when editing the program.

IDz can be installed with Data studio that is very helpful when working with a Database.

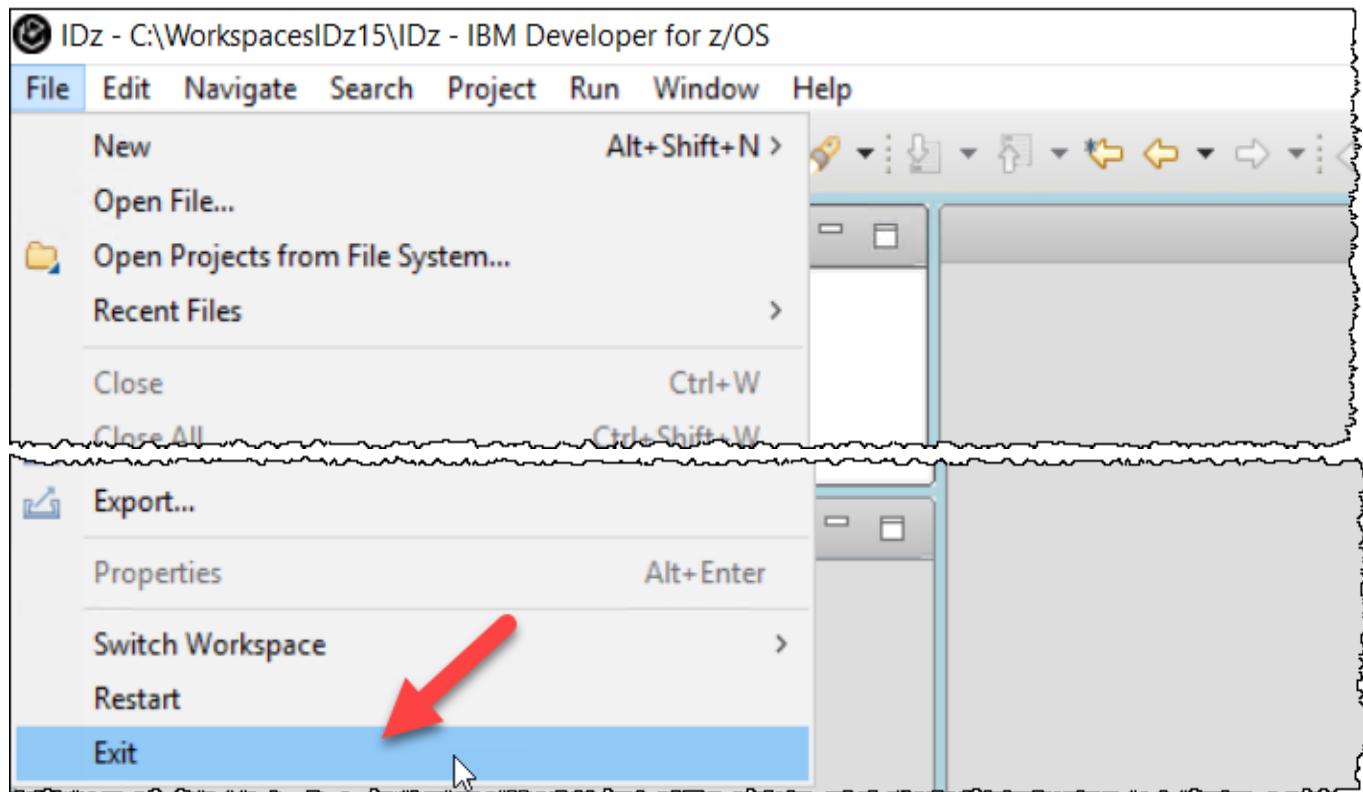
Data Studio is a foundational offering that includes support for key tasks across the data management lifecycle, including administration, application development, and query tuning.

7.0 Starting IDz version 14 and connect to z/OS

When we created this lab Data studio could be installed only on IDz V 14. So for this section you will use IDz Version 14 instead of IDz version 15. Notice that Data Studio is a free plugin.

7.0.1 Before starting IDz version 14, you must close IDz version 15 to save some memory on your windows client.

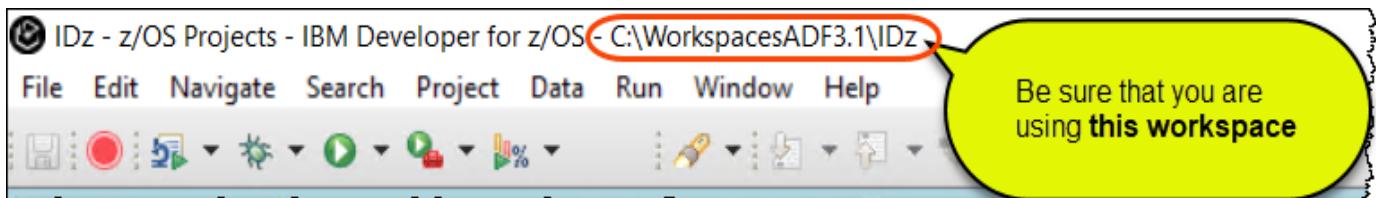
► Click on File > Exit



7.0.2 Start **IBM Developer for z Systems version 14**

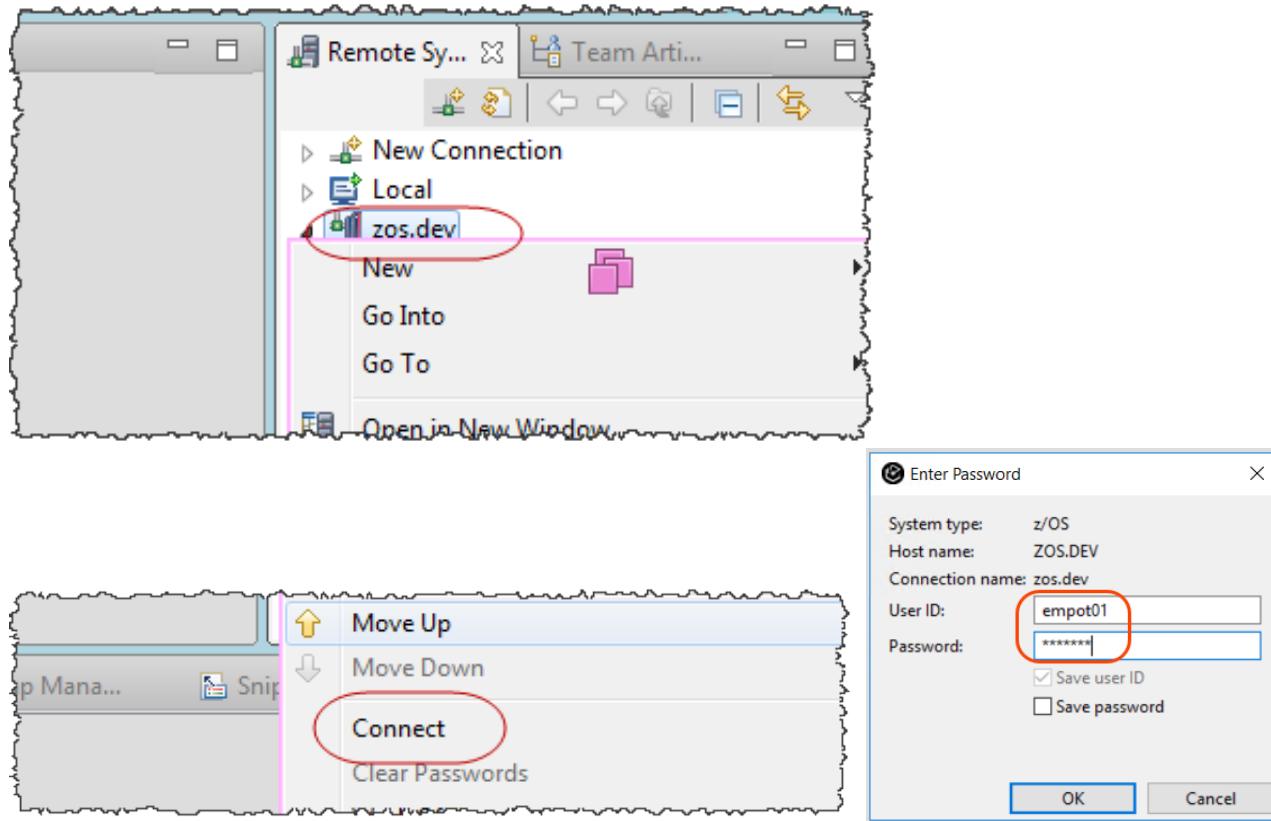
► Using the windows desktop double click on **IDz 14.2.4** icon.

IMPORTANT -> This icon will start an eclipse workspace that has already some definitions required for this lab.
PLEASE DO NOT start IDz using other way than this icon.



7.0.3 To Connect to the z/OS system:

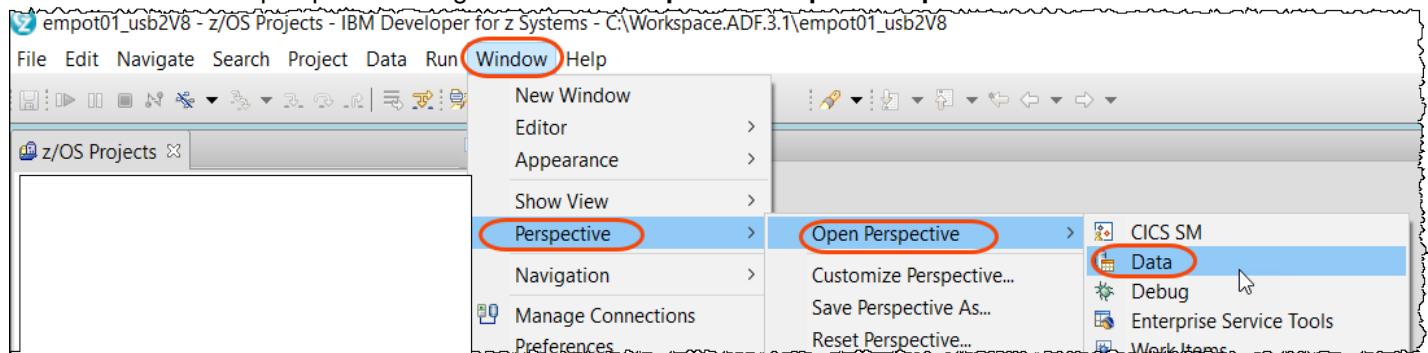
- ▶ Using the **Remote Systems** view, right-click on **zos.dev** and select **Connect**.
- ▶ Use **empot01** and password **empot01**



7.1 Creating a connection to the DB2 on z/OS

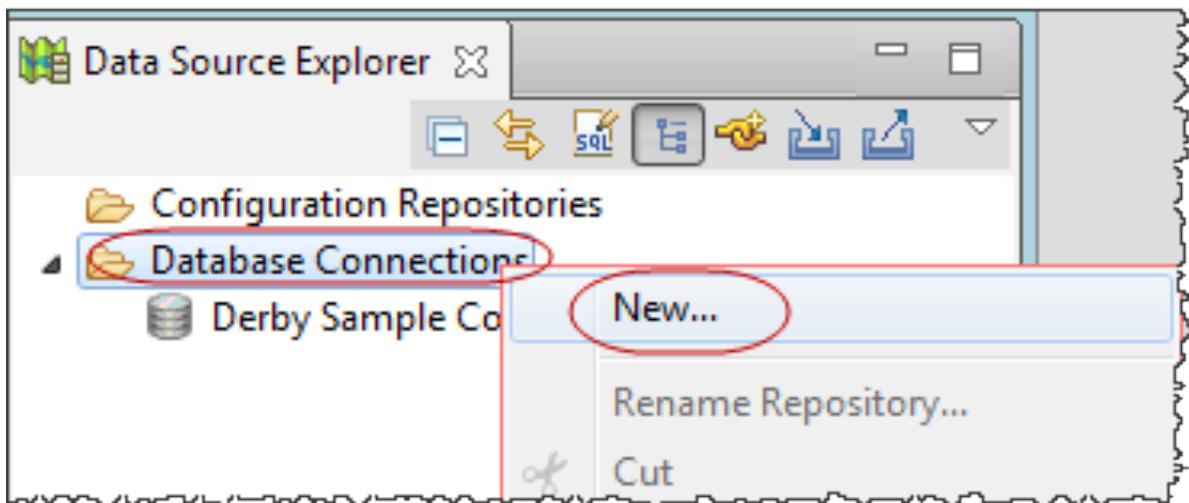
You will need to have authorization to connect to the DB2. Now you will use **IBMUSER** as a new z/OS ID

7.1.1 Go to Data perspective using **Window > Perspective > Open Perspective > Data**

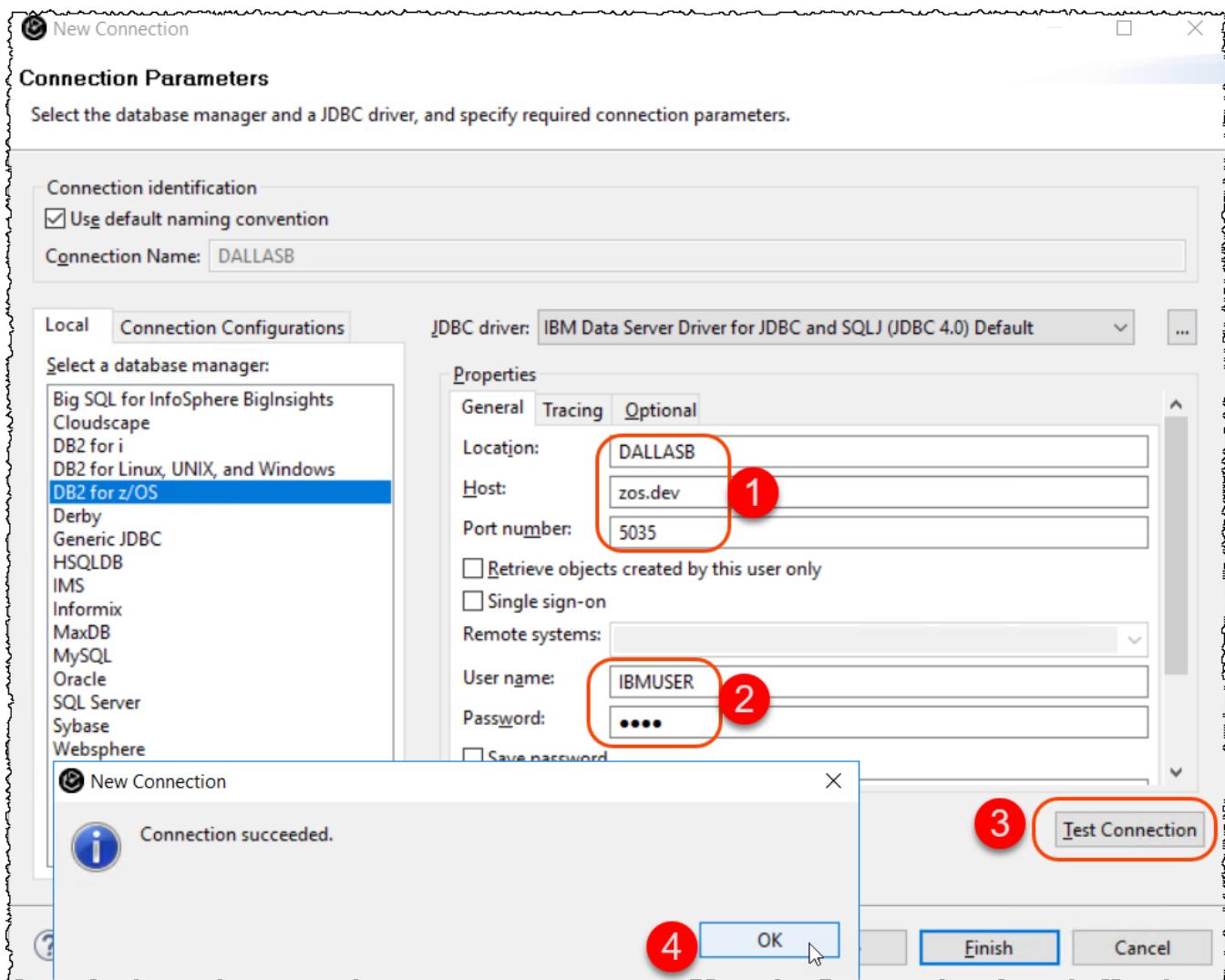


7.1.2 Using the **Data Source Explorer** view, create a new z/OS DB2 connection:

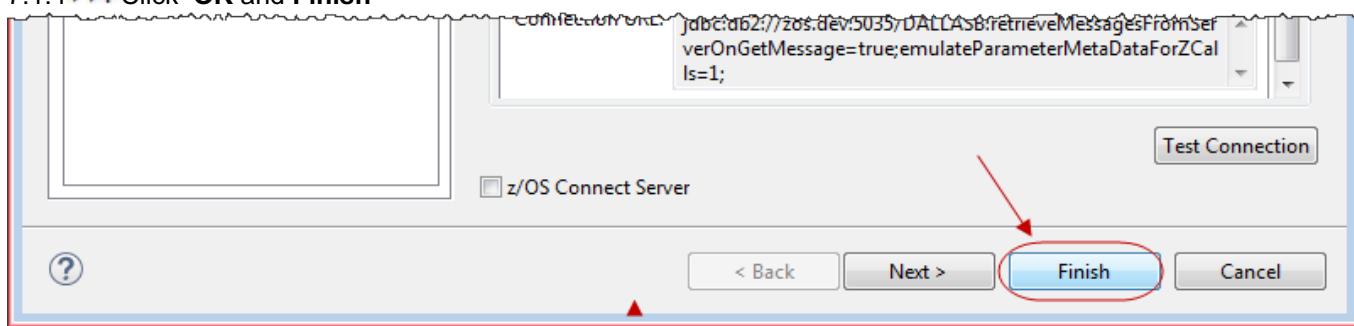
- ▶ Right click on **Database Connections** and select **New..**



7.1.3 ► Select **DB2 for z/OS**, use *Location* as **DALLASB**,
Host is **zos.dev** and port is **5035** , **IBMUSER** and password **SYS1**
Click **Test Connection** to be sure you can successfully connect to the DB2.

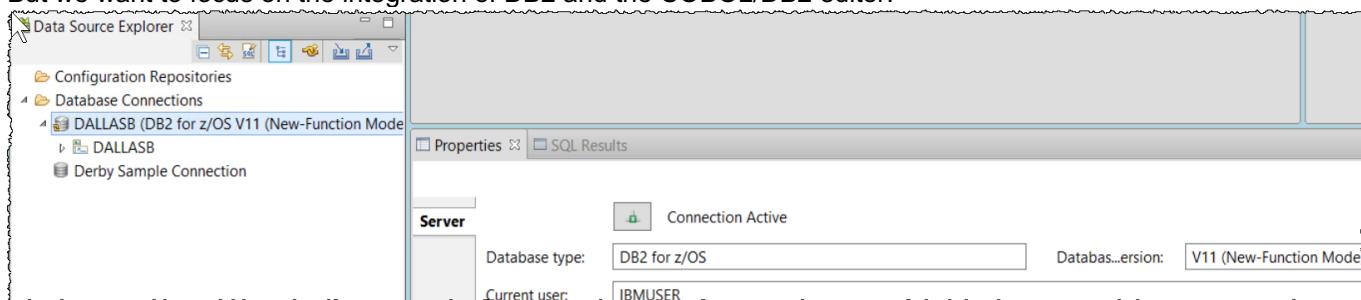


7.1.4 ► Click OK and Finish



The connection is created. You could see tables, modify contents etc..

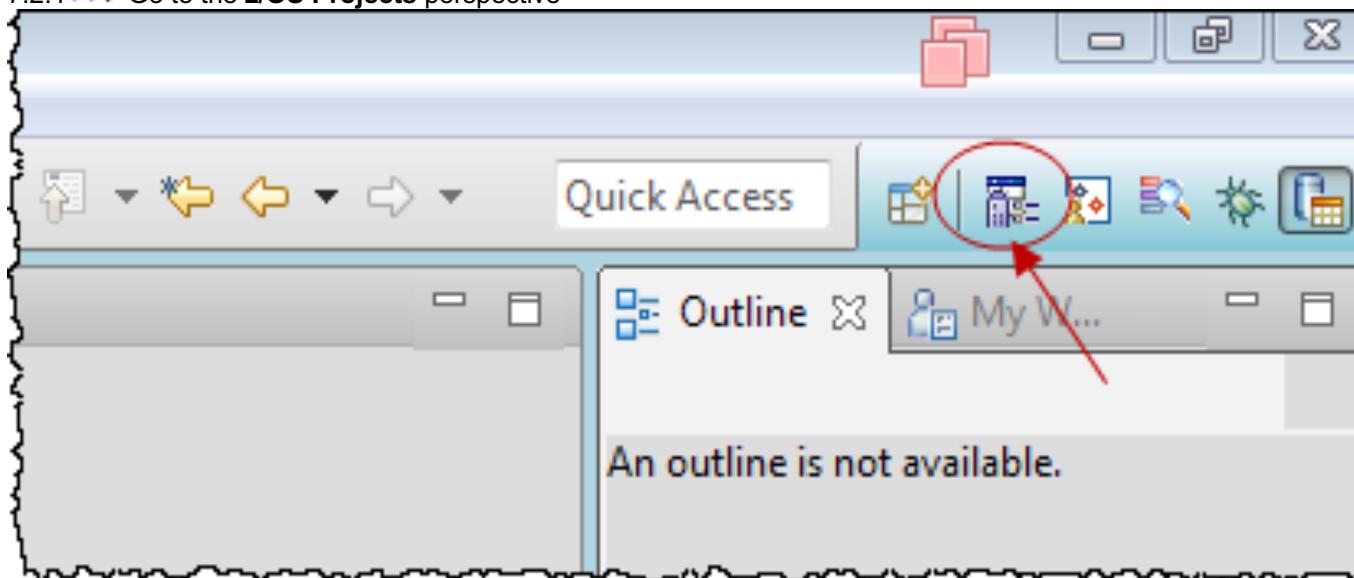
But we want to focus on the integration of DB2 and the COBOL/DB2 editor.



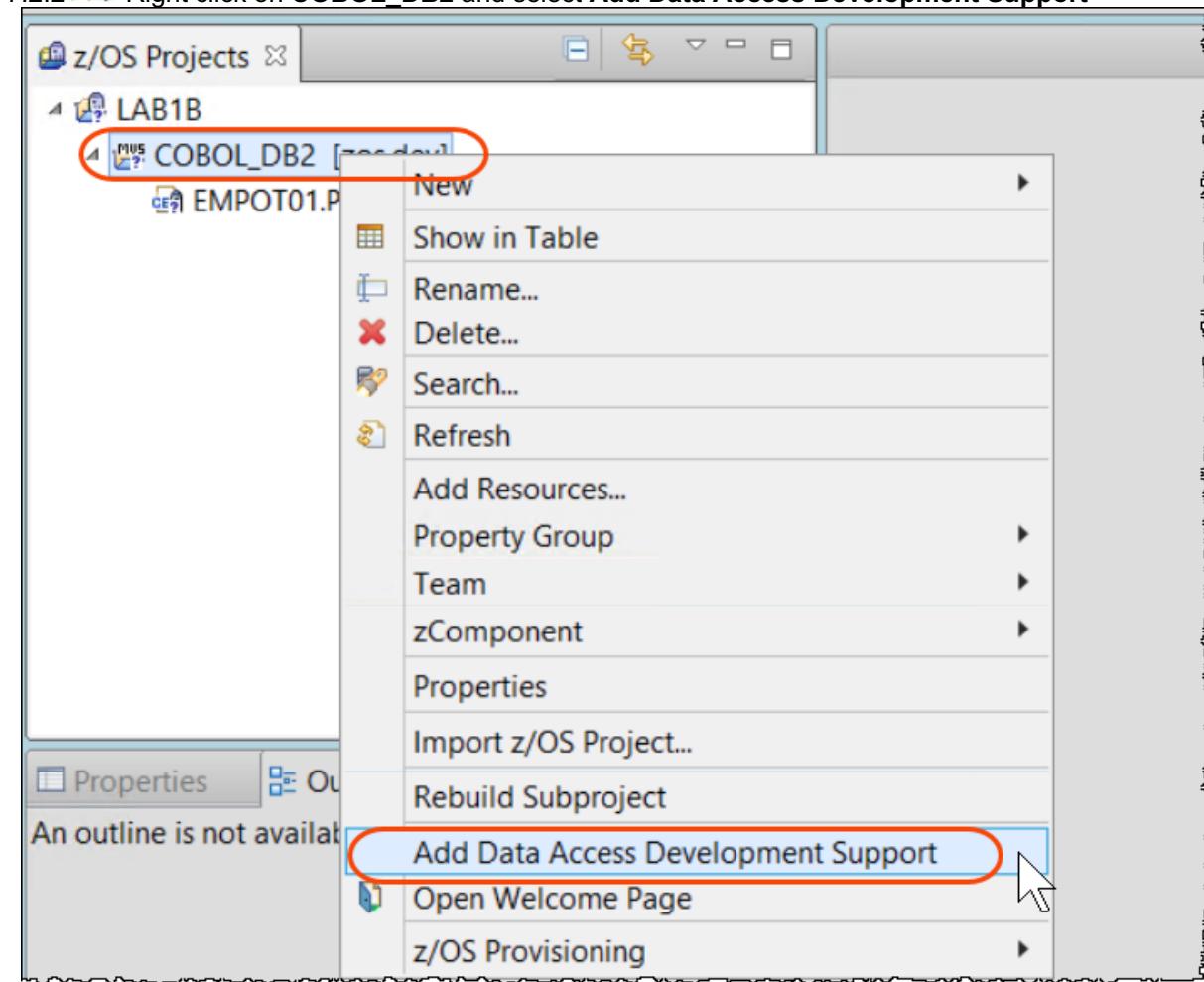
7.2 Running a SQL query from COBOL program

You will need to have authorization to run the queries.

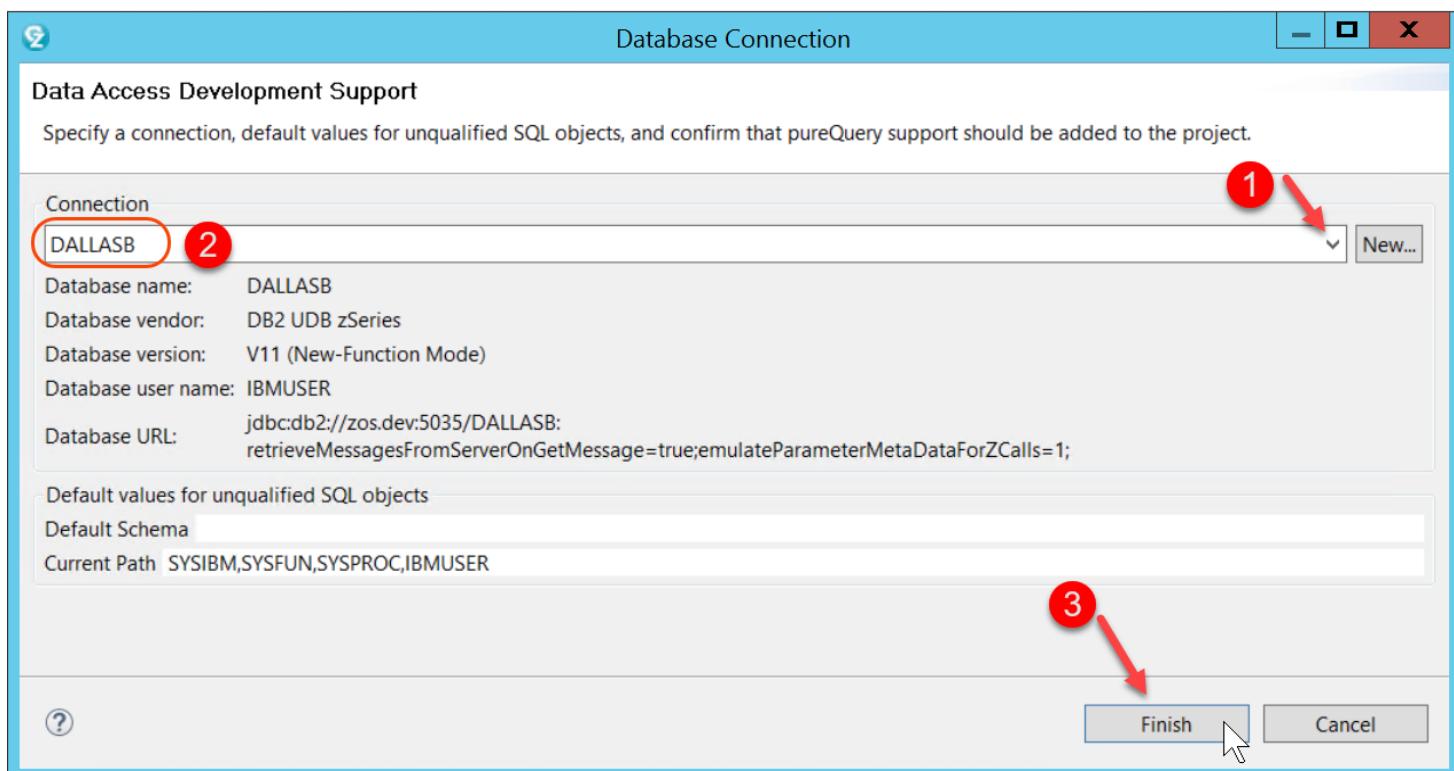
7.2.1 ► Go to the z/OS Projects perspective



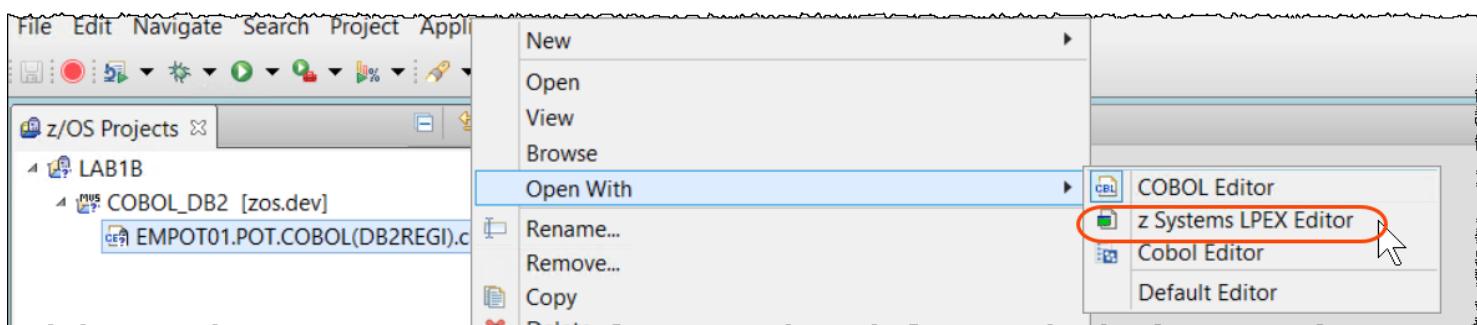
7.2.2 ► Right click on **COBOL_DB2** and select **Add Data Access Development Support**



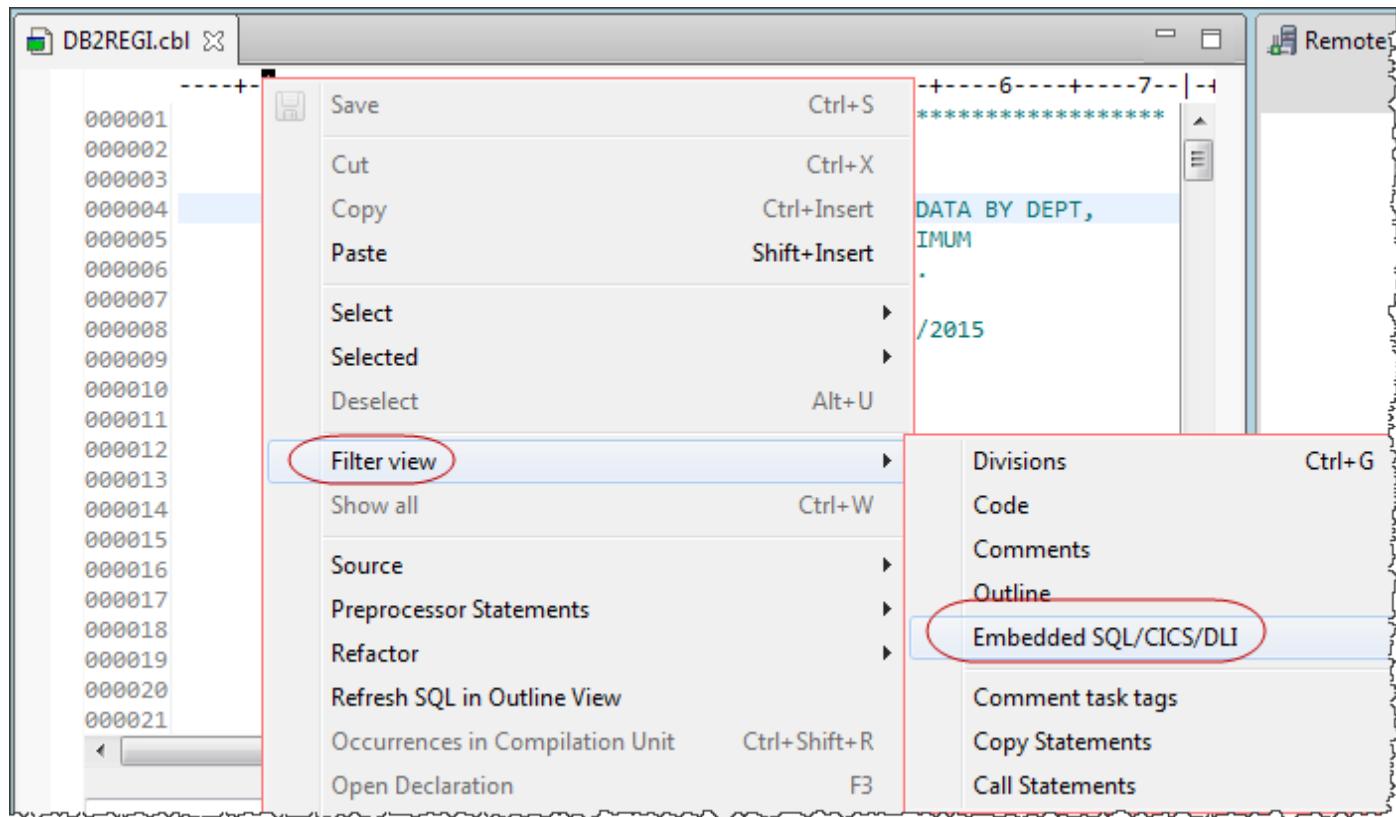
7.2.3 ► Select the connection created **DALLASB** and click **Finish**



7.2.4 ► Right click on program **DB2REGI.cbl** and select **Open With > z Systems LPEX editor**



7.2.5 ► Right click and select **Filter View > Embedded SQL/CICS/DLI**



7.2.6 ► Highlight the SQL statements as shown **right click** and choose **Run SQL**

The screenshot shows a code editor window titled "DB2REGL.cbl". The code is a COBOL program with several EXEC SQL statements. A portion of the SQL code is selected (highlighted with a blue background). A context menu is open on the selected text, and the "Run SQL" option is highlighted with a red circle. The menu also includes other options like Paste, Select, Selected, Deselect, Filter view, Show all, Source, Preprocessor Statements, Refactor, Tune SQL, Refresh SQL in Outline View, Occurrences in Compilation Unit, Open Declaration, Open Perform Hierarchy, and View.

```

-----+--*A-1-B---+----2----+---3---+----4---
+ 000030      EXEC SQL INCLUDE SQLCA END-EXEC
 000068      EXEC SQL INCLUDE DIAGCODE END-EXEC
 000069      EXEC SQL INCLUDE EMPL END-EXEC.
+ 000070      EXEC SQL INCLUDE CUST1 END-EXEC.
 000121      EXEC SQL
 000122      DECLARE C1 CURSOR FOR
 000123      SELECT DEPT, MIN(PERF), MAX(
 000124      MIN(HOURS), MAX(HOURS), AVG(
 000125      FROM RBAROSA.EMPL E, RBAROS
 000126      WHERE E.NBR = P.NBR
 000127      * AND PERF > :PERF
 000128      GROUP BY DEPT
 000129      END-EXEC.
 000135      EXEC SQL OPEN C1
+ 000136      END-EXEC.
 000187      EXEC SQL FETCH C1 INTO
 000188      :DEPT,:TRI,:DEPT,NULL

```

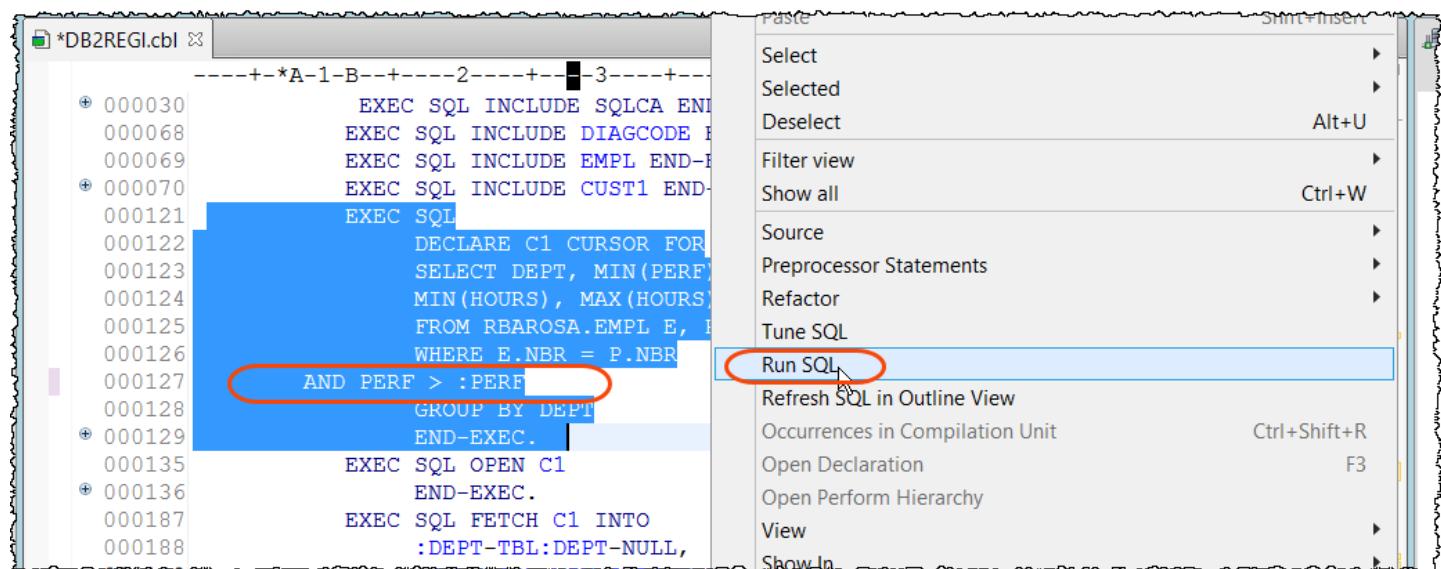
7.2.7 ► Click as below and you will have the query results

The screenshot shows the Rational Application Developer environment. On the left, a code editor window titled "DB2REGI.cbl" displays a COBOL program. The program includes several EXEC SQL INCLUDE statements and a complex SELECT statement that joins tables "RBAROSA.EMPL E" and "RBAROSA.PAY P". On the right, a navigation tree lists various system components like TSO, JES, CICS, DB2, IMS, and Mes. Below the code editor is a toolbar with icons for Remote, z/OS File, Property, Snippets, Remote, Debug, Program, Data Ele..., and Search. A red circle labeled "1" highlights the status bar entry "Succe SELECT D... 8/12/19, ... DALLASB". Another red circle labeled "2" highlights the "Result1" tab of a results table. The results table has columns DEPT, 2, 3, 4, 5, 6, and 7, and contains the following data:

	DEPT	2	3	4	5	6	7
1	ACC	8	8	8	15.99	15.99	15.9900...
2	FIN	3	9	5	8.89	32.45	22.6966...
3	MKT	1	3	1	13.23	32.41	22.8200...
4	R&D	1	1	1	NULL	NULL	NULL
5	REG	9	9	9	26.75	26.75	26.7500...
6	NULL	0	0	0	35.45	35.45	35.4500...

7.2.8 Modifying the SQL query..

► Remove the COBOL comment (*) from the line 127,
Select the SQL statements again right click and choose Run SQL

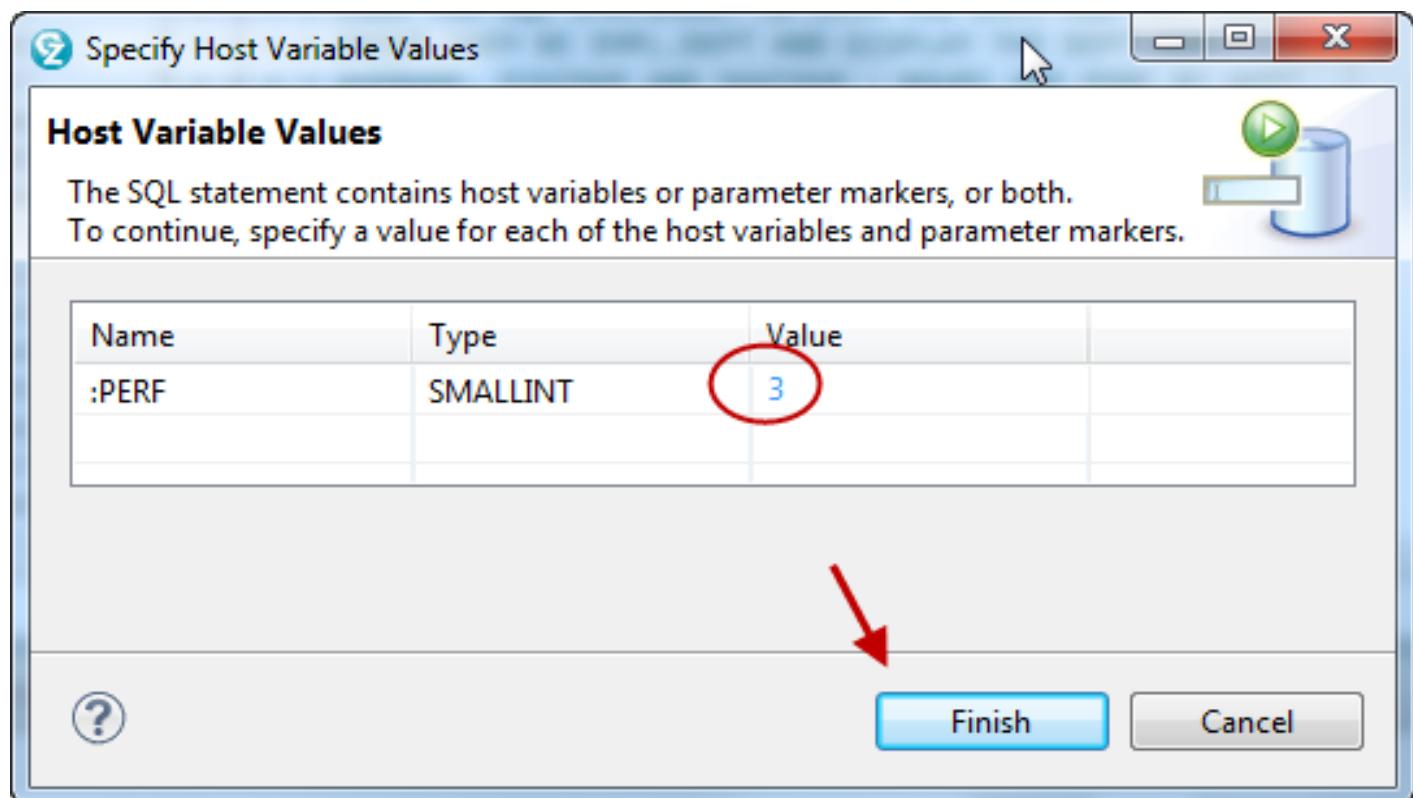


```

-----+--A-1-B---+--2---+---3---+
+ 000030      EXEC SQL INCLUDE SQLCA END-EXEC.
000068      EXEC SQL INCLUDE DIAGCODE END-EXEC.
000069      EXEC SQL INCLUDE EMPL END-EXEC.
+ 000070      EXEC SQL INCLUDE CUST1 END-EXEC.
000121      EXEC SQL
000122      DECLARE C1 CURSOR FOR
000123      SELECT DEPT, MIN(PERF)
000124      MIN(HOURS), MAX(HOURS)
000125      FROM RBAROSA.EMPL E, RBAROSA.CUST1 P
000126      WHERE E.NBR = P.NBR
000127      AND PERF > :PERF
000128      GROUP BY DEPT
000129      END-EXEC.
+ 000130      EXEC SQL OPEN C1
000131      END-EXEC.
000135      EXEC SQL FETCH C1 INTO
000136      :DEPT-TBL:DEPT-NULL,
000187
000188

```

7.2.9 ► Specify a value like 3 and click Finish



7.2.10 The new results now will be:

Type query expression here

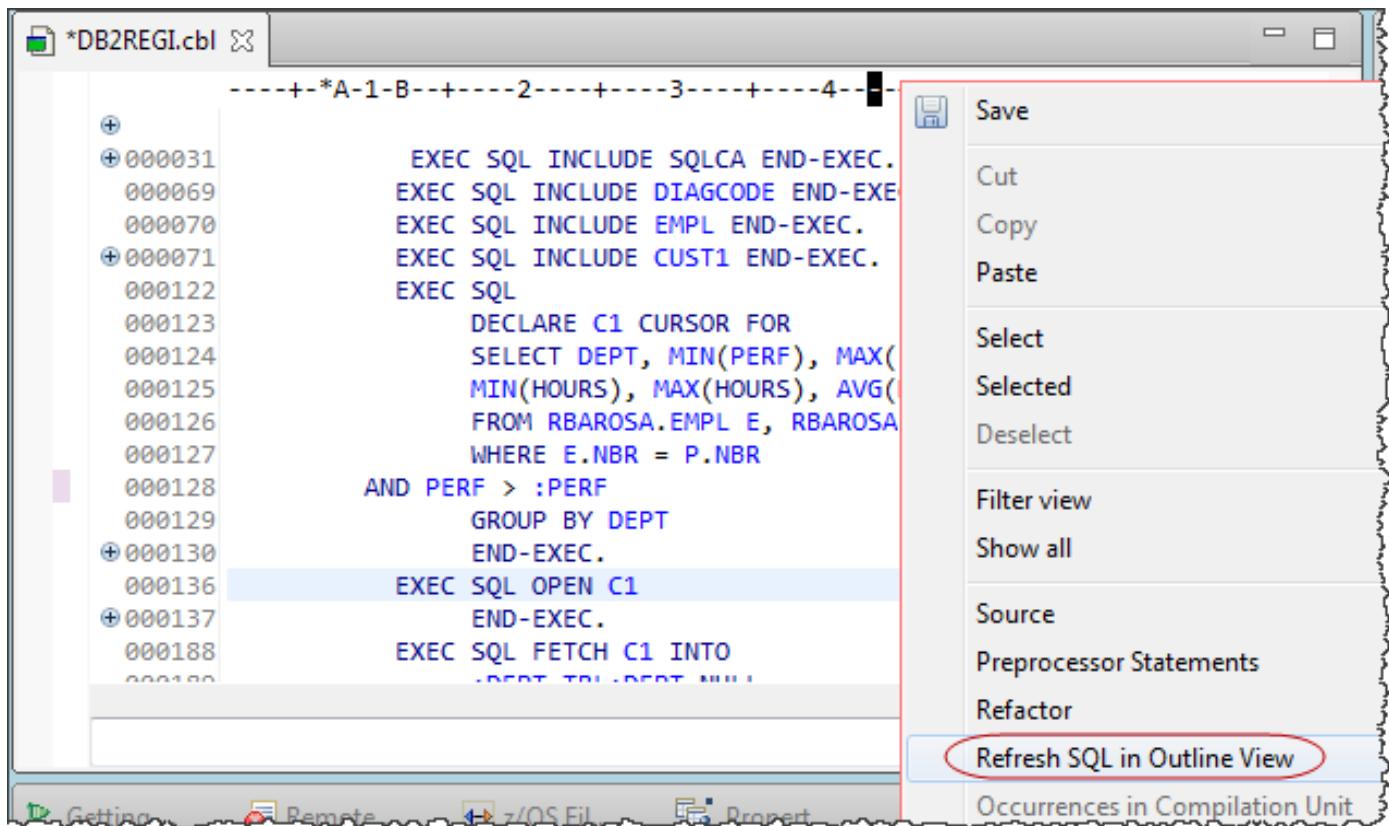
Status	Operation	Date	Connectio...
✓ Success	SELECT DEP...	1/6/17, 11:2...	DALLASB
✓ Success	SELECT DEP...	1/6/17, 11:2...	DALLASE

Status Result1 2

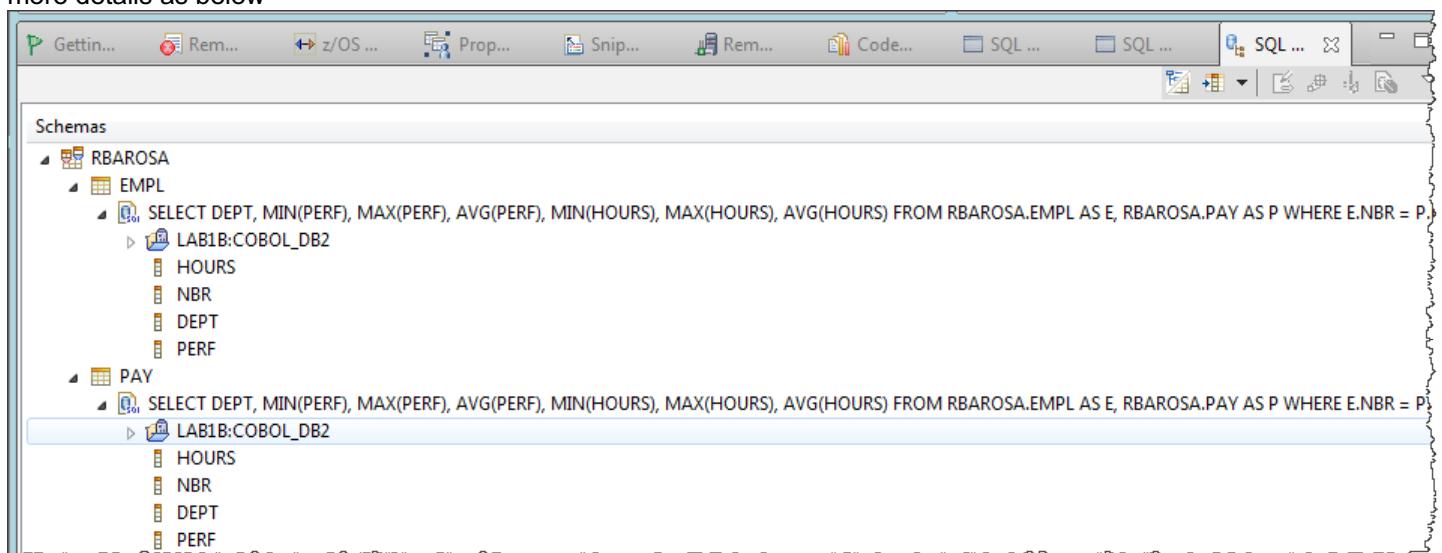
	DEPT	2	3	4	5	6	7
1	ACC	8	8	8	15.99	15.99	15.99
2	FIN	4	9	6	8.89	32.45	20.67
3	REG	9	9	9	26.75	26.75	26.75

7.3 Using the SQL Outline view

7.3.1 ► Without selecting any statement, right click and select Refresh SQL in Outline View

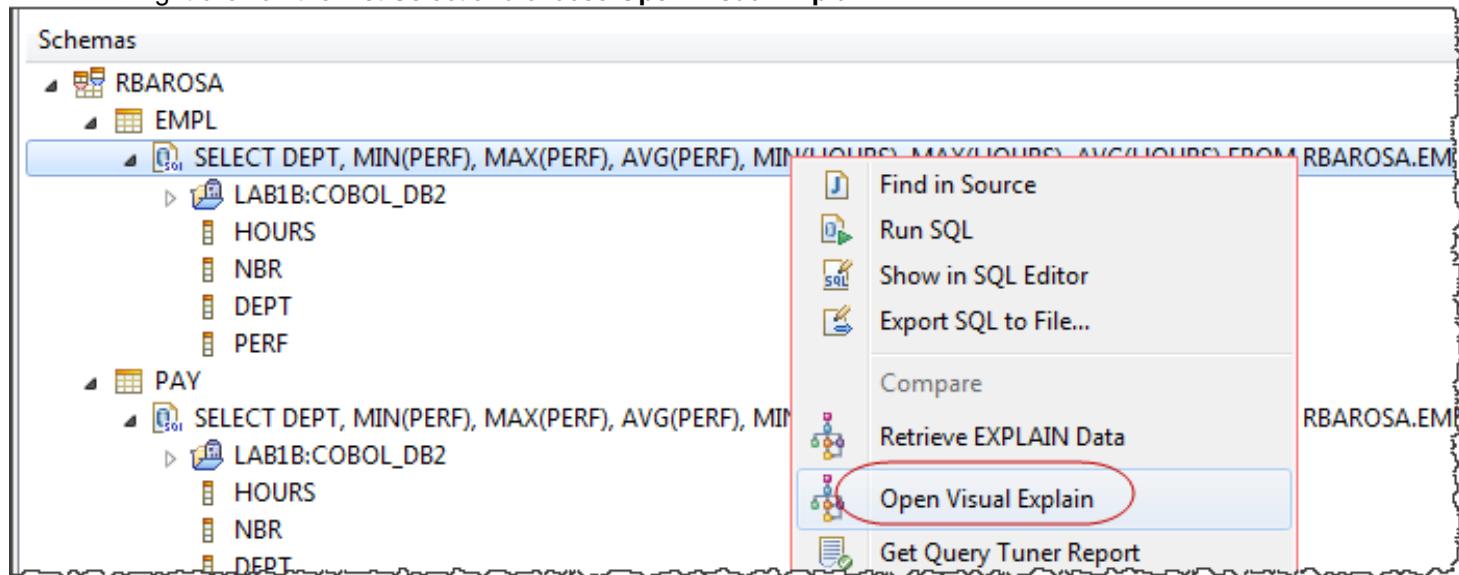


7.3.2 ► Using SQL Outline View, expand RBAROSA, EMPL, PAY and the SQL statements and you will have more details as below

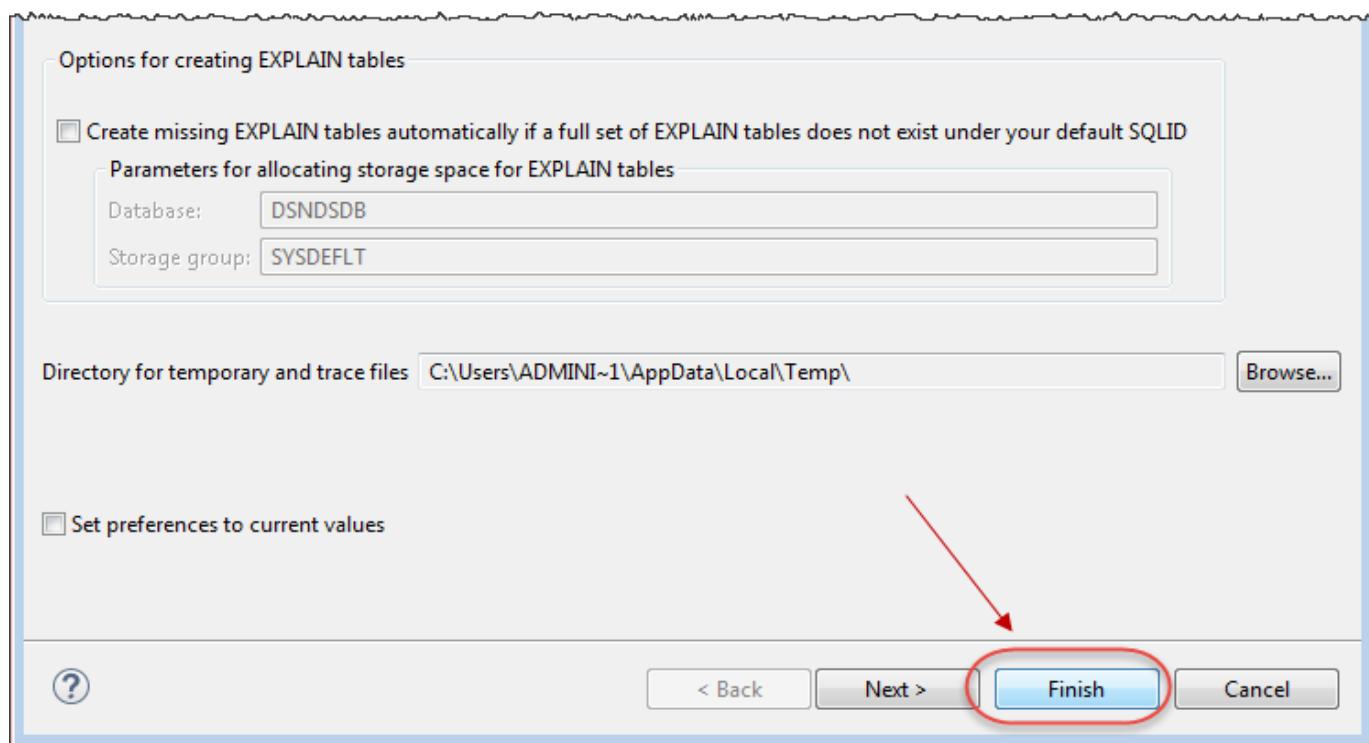


7.4 Using SQL Visual Explain

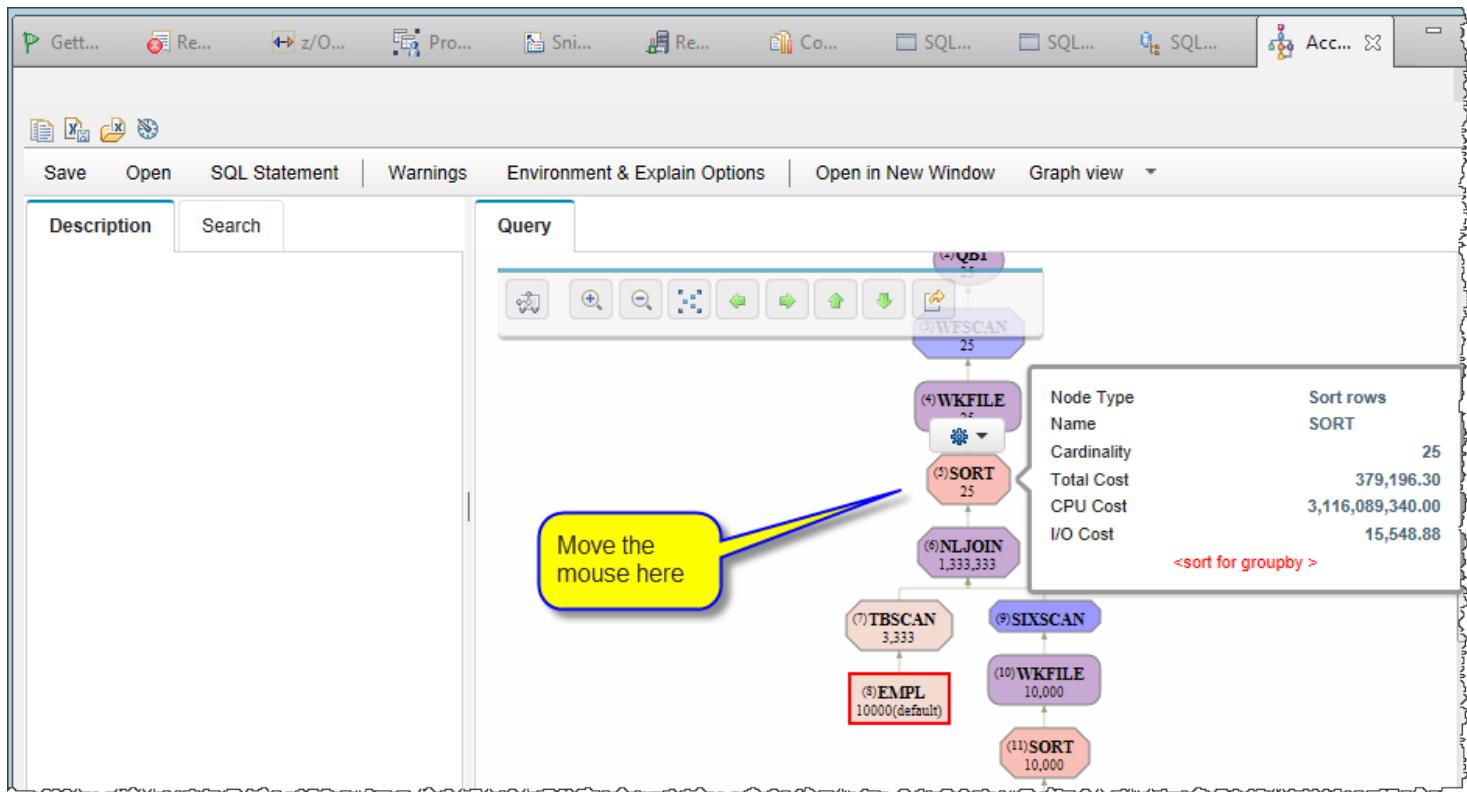
7.4.1 Right click on the first Select and choose **Open Visual Explain**



7.4.2 ► Click **Finish**. A new report view (*Access Plan Diagram*) will be generated in the bottom of the screen...



7.4.3 ► Double click on the tab to have a bigger view.
You will have the report below to work with the SQL explain

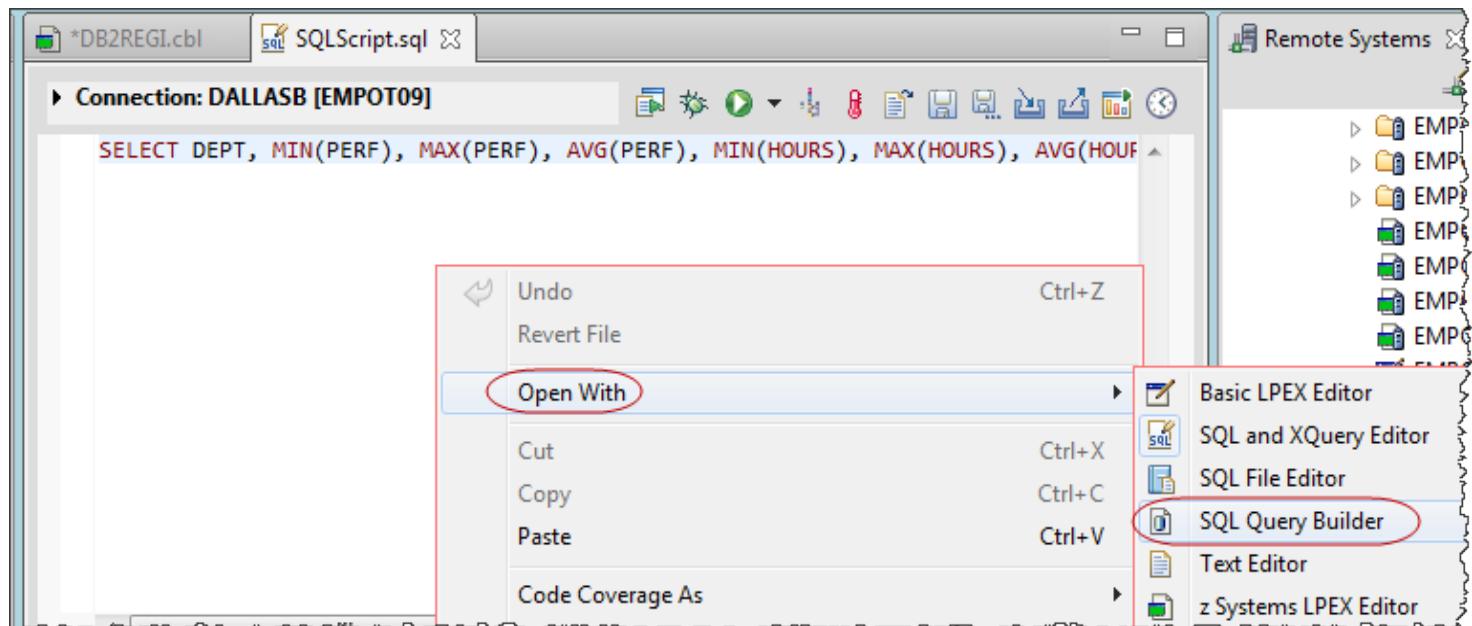


7.5 Reverse engineer the Query

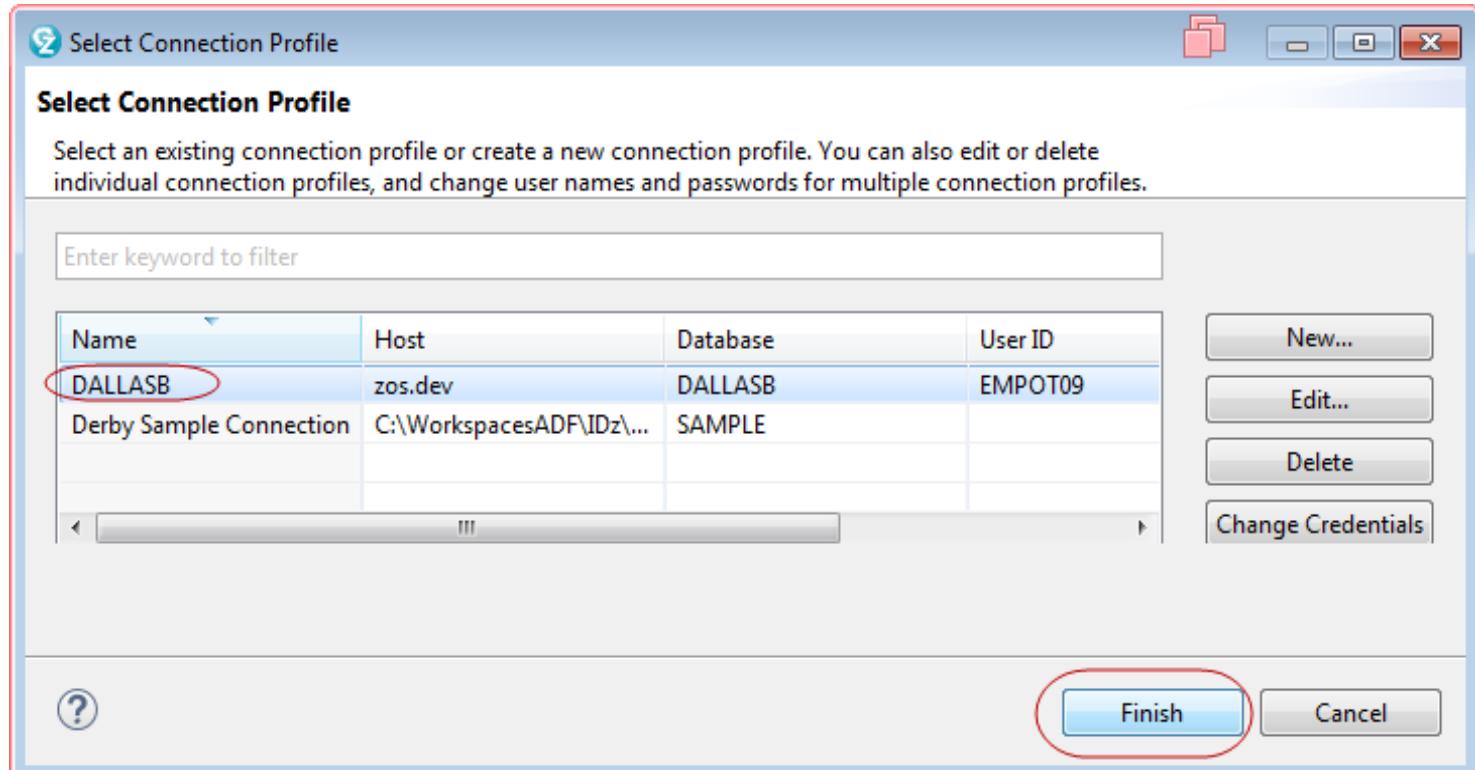
7.5.1 ► Using the SQL Outline view, right click on the Select and choose **Show in SQL Editor**

The screenshot shows the IBM DB2 SQL Outline view. On the left, there's a tree structure under 'Schemas' showing 'RBAROSA' and 'EMPL' tables, and 'PAY' and another 'SELECT' statement. A context menu is open over the first 'SELECT' statement, listing options: 'Find in Source', 'Run SQL', 'Show in SQL Editor' (which is highlighted with a red oval), 'Export SQL to File...', 'Compare', and 'Retrieve EXPLAIN Data'. The 'Show in SQL Editor' option is clearly circled in red.

7.5.2 ► When opened,, select **Open With > SQL Query Builder**



7.5.3 ► Choose **DALLASB** and click **Finish**



7.5.4 Now we have the graphical editor

The screenshot shows a graphical interface for editing SQL queries. At the top, there are three tabs: *DB2REGI.cbl, SQLScript.sql (which is selected), and SQLScript.sql. The main area contains an SQL query:

```
SELECT E.DEPT, MIN(E.PERF), MAX(E.PERF), AVG(E.PERF), MIN(P.HOURS),
       MAX(P.HOURS), AVG(P.HOURS)
  FROM RBAROSA.EMPL AS E, RBAROSA.PAY AS P
 WHERE E.NBR = P.NBR AND E.PERF > :PERF
 GROUP BY E.DEPT
```

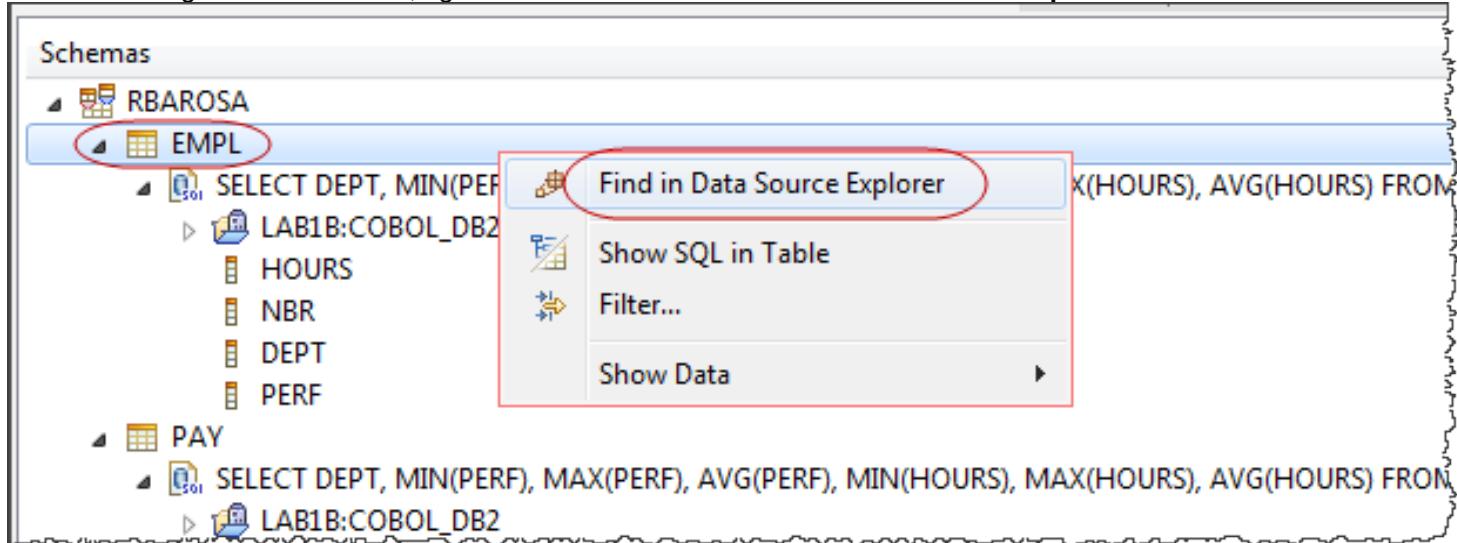
Below the query is a graphical representation of the database schema. It shows two tables, E and P, connected by a diamond symbol indicating a many-to-one relationship. Table E has columns NBR, LNAME, FNAME, DOB, and UDEPT. Table P has columns NBR, HOURS, RATE, DED, and VTP. A checkbox labeled "DISTINCT" is present.

The bottom section contains a table with tabs for Columns, Conditions, Groups, and Group Conditions. The Columns tab is selected, showing a single row for the column E.DEPT, which is checked under the Output column.

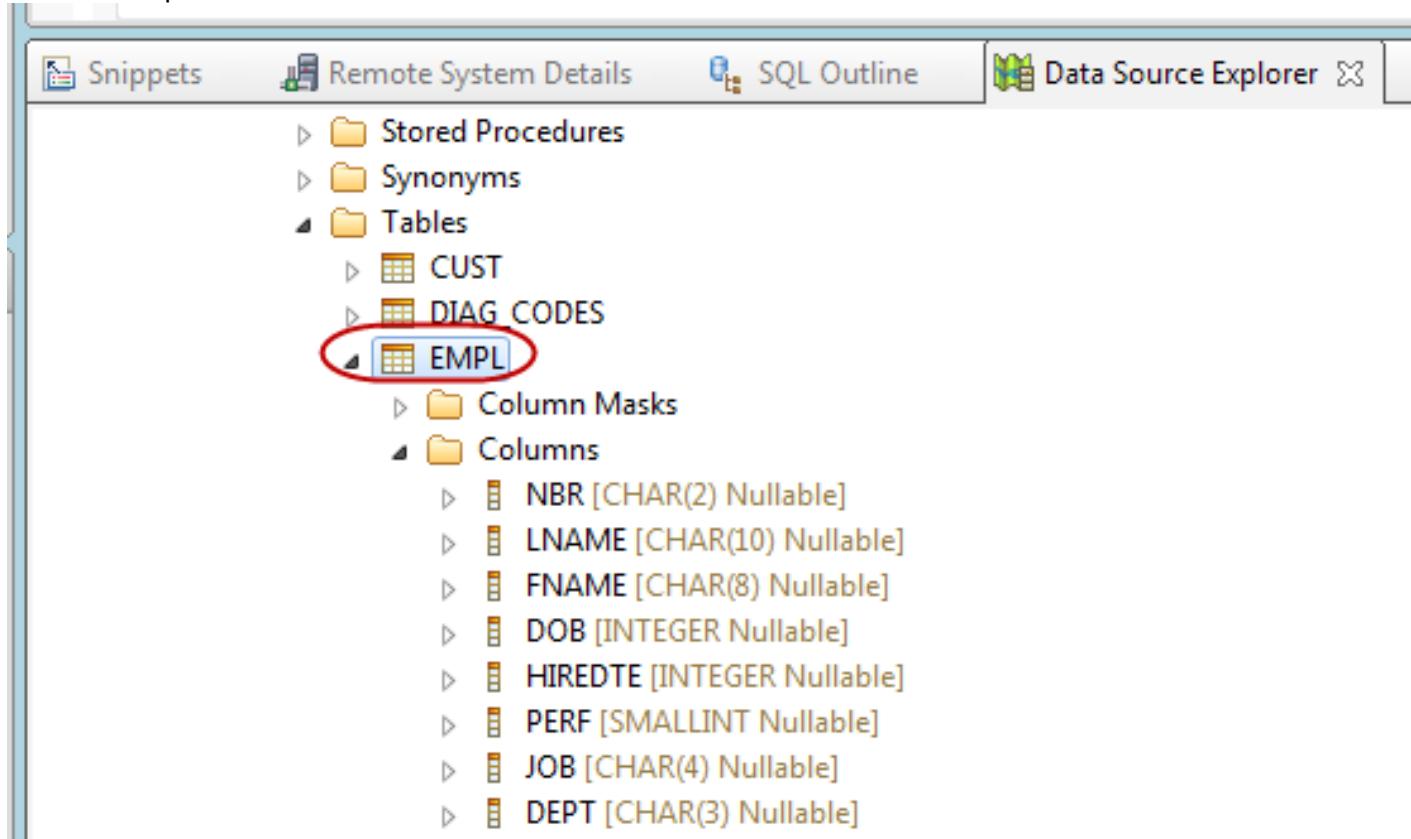
7.5.5 ► You might use this graphical editor to make changes on SQL statements. Notice that the changes reflects on the COBOL code..

7.6 Displaying DB2 table content

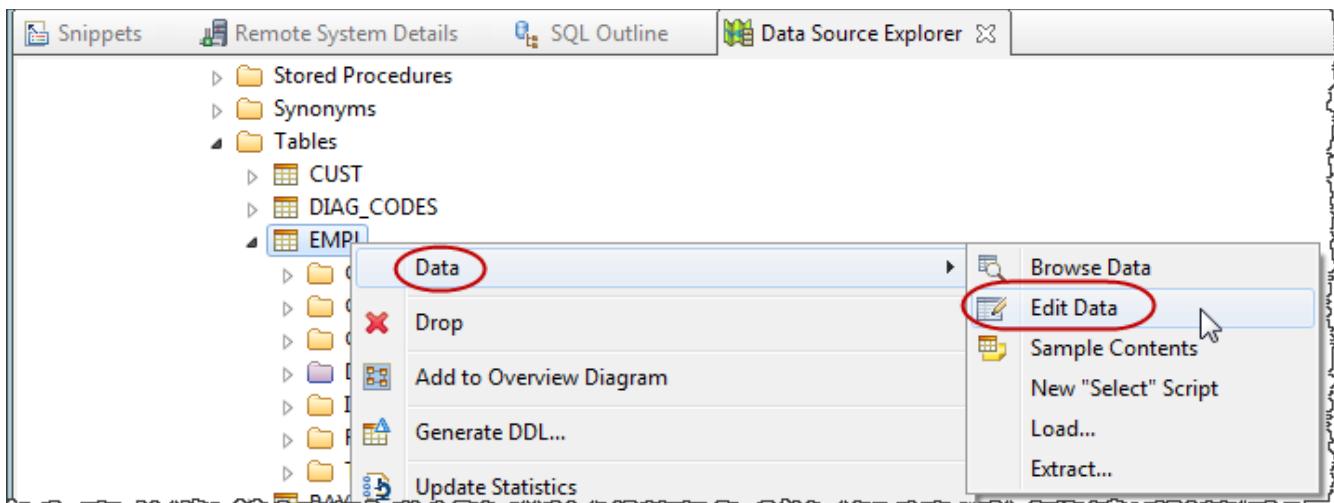
7.6.1 ► Using SQL Outline view, right click on **EMPL** and select **Find Data Source Explorer**



7.6.2 ► Expand **Tables** and **EMPL**

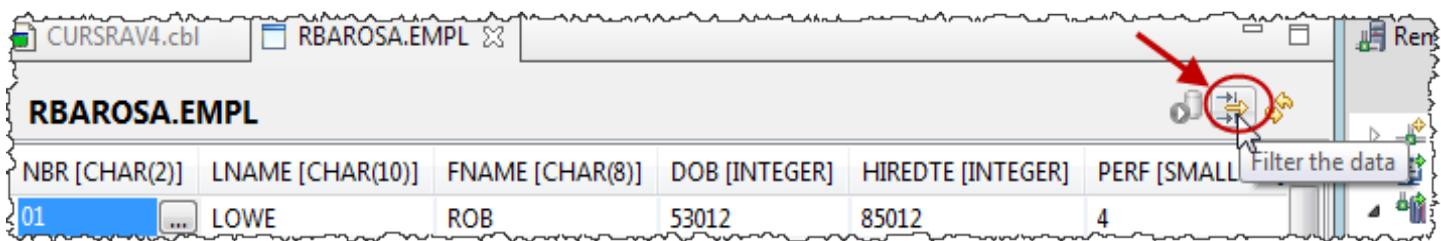


7.6.3 ► Right click on **EMPL** and select **Data > Edit Data**

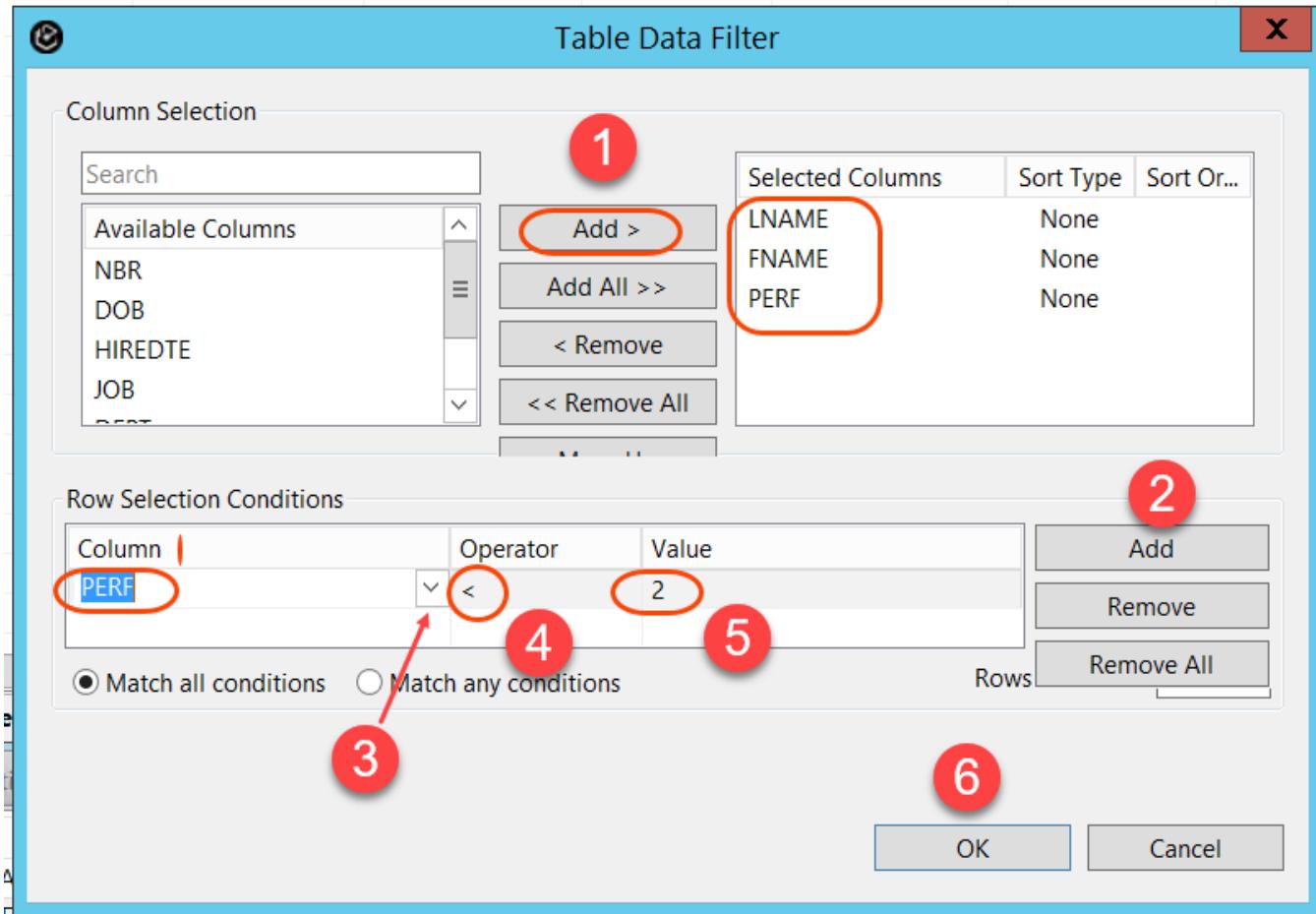


Filtering data results.

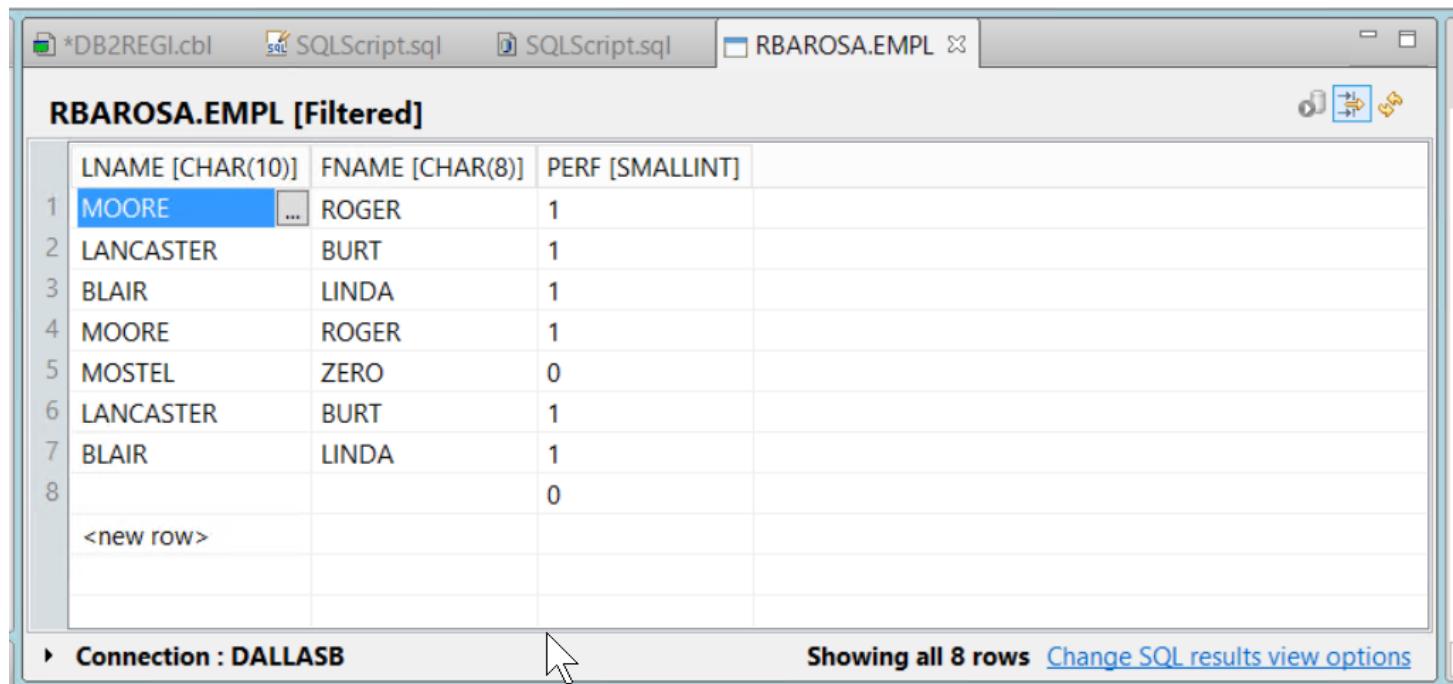
7.6.4 ► Click **Filter** the data icon



7.6.5 ► On the *Table Data Filter* dialog, using **Add >** button, select **LNAME**, **FNAME** and **PERF**. Using the **second Add** button, use the pull down and select **EMPL.PERF** for *Column*, **<** as *Operator* and type **2** as *Value*.



7.6.6 ► Clicking **OK** you will get the results below:



The screenshot shows the IBM i Navigator interface with the RBAROSA.EMPL table open. The table has three columns: LNAME [CHAR(10)], FNAME [CHAR(8)], and PERF [SMALLINT]. The data is as follows:

	LNAME [CHAR(10)]	FNAME [CHAR(8)]	PERF [SMALLINT]
1	MOORE	ROGER	1
2	LANCASTER	BURT	1
3	BLAIR	LINDA	1
4	MOORE	ROGER	1
5	MOSTEL	ZERO	0
6	LANCASTER	BURT	1
7	BLAIR	LINDA	1
8			0
	<new row>		

At the bottom, there are buttons for 'Connection : DALLASB' and 'Showing all 8 rows [Change SQL results view options](#)'. There are also icons for saving, printing, and closing.

7.6.7 ►| Close all editors pressing **Ctrl + Shift + F4**. Or just click on the  of each opened editor

►| Say **NO** to saving the COBOL program.

Section 8. (Optional) Using File Manager

ADFz may optionally have the File Manager (FM) plugin installed. FM works with a broad spectrum of z/OS files and data bases, including VSAM, IAM, and QSAM files, PDS and libraries, DB2 and IMS databases, HFS files, OAM files, CICS queues, MQ queues, and tapes.

File Manager provides powerful formatted editors and viewers, and also provides a full complement of on-line and batch utilities to copy, extract, and load data, to create files and databases, compare and print, and many other utility functions.

File Manager has a conventional 3270 interface that can be accessed from TSO or CICS. here is also an eclipse GUI interface that is available on ADFz.

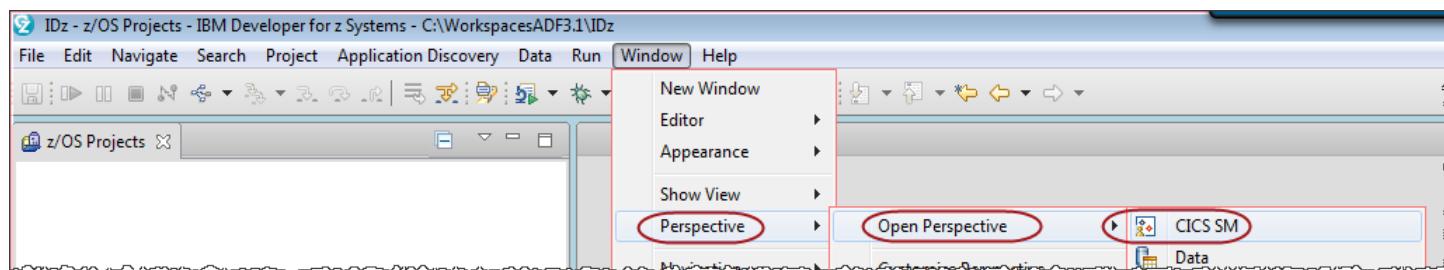
File Manager has many capabilities we will explore just few examples on this optional lab.

Note that this exercise could be also done with either IDz version 14 or version 15. Here we are using version 14, but instructions will be the same.

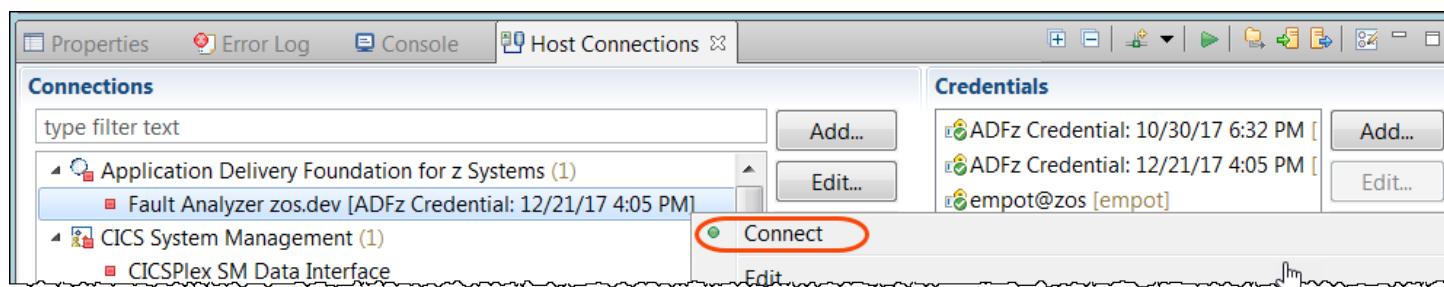
8.1 Verify that you are connected to the PDTTOOLS Common components

On step, 3.1 you have used the PDTTOOLS Common Components. Let's be sure that you are still connected.

8.1.1 ► Go to **CICS SM** perspective using
Window > Perspective> Open Perspective > CICS SM
 (If CICS SM is not shown click **Other** and find **CICS SM**)



8.1.2 ► Click on **Host Connections** tab (bottom) and verify that **Application Delivery Foundation for z Systems** has a green icon:
 If it is not green you must connect it. **Right click and select Connect**

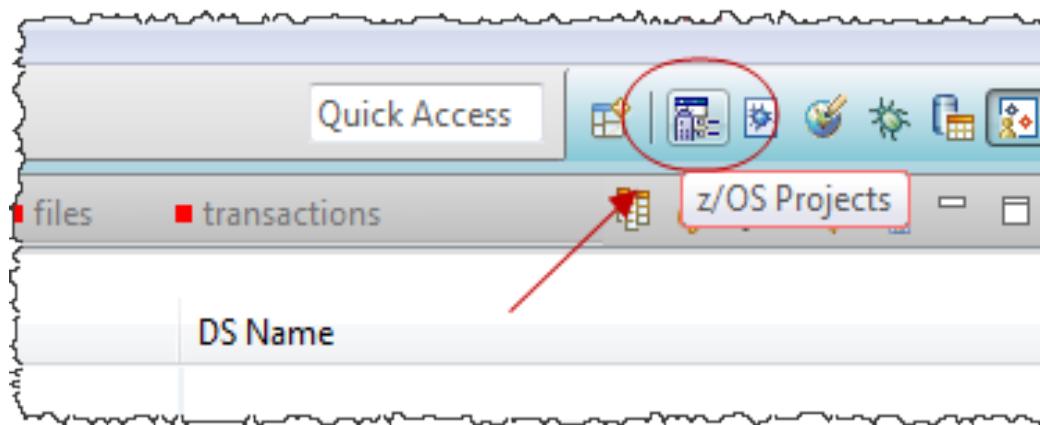


8.2 Using View Load Module utility

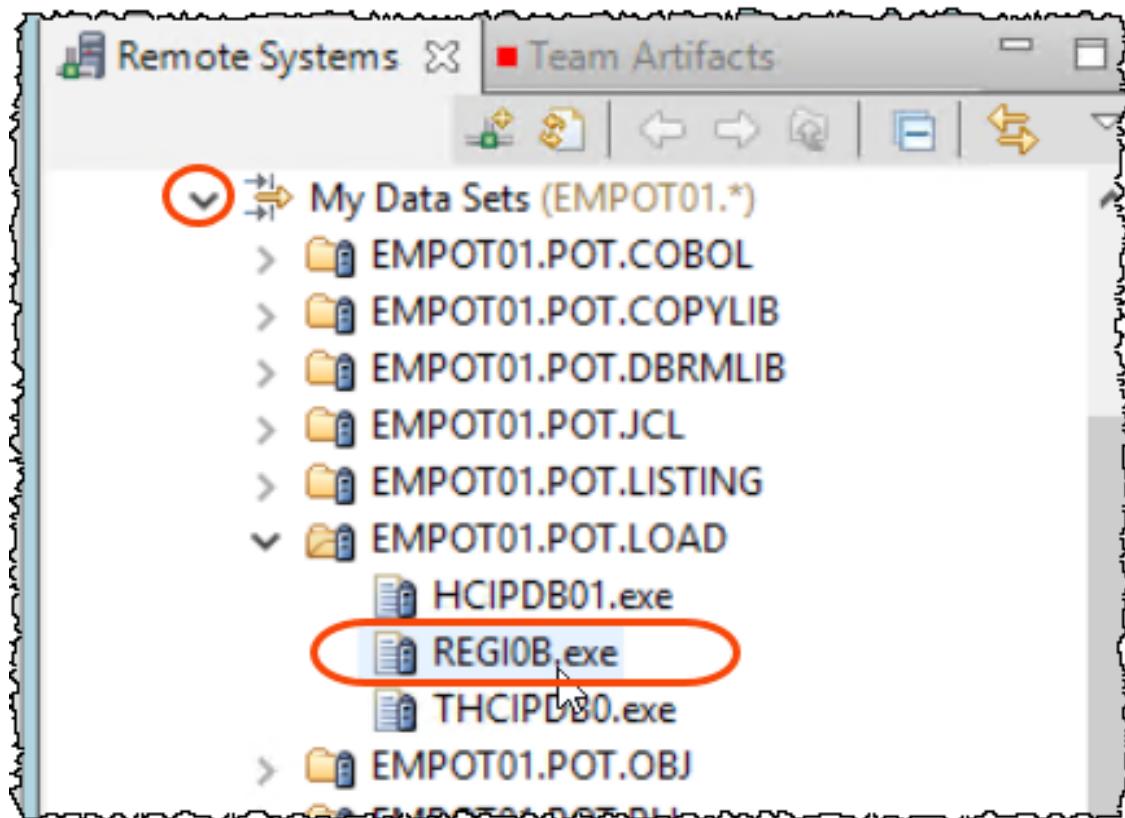
You can use the **View Load Module** utility function to print a list of the symbols (CSECTs, common sections, entry points, and ZAPs) in a load module. You also could compare load modules, using the Compare wizard.

This page details the mapping of fields to batch parameters and miscellaneous notes. For the full description of each parameter, refer to the [IBM File Manager Users Guide and Reference](#).

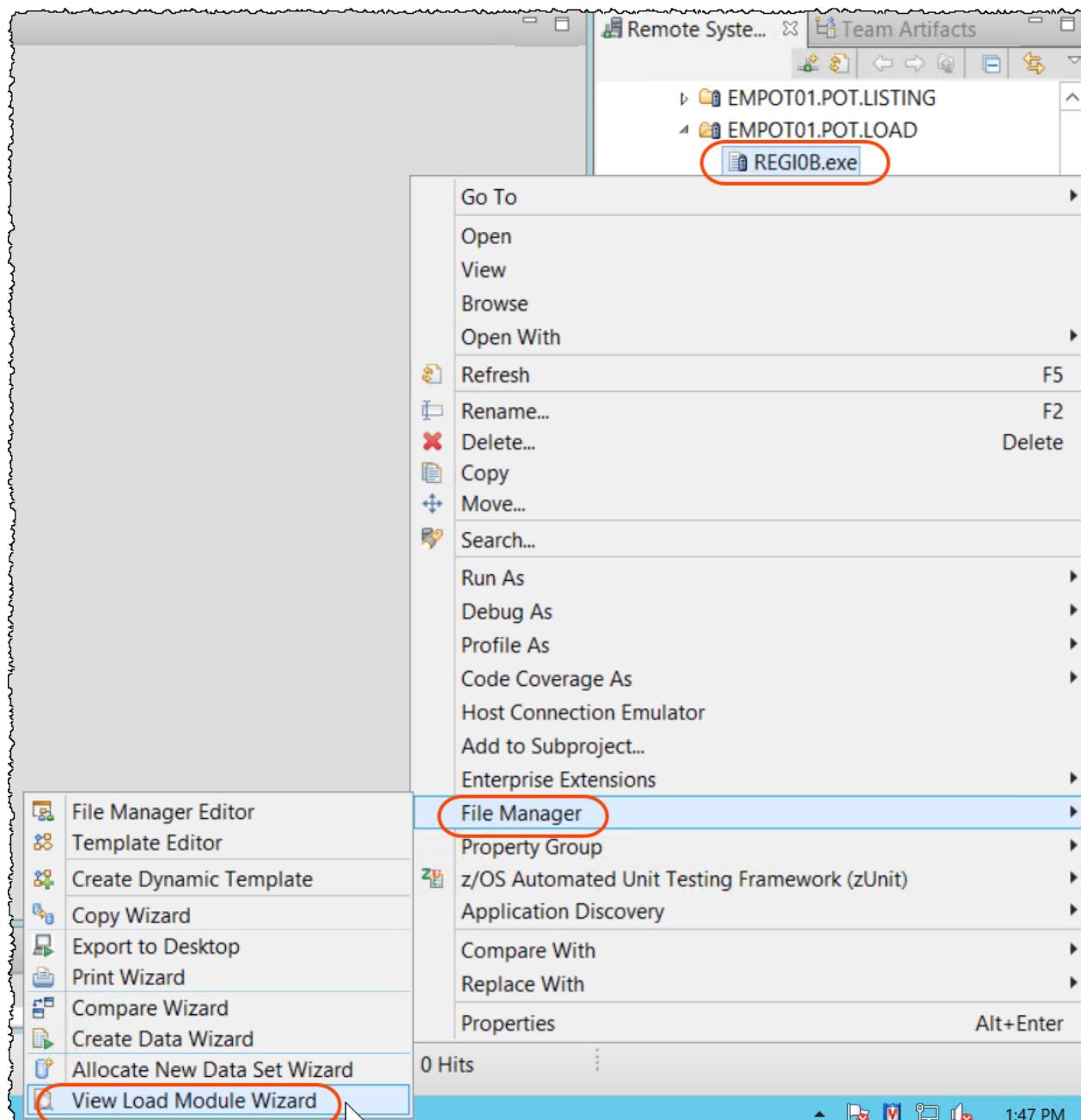
- 8.2.1 ► Go to **z/OS Projects** perspective clicking on the upper top right icon as below



- 8.2.2 ► Using Remote System view on the right, expand **MVS Files**, **My Data Sets** and **EMPOT01.POT.LOAD** and you should see the load module **REGI0B.exe** that you created before..

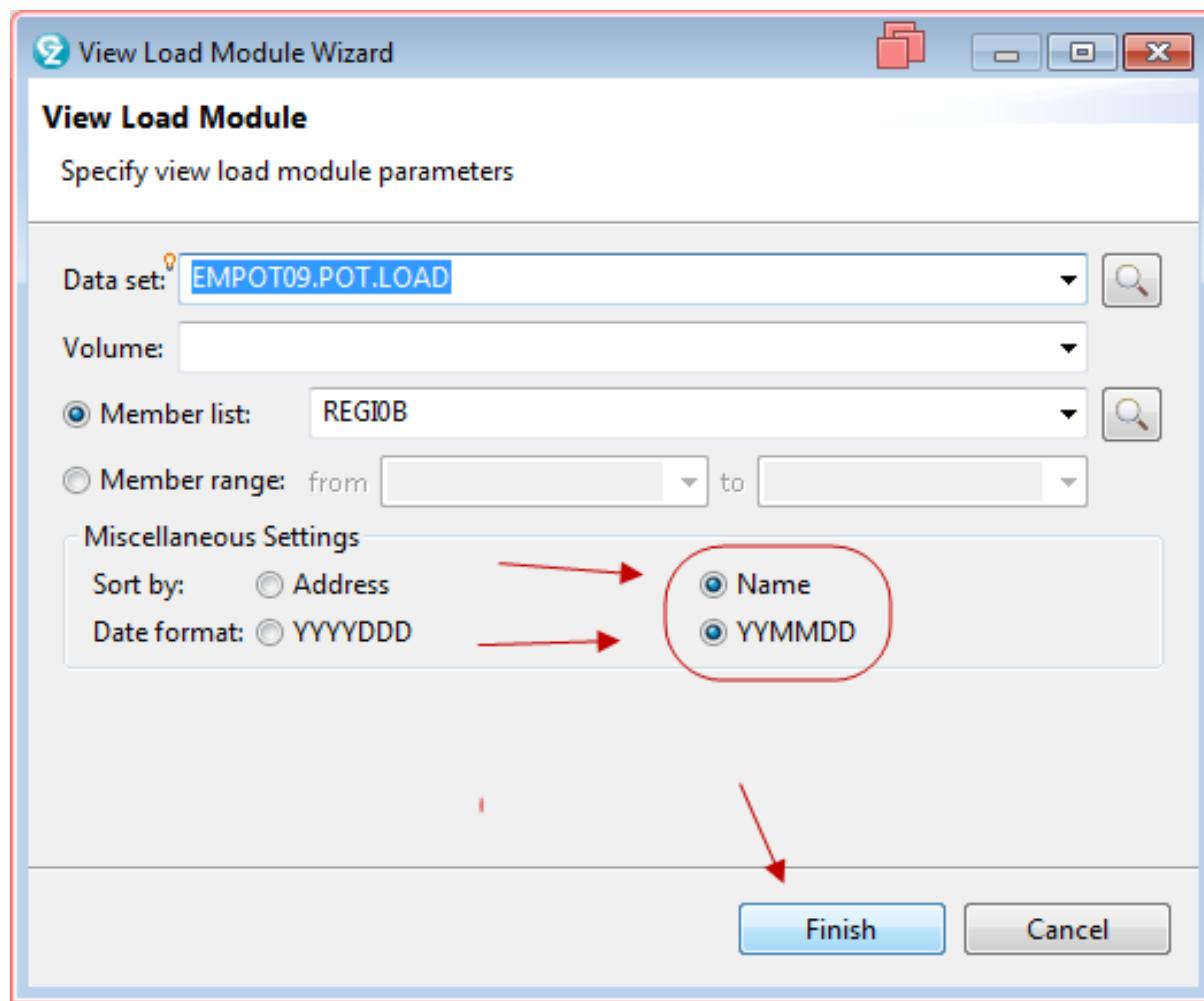


8.2.3 ► Right click on **REGI0B.exe** and select **File Manager** and **View Load Module Wizard**.



► If you get a **Sign on** dialog for **Fault Analyzer** use **empot01** credentials and click **OK**

8.2.4 ► Select **Name** to sort by name and **YYMMDD** to show the date on this format and click **Finish**.



8.2.5 The result is below..

► Scroll down and you may find interesting the date and time that this module was linked as well other information. You also could compare load modules.

The screenshot shows the Rational Application Developer environment. On the left is a code editor window titled "tmp1559069406042.txt" displaying assembly-like code. On the right is a file system browser titled "Remote System..." showing a tree structure of files and directories under "zos.dev". A red oval highlights the line "Linked on 19/05/28 at 10:33:32 by PROGRAM BINDER 5695-PMB V2R2" in the editor, which corresponds to the linked date mentioned in the text below.

```

000018 0
000019 Load Module Information
000020
000021 Load Library EMPOT01.POT.LOAD
000022 Load Module REGI0B
000023 Linked on 19/05/28 at 10:33:32 by PROGRAM BINDER 5695-PMB V2R2
000024 EPA 000000 Size 000142C TTR 000005 SSI AC 00 AN
000025
000026 Name Type Address Size Class A/RMODE Compiler 1
000027 -----
000028 -PRIVATE PC 0000000 0000008 C_@PPA2 ANY/ANY
000029 -PRIVATE PC 0000000 0000004 C_@CSINIT ANY/ANY
000030 -PRIVATE PC 0000000 0000084 C_WSA ANY/ANY
000031 CEEARLU SD 0000718 00000B8 B_TEXT MIN/ANY PL/X 390 V2R4
000032 CEEBETBL SD 00004C0 0000028 B_TEXT MIN/ANY HIGH-LEVEL AS
000033 CEEBINT SD 0000710 0000008 B_TEXT MIN/ANY PL/X 390 V2R4
000034 CEEBLIST SD 00006B0 000005C B_TEXT MIN/ANY HIGH-LEVEL AS
000035 CEELLIST LD 00006C0 000004C B_TEXT
000036 CEEBPIRA SD 00007D0 00002A0 B_TEXT 31/ANY PL/X 390 V2R4

```

Notice: The date that is shown is when your REGI0B module was created and will be different from what is shown on your report.

8.2.6 ► Close the edited file. Can use **CTRL + Shift + F4** to close all opened editors. Or just click on the of each opened editor

8.3 Working with z/OS data sets

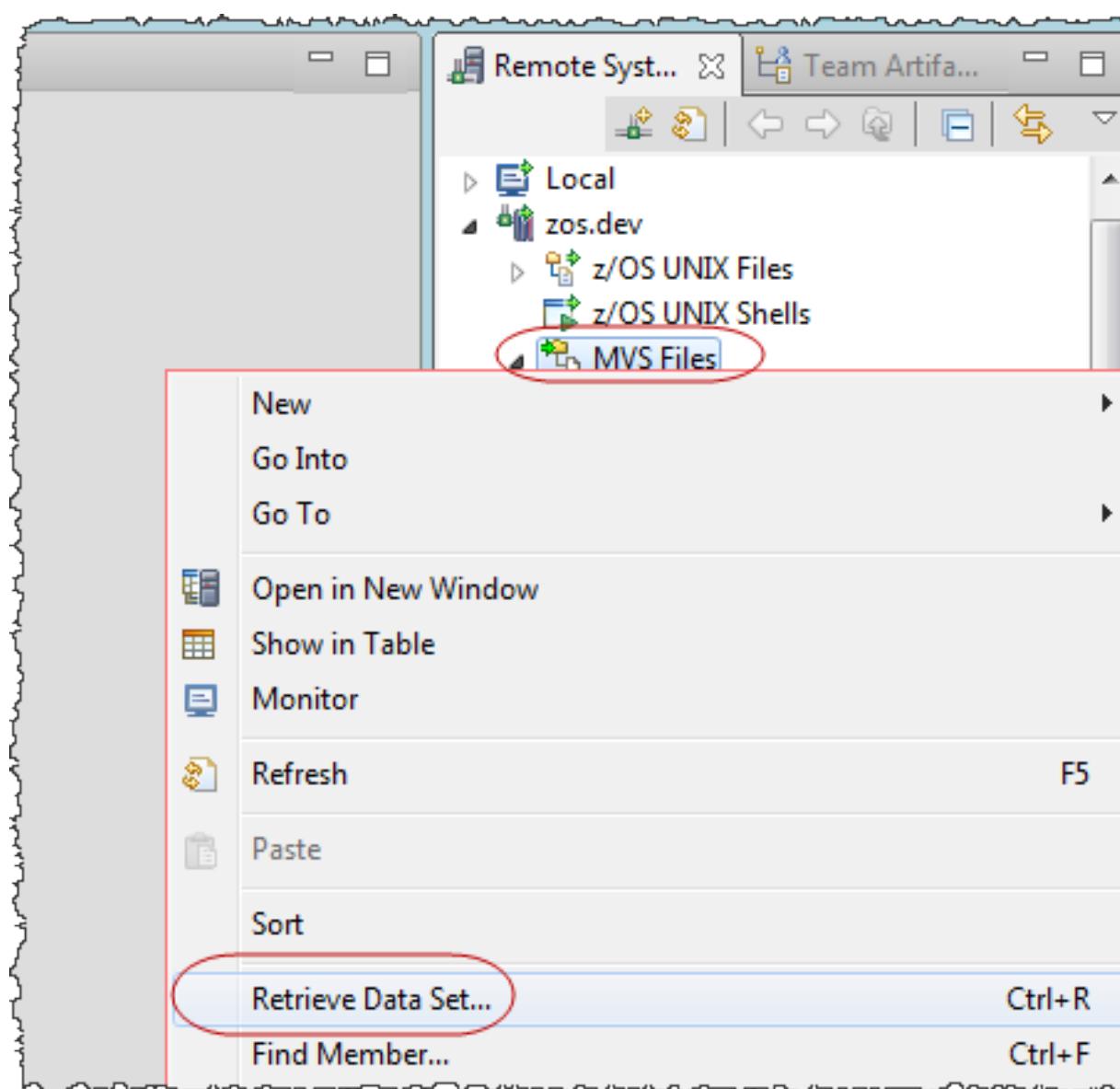
In this section, you will open a variable blocked sequential dataset in the editor or viewer, using a copybook as a layout.

You will use the very basic features of the editor, including navigating a file, we will not cover finding data in a file, changing data, inserting, deleting and sorting records.

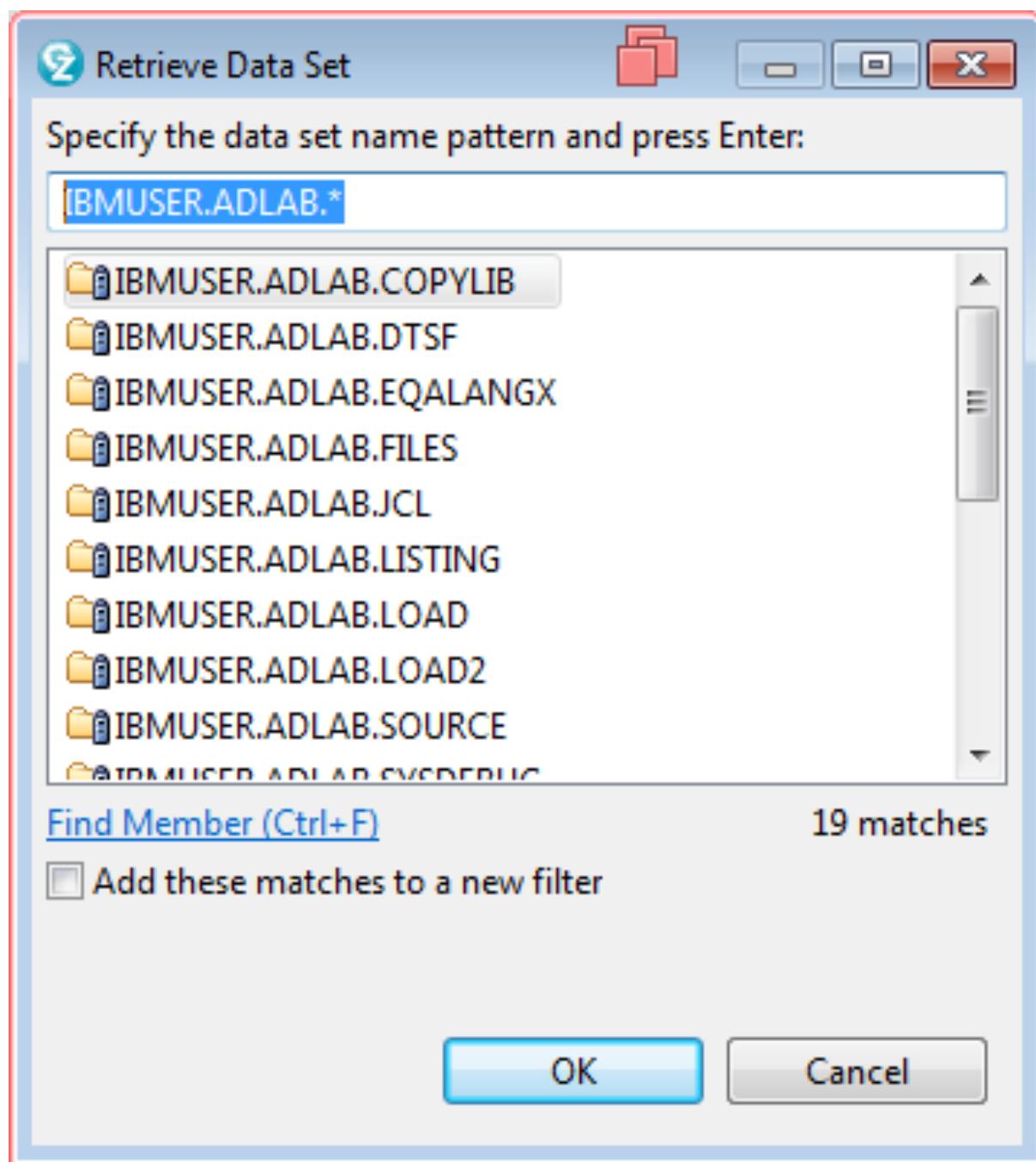
8.3.1 Before you start mapping the dataset using a COBOL COPYBOOK, let us copy the copybook that maps the file to your PDS.

► Using the *Remote Systems* view, right click on **MVS Files** and select **Retrieve Data Set**

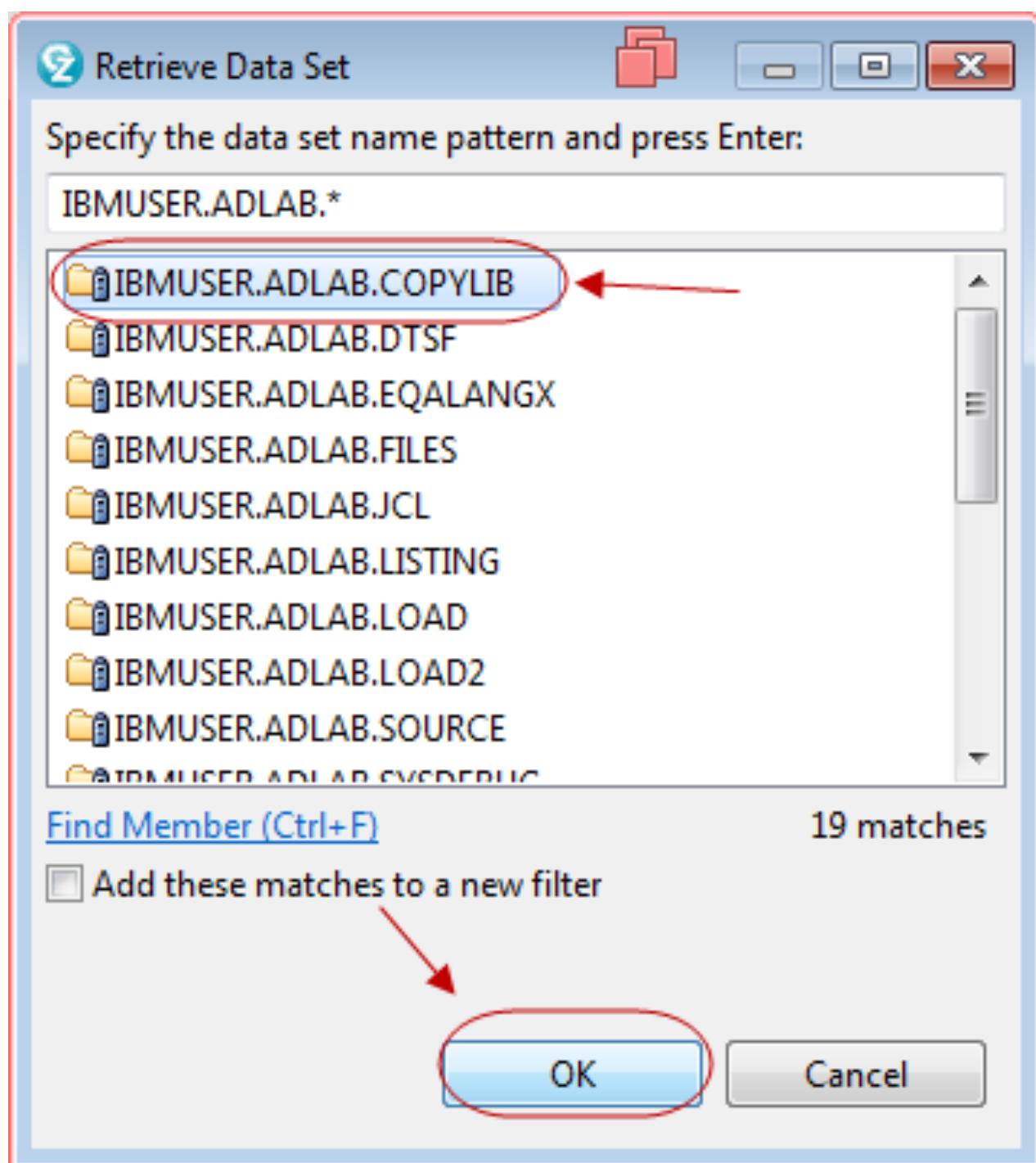
This is equivalent at the TSO 3.4 function.



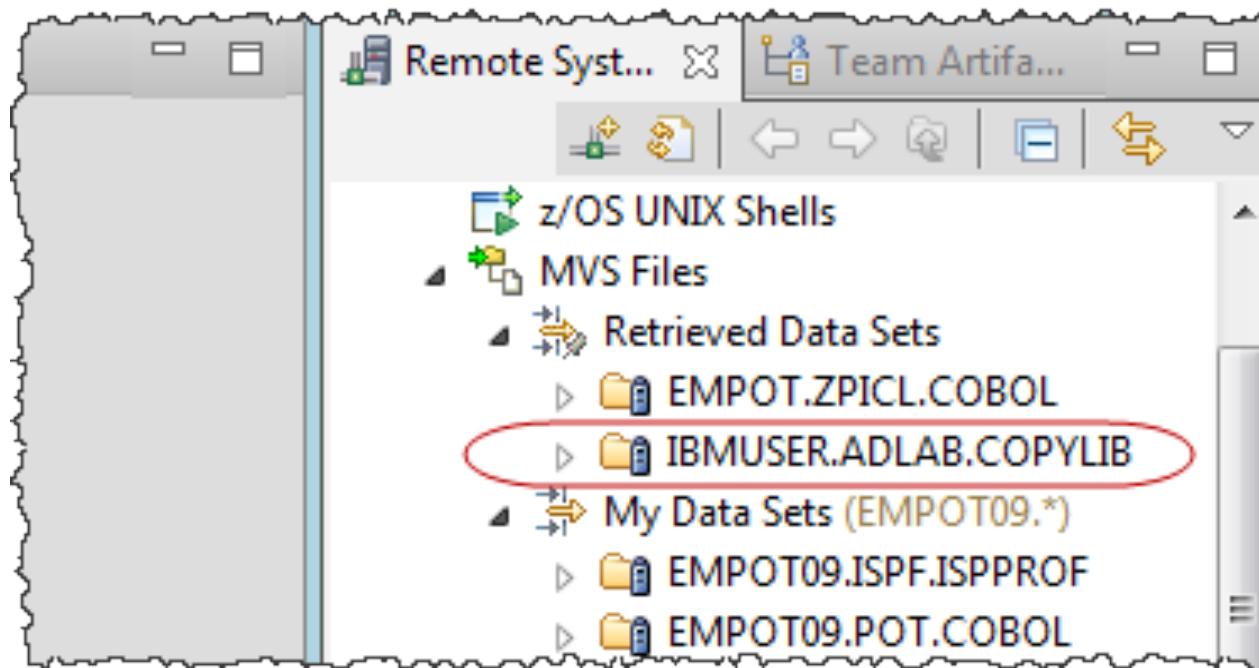
8.3.2 ► Type IBMUSER.ADLAB.* and press **Enter**.



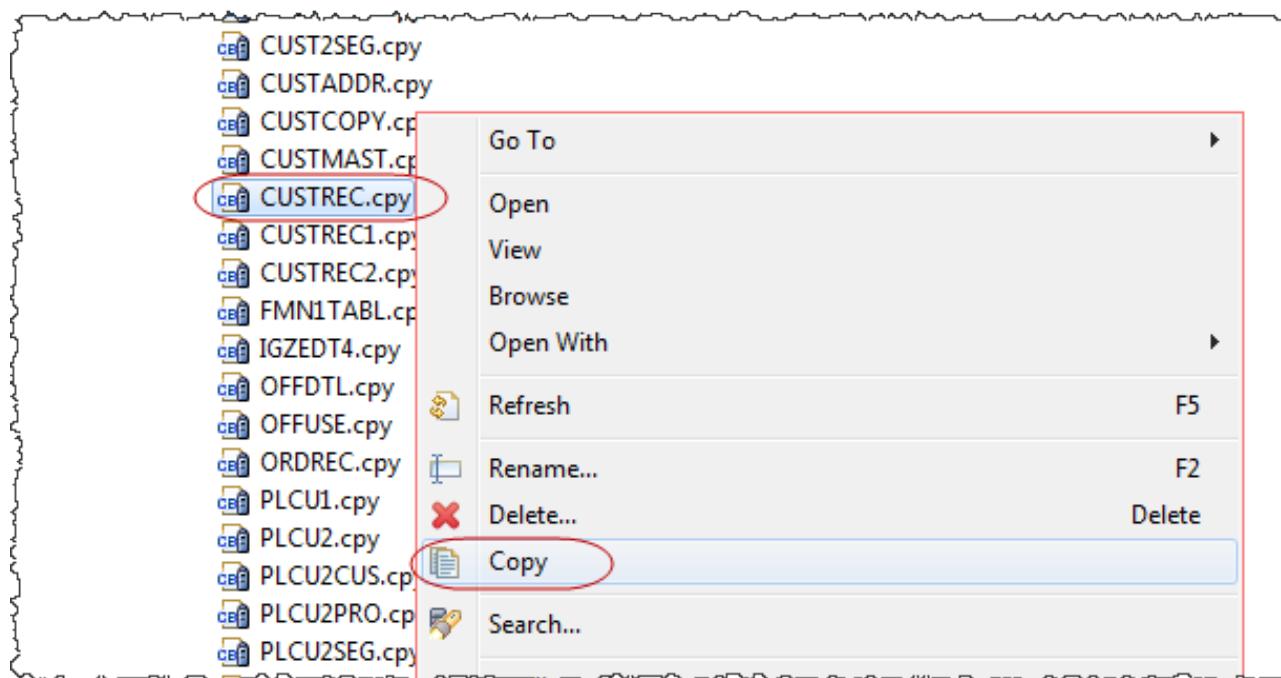
8.3.3 ► Select IBMUSER.ADLAB.COPYLIB and click OK.



8.3.4 The data set will be under Retrieved Data Sets.

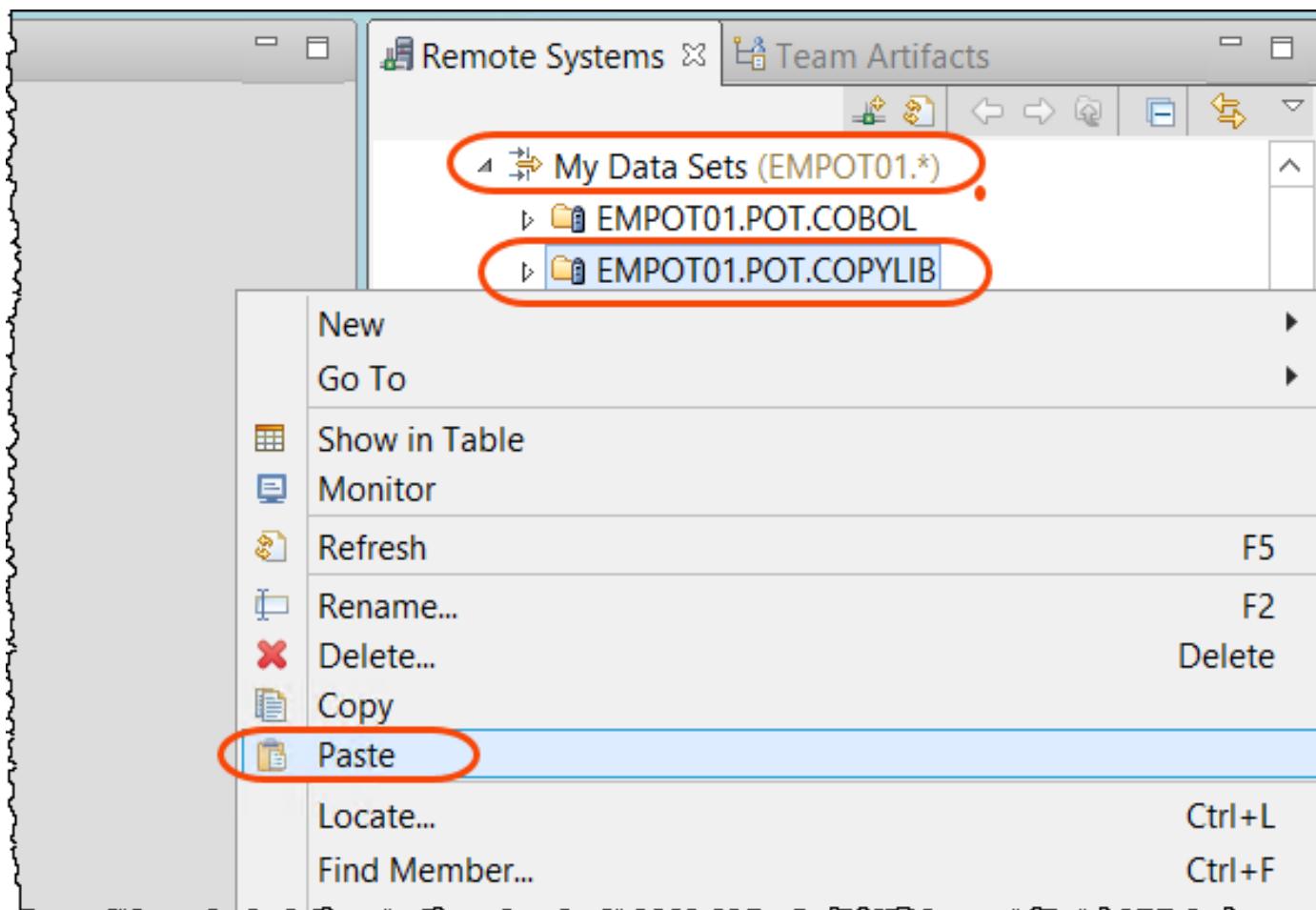


8.3.5 ► Expand **IBMUSER.ADLAB.COPYLIB**, right click on **CUSTREC.cpy** and select **Copy**.
You want to copy this member to your PDS.

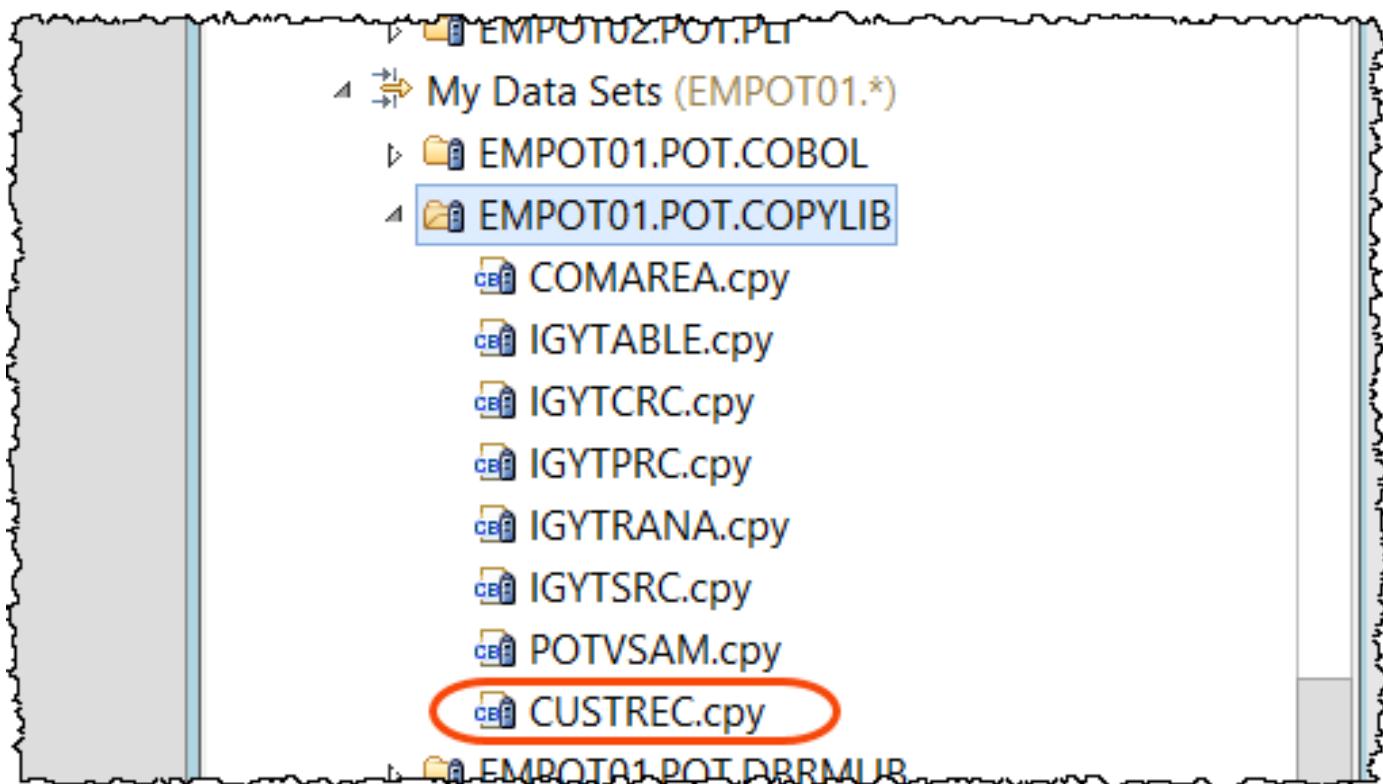


8.3.6 ► Navigate to **My Data Sets**, right click on **EMPOT01.POT.COPYLIB** and select **Paste**

If this member is already there select **Overwrite**



8.3.7 You should have:

8.3.8 ► Double click on **CUSTREC.cpy** to edit it.

This copybook is the file layout of the file that you will see later. Note than some fields are COMP-3.

```

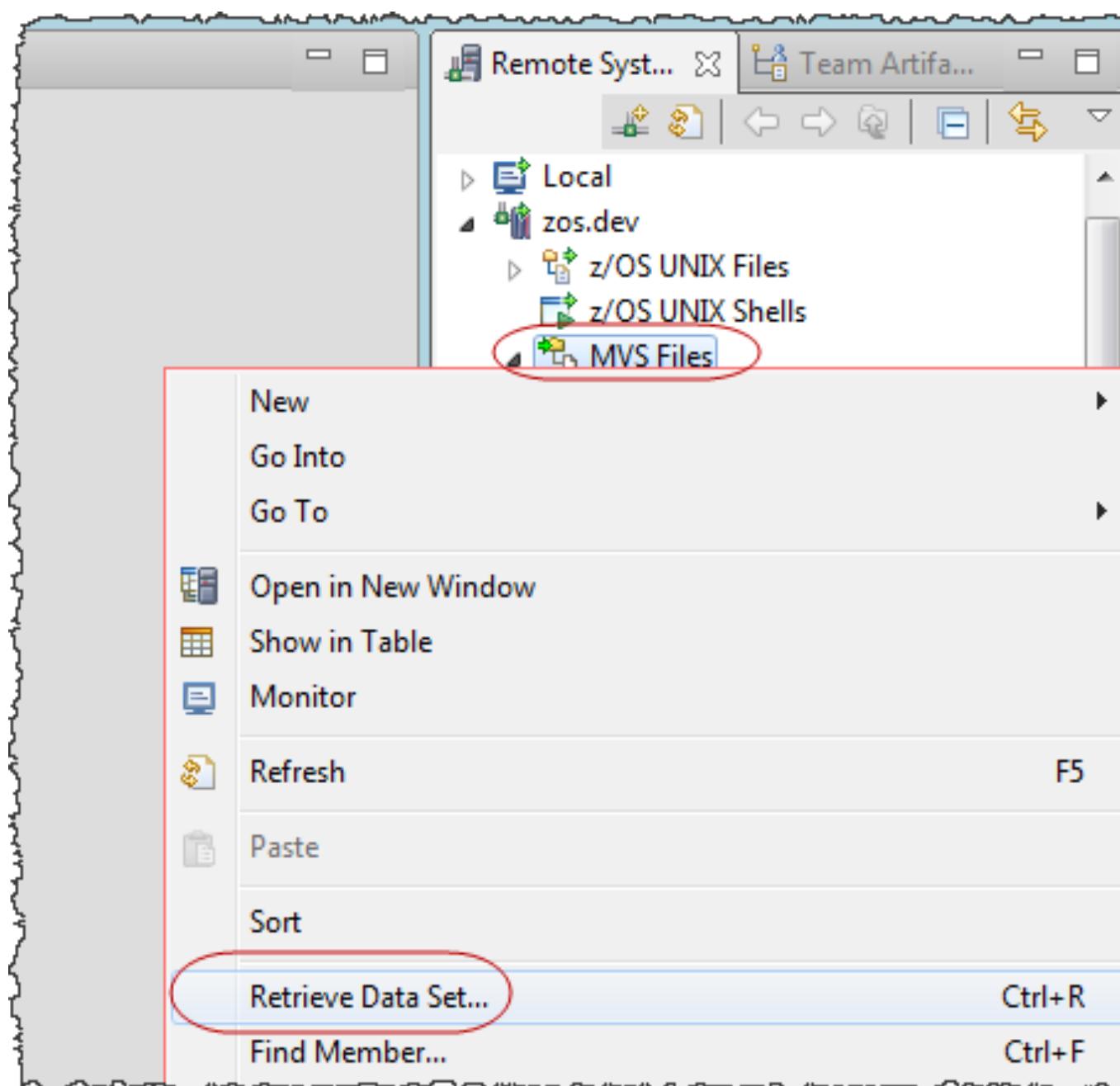
CUSTREC.cpy
-----+-----+-----+-----+-----+-----+-----+-----+
1 | *** ++++++-----+-----+-----+-----+-----+-----+
2 | * Sample COBOL Copybook for IBM PD Tools Workshops
3 |
4 | * The sample data described by this copybook
5 | * is <USERID>.ADLAB.CUSTFILE
6 | *** ++++++-----+-----+-----+-----+-----+-----+
7 | 01 CUST-REC.
8 |   05 CUSTOMER-KEY.
9 |     10 CUST-ID          PIC X(5).
10 |     10 REC-TYPE         PIC X.
11 |   05 NAME              PIC X(17).
12 |   05 ACCT-BALANCE      PIC S9(7)V99  COMP-3.
13 |   05 ORDERS-YTD        PIC S9(5)    COMP.
14 |   05 ADDR               PIC X(20).
15 |   05 CITY               PIC X(14).
16 |   05 STATE              PIC X(02).
17 |   05 COUNTRY             PIC X(11).
18 |   05 MONTH              PIC S9(7)V99  COMP-3  OCCURS 12.
19 |   05 OCCUPATION          PIC X(30).
20 |   05 NOTES              PIC X(120).
21 |   05 LAB-DATA-1          PIC X(05).

```

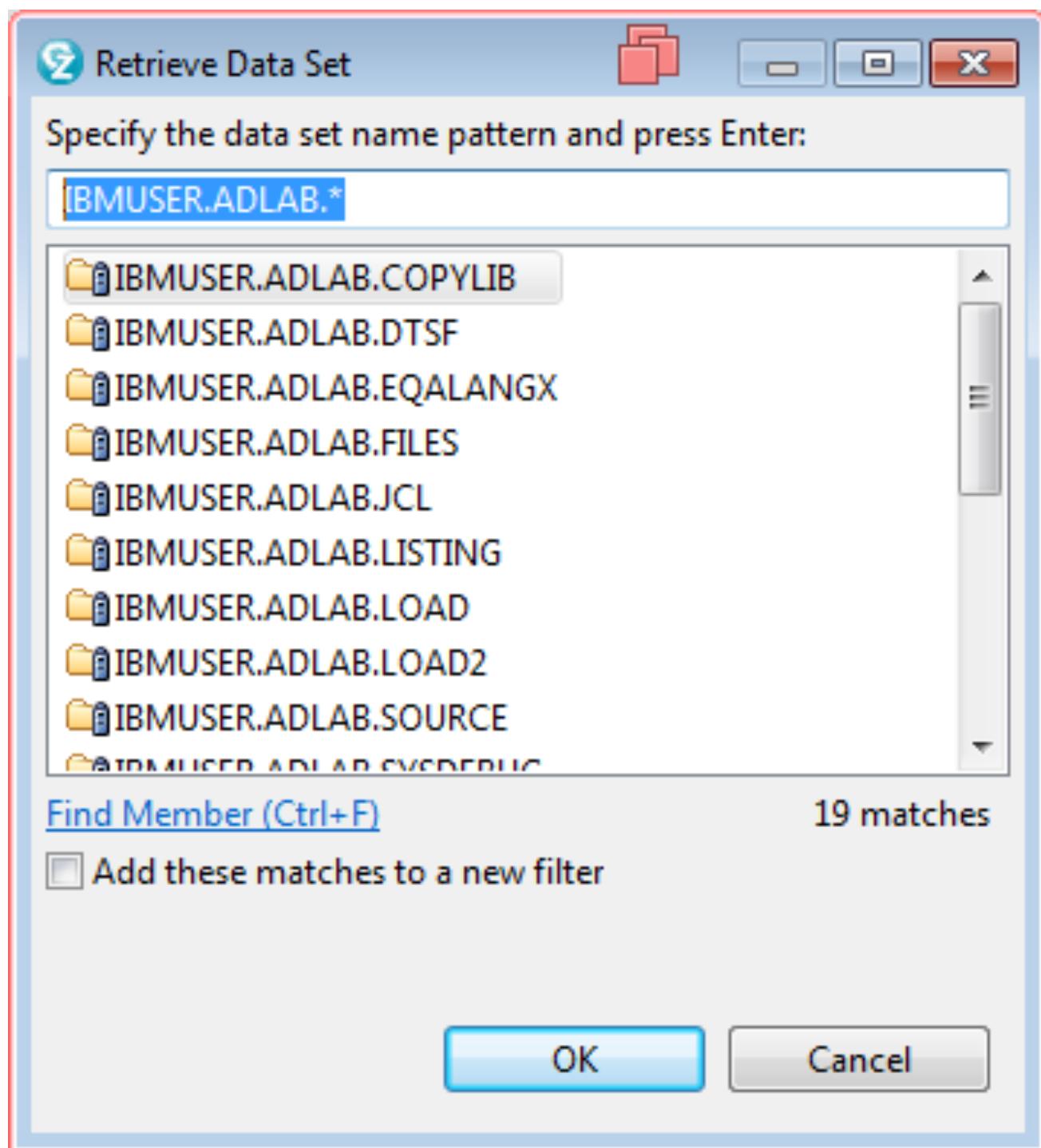
The screenshot shows the COBOL copybook `CUSTREC.cpy` in the main editor window. The sidebar on the right shows the project structure under `My Data Sets (EMPOT01.*)`, with the file `CUSTREC.cpy` also circled with a red oval. A red arrow points from the sidebar to the file in the editor.

8.3.9 ► To see the data set, using the *Remote Systems* view, right click on **MVS Files** and select **Retrieve Data Set**

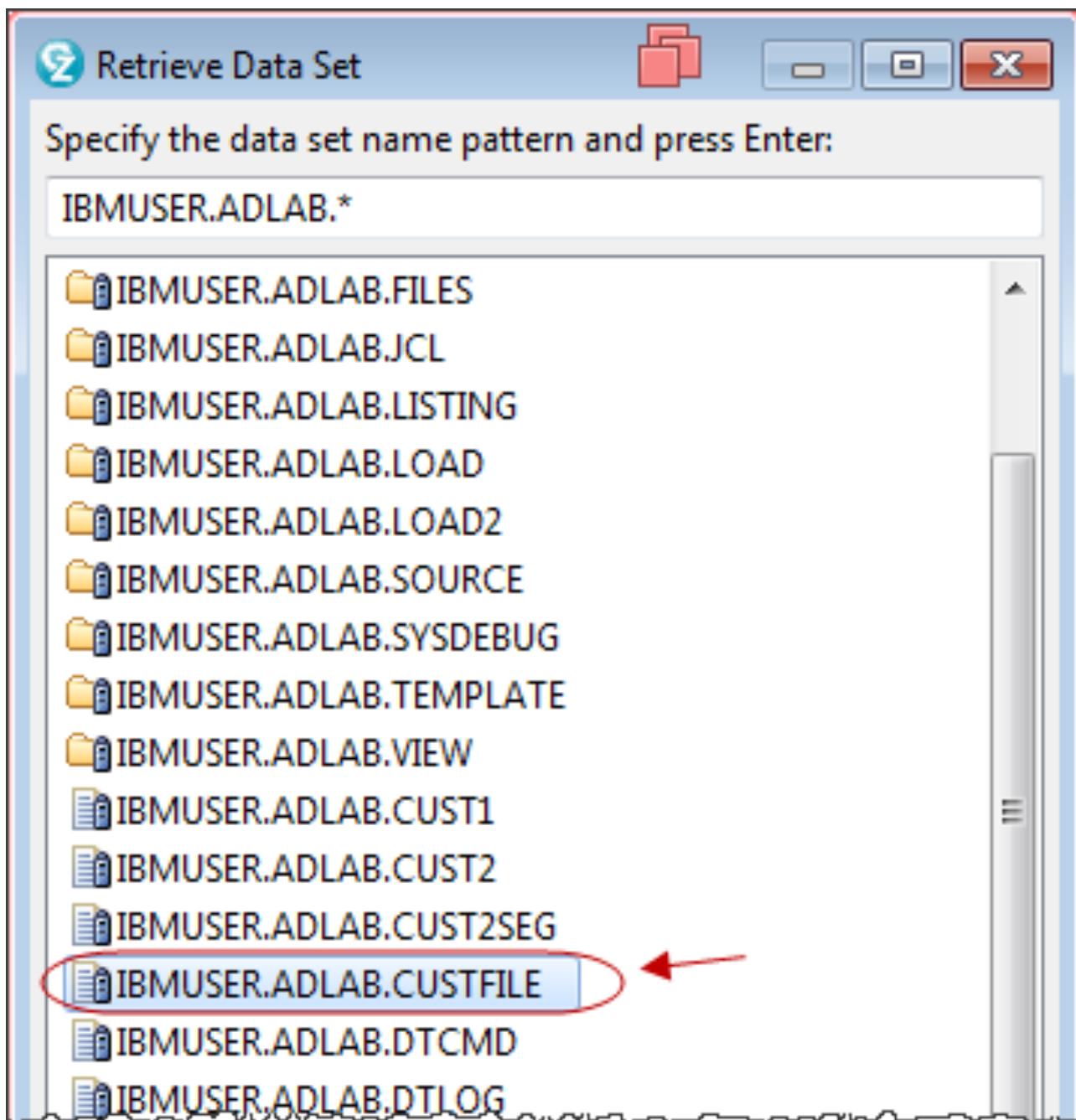
This is equivalent at the TSO 3.4 function.



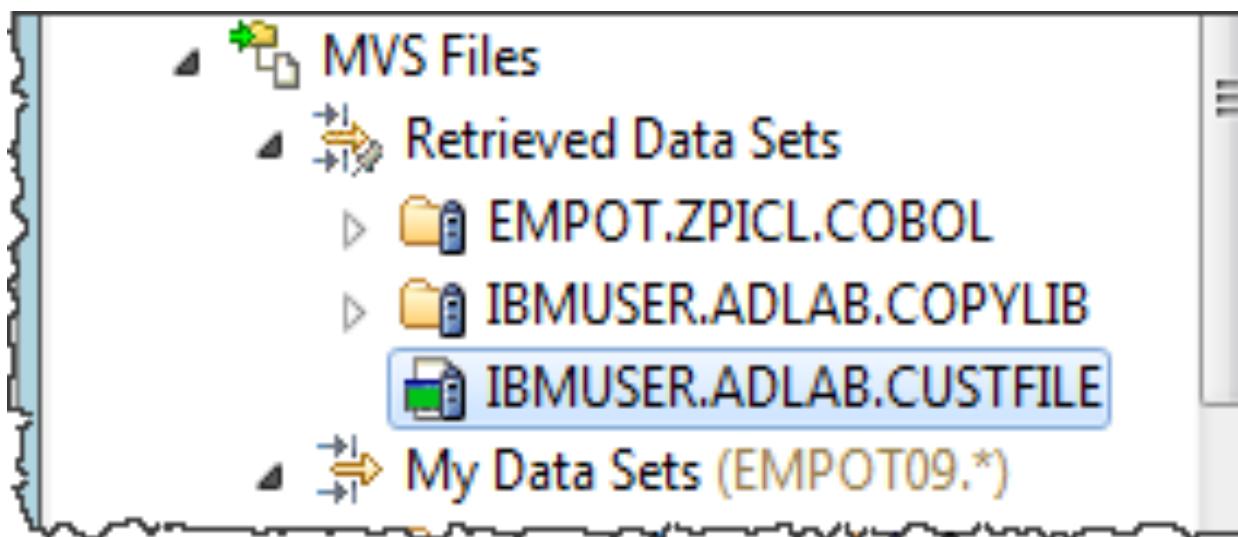
8.3.10 ➤ Type IBMUSER.ADLAB.* and press **Enter**.



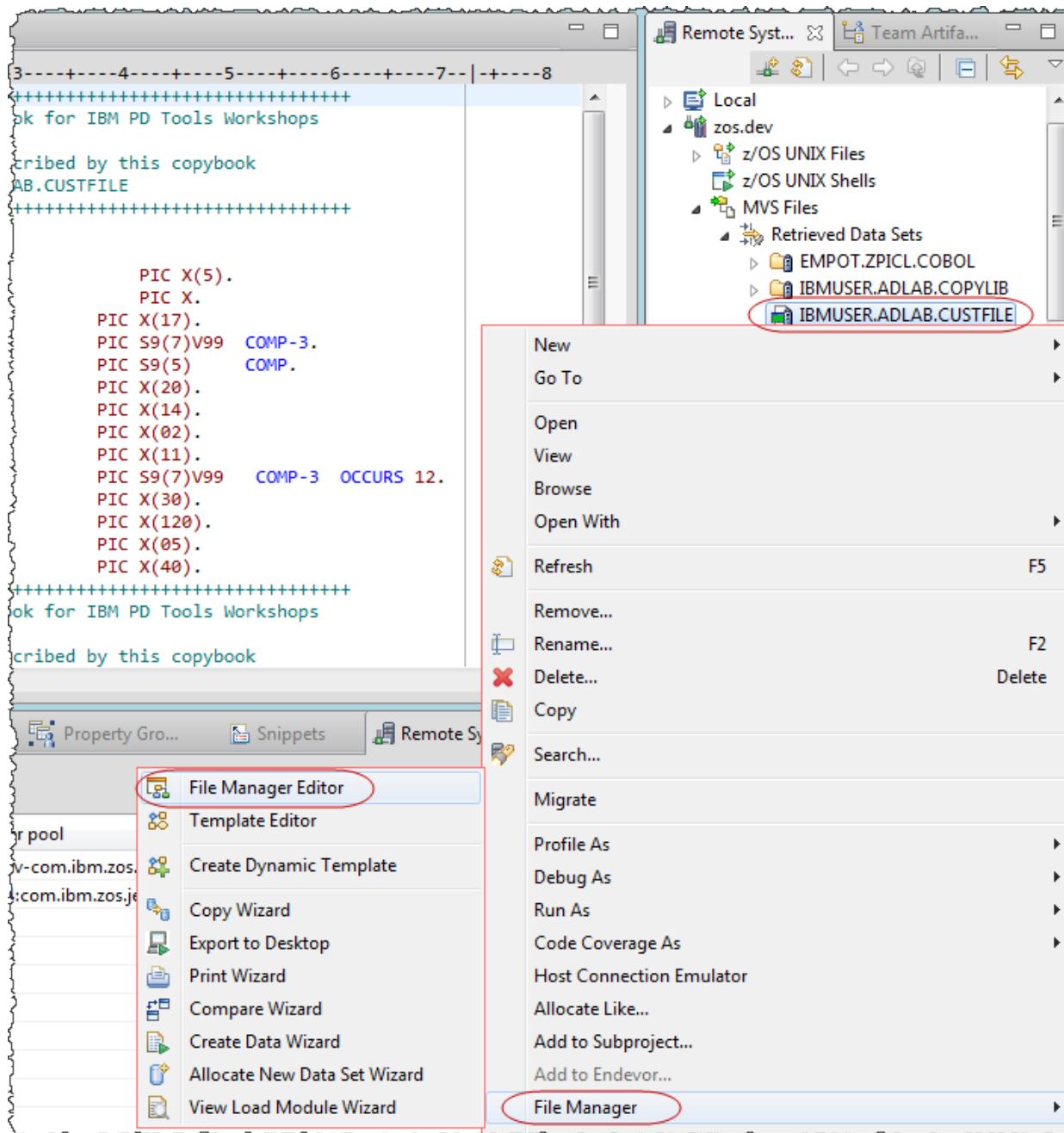
8.3.11 ►| Scroll down, select **IBMUSER.ADLAB.CUSTFILE** and click **OK**.



8.3.12 The data set will be under *Retrieved Data Sets*.

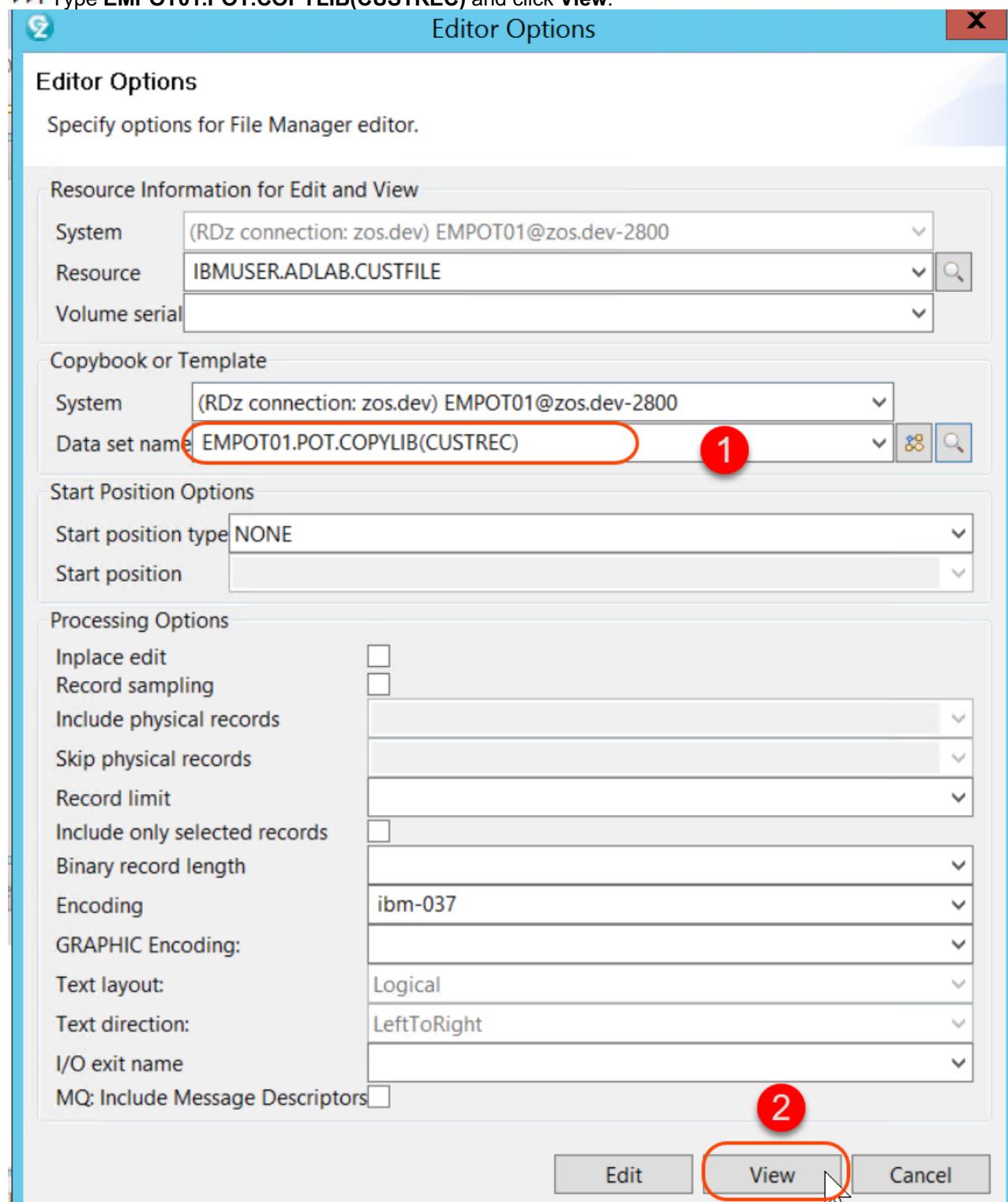


8.3.13 ►► Right click on **IBMUSER.ADLAB.CUSTFILE** and select **File Manager -> File Manager Editor**. You want to map this file against the COBOL COPYBOOK that you copied.



8.3.14 On *Data set name* you will use the PDS and member that you have copied the COPYBOOK.

► Type **EMPOT01.POT.COPYLIB(CUSTREC)** and click **View**.



8.3.15 The file record is displayed (type A) in formatted mode.

The screenshot shows the IBM i Navigator interface with two main windows:

- Left Window (File Record Display):** A table titled "RDz connection: zos.dev) EMPOT03@zos.dev-2800:IBMUSER.ADLAB.CUSTFILE". The table has columns: CUST-ID, REC-TYPE, NAME, ACCT-BALANCE, ORDERS-YTD, and ADC. The data is as follows:

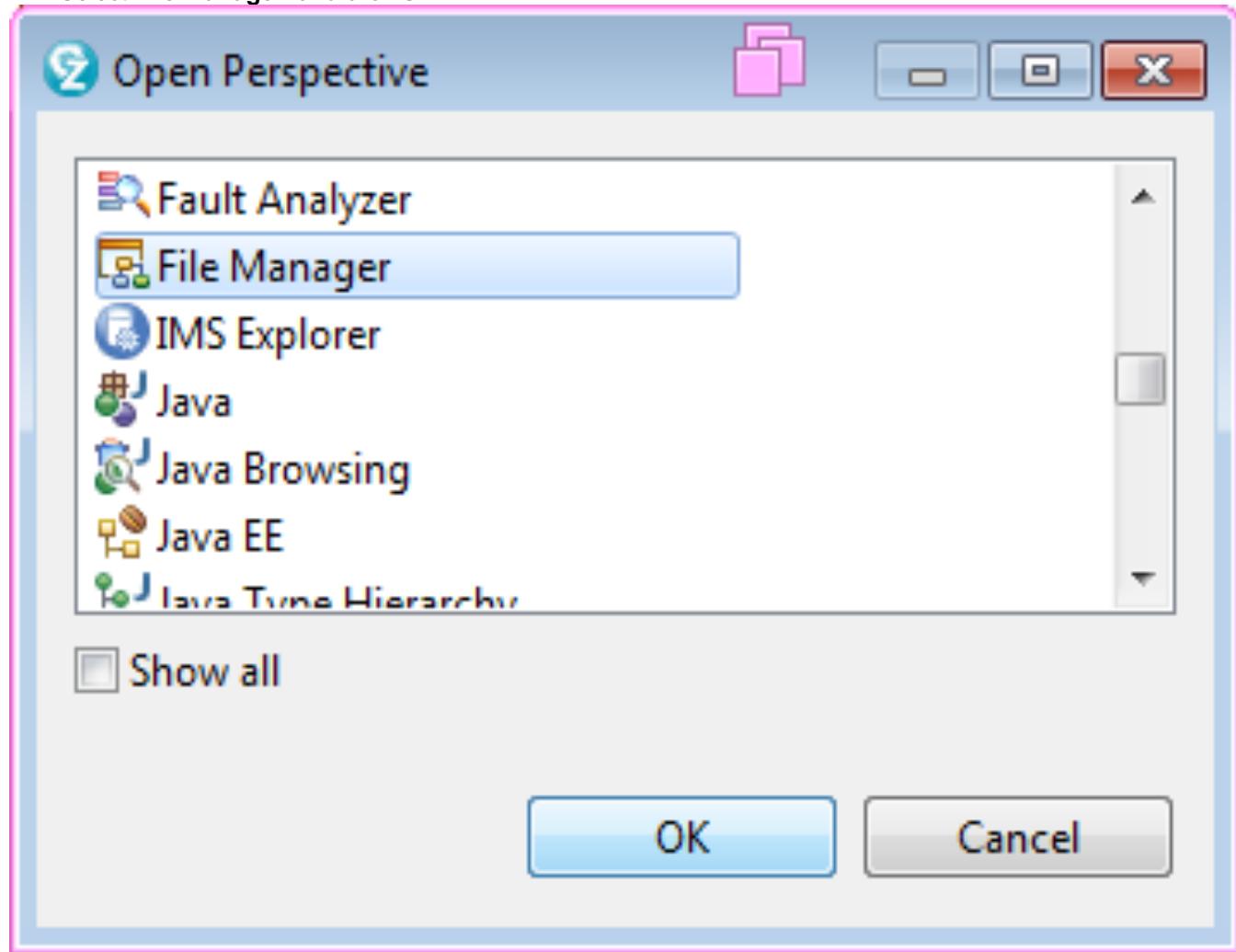
	CUST-ID	REC-TYPE	NAME	ACCT-BALANCE	ORDERS-YTD	ADC
1	03115	A	Graham, Holly	254.53	1	318
2	CONT...					
3	CONT...					
4	05580	A	Moore, Adeline	498.95	3	476
5	CONT...					
6	CONT...					
7	CONT...					
8	06075	A	Dubree, Dustin	192.98	1	922
9	06927	A	Buchs, Jillian	99.99	0	41
10	07008	A	Houston, Roger	296.97	1	411
11	07025	A	Marx, Audrey	450.51	2	90
12	CONT...					
13	CONT...					
14	CONT...					
15	11204	A	Ness, Luke	513.06	3	516
16	CONT...					
17	CONT...					

- Right Window (File Browser):** A tree view of the file system under "Remote Syst...". The root node is "REG10C.cbl", which contains several COBOL copybooks and a load module:

 - REG10C.cbl
 - COMAREA.cpy
 - CUSTREC.cpy
 - IGYTABLE.cpy
 - IGYTCRC.cpy
 - IGYTPRC.cpy
 - IGYTRANA.cpy
 - IGYTSRC.cpy
 - POTVSAM.cpy
 - EMPOT03.POT.DBRLIB
 - EMPOT03.POT.JCL
 - EMPOT03.POT.LISTING
 - EMPOT03.POT.LOAD
 - REG10B.exe
 - EMPOT03.POT.OBJ
 - EMPOT03.POT.PLI
 - EMPOT03.POT.PLI.LISTING
 - EMPOT03.POT.SP2.COBOL
 - EMPOT03.D032.T1232502.POT03DB
 - My Favorites
 - TSO Commands
 - JES
 - Retrieved Jobs
 - POT03CC.IOPR0001 (ACTIVE)

8.3.16 We can see more details using the File manager perspective.

- Select **Window > Perspective > Open Perspective > Other...**
- Select **File Manager** and click **OK**



- Click on the record #1.

Notice that the field **ACCT-BALANCE** that is **COMP-3** and **ORDERS-YTD** that is **COMP** are correctly displayed.

1

	CUST-ID	REC-TYPE	NAME	ACCT-BALANCE	ORDERS-YTD	ADDR	CITY	STATE	COUNTRY
1	03115	A	Graham, Holly	254.53		1 3100 Oaktree Ct	Raleigh	NC	USA
2 CO...									
3 CO...									
4	05580	A	Moore, Adeline		498.95	3 4700 S. Syracuse	Denver	CO	USA
5 CO...									
6 CO...									
7 CO...									
8	06075	A	Dubree, Dustin		192.98	1 9229 Delegate's Row	Indianapolis	IN	USA
9	06927	A	Buchs, Jillian		99.99	0 41 Avendale Drive	Carlisle	PA	USA
10	07008	A	Houston, Roger		296.97	1 4111 Northside PkWay	Atlanta	GA	USA
11	07025	A	Marx, Audrey		450.51	2 90 South Cascade	Boulder	CO	USA
12 C...									
13 C...									
14 C...									

Layout CUST-REC System (IDz connection: zos.dev) EMPOT01@zos.dev-2800 Template EMPOT01.POT.COPYLIB(CUSTREC)

FORMATTED

IBMUSER.ADLAB.CUSTFILE IMS Segment Editor Lookup Console

Field	Picture	Type	Start	Length	Data
CUST-ID	X(5)	AN	1	5	03115
REC-TYPE	X	AN	6	1	A
NAME	X(17)	AN	7	17	Graham, Holly
ACCT-B...	S9(7)V99	FD	24	5	254.53
ORDERS...	S9(5)	BI	29	4	1
ADDR	X(20)	AN	33	20	3100 Oaktree Ct
CITY	X(14)	AN	53	14	Raleigh
STATE	X(02)	AN	67	2	NC
COUNTTRY	Y(11)	AN	69	11	USA

View Mode Single Mode Insert Mode Insert

You may have to scroll down this view to see the formatted field names.

Also notice that there are two views of the data. The multiple record view appears at the top by default, and the single record view appears at the bottom and displays only one record at a time.

If you select (click) a record in the multiple-record view that record displays in the single record view.

8.3.16 ► Click on the record #2 and verify the new layout (CONTACT-REC)..

The screenshot shows the RDz interface with the CUSTREC.cpy editor open. The top part displays a list of customer records (CUST-ID, REC-TYPE, NAME, ACCT-BALANCE, ORDERS-YTD, ADDR). Row 2, which contains the data for 'Graham, Holly' (CUST-ID 03115), is highlighted and circled in red. A red arrow points from this circled row to the bottom part of the interface, which shows the 'FORMATTED' layout for CONTACT-REC. This layout table defines the fields, their types, and their positions in the record.

Field	Picture	Type	Start	Length	Data
CUST-ID	X(5)	AN	1	5	03115
REC-TYPE	X	AN	6	1	B
NAME	X(17)	AN	7	17	Graham, Holly
DESCRIPTION	X(10)	AN	24	10	Home Phone
CONTACT-INFO	X(20)	AN	34	20	112-555-6736

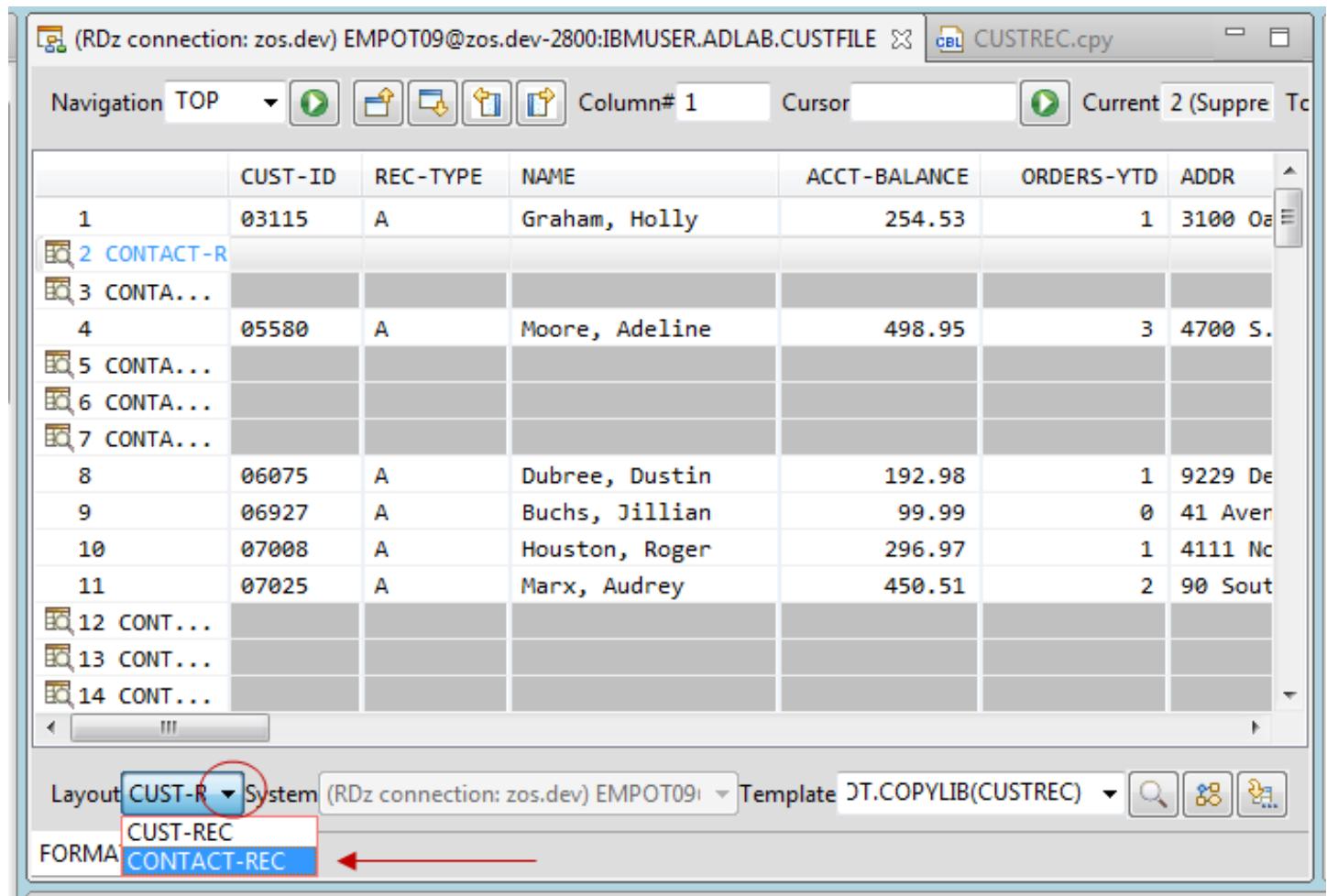
8.3.17 ► Click on the Layout pull-down in the editor and select CONTACT-REC. As you see, you can work with multiple layouts.

(RDz connection: zos.dev) EMPO109@zos.dev-2800:IBMUSER.ADLAB.CUSTFILE CBL CUSTREC.cpy

Navigation TOP Column# 1 Cursor Current 2 (Supre Tc

	CUST-ID	REC-TYPE	NAME	ACCT-BALANCE	ORDERS-YTD	ADDR	
1	03115	A	Graham, Holly	254.53	1	3100 Oa	
2 CONTACT-R							
3 CONTA...							
4	05580	A	Moore, Adeline	498.95	3	4700 S.	
5 CONTA...							
6 CONTA...							
7 CONTA...							
8	06075	A	Dubree, Dustin	192.98	1	9229 De	
9	06927	A	Buchs, Jillian	99.99	0	41 Aver	
10	07008	A	Houston, Roger	296.97	1	4111 Nc	
11	07025	A	Marx, Audrey	450.51	2	90 Sout	
12 CONTA...							
13 CONTA...							
14 CONTA...							

Layout CUST-R System (RDz connection: zos.dev) EMPO109 Template DT.COPYLIB(CUSTREC) FORMA CUST-REC CONTACT-REC



8.3.18 You now can see that records 2 and 3 are displayed (type B and C) in formatted mode.

The screenshot shows the DB2 Command Line Processor interface. The title bar indicates a connection to 'zos.dev' with user 'EMPOT09'. The window title is 'CUSTREC.cpy'. The toolbar includes navigation buttons (TOP, BOTTOM, PREV, NEXT, LAST, FIRST), a column selector ('Column# 1'), and a cursor indicator. The main area displays a table with the following columns: CUST-ID, REC-TYPE, NAME, DESCRIPTION, and CONTACT-INFO. The data is as follows:

	CUST-ID	REC-TYPE	NAME	DESCRIPTION	CONTACT-INFO
1 CUST-...					
2	03115	B	Graham, Holly	Home Phone	112-555-6736
3	03115	C	Graham, Holly	Cell Phone	135-555-2338
4 CUST-...					
5	05580	B	Moore, Adeline	Work Phone	161-555-4024
6	05580	C	Moore, Adeline	Home Phone	221-555-7598
7	05580	D	Moore, Adeline	Cell Phone	138-555-2410
8 CUST-...					
9 CUST-...					
10 CUST...					
11 CUST...					
12	07025	B	Marx, Audrey	Cell Phone	232-555-7244
13	07025	C	Marx, Audrey	Home Phone	240-555-4245
14	07025	D	Marx, Audrey	Work Phone	232-555-8753

The 'Layout' button in the toolbar is highlighted with a red oval. The status bar at the bottom shows 'FORMATTED'.

8.3.19 ►► Right click on the editor and verify that there are many options.

►► Click on **Switch Mode**

(RDz connection: zos.dev) EMPOT09@zos.dev-2800:IBMUSER.ADLAB.CUSTFILE CBL CUSTREC.cpy

Navigation TOP Cursor Current 7 To

Compare With

- Switch Mode Alt+M
- Page Up F7
- Page Down F8
- Page Left F10
- Page Right F11
- Copy Records Ctrl+Shift+C
- Find/Replace Ctrl+Shift+F
- Locate Column Ctrl+Shift+L
- Sort Records Alt+S
- Hex on/off Alt+H
- Show Options Ctrl+Alt+Shift+O
- Exclude Records Ctrl+Alt+Shift+X
- Reset Excludes Ctrl+Alt+Shift+R
- Save Records Ctrl+S
- SaveAs Records Ctrl+Alt+Shift+S
- Validate
- Software Analysis
- Team
- Replace With

	DESCRIPTION	CONTACT-INFO
1 CUST-	Home Phone	112-555-6736
2	Cell Phone	135-555-2338
3	Work Phone	161-555-4024
4 CUST-	Home Phone	221-555-7598
5	Cell Phone	138-555-2410
6	Cell Phone	232-555-7244
7	Home Phone	240-555-4245
8 CUST-	Work Phone	232-555-8753
9 CUST-		
10 CUST		
11 CUST		
12		
13		
14		

Template DT.COPYLIB(CUSTREC)

Snippets Remote ... Debug

Total 175

Part Length Data

8.3.20 The multiple-record view display is changed to character mode. In character mode, the data is not formatted according to the fields in layout.

The screenshot shows the IBM i Navigator interface with the following details:

- Title Bar:** (RDz connection: zos.dev) EMPOT09@zos.dev-2800:IBMUSER.ADLAB.CUSTFILE CBL CUSTREC.cpy
- Toolbar:** Navigation, TOP, and various icons for file operations.
- Table View:** A grid of data rows numbered 1 to 19. The data columns include names, addresses, phone numbers, and locations. Row 7 is highlighted with a blue background.
- Bottom Bar:**
 - Layout: CONTA (highlighted with a red oval)
 - System: (RDz connection: zos.dev) EMPOT09
 - Template: DT.COPYLIB(CUSTREC)
 - View Mode dropdown: CHARACTER (highlighted with a red oval)
 - Search and other navigation icons.

8.3.21 ► On the lower view notice the **View Mode** options.
Use the drop down and select **Dump Mode**:

Layout CONTACT-REC Current 7 Total 175

Field	Picture	Type	Start	Length	Data
CUST-ID	X(5)	AN	1	5	05580
REC-TYPE	X	AN	6	1	D
NAME	X(17)	AN	7	17	Moore, Adeline
DESCRIPTION	X(10)	AN	24	10	Cell Phone
CONTACT-INFO	X(20)	AN	34	20	138-555-2410
LAB-DATA-3	X(05)	AN	54	5	3456
LAB-DATA-4	X(05)	AN	59	5	7890

View Mode Single Mode Insert Mode Insert

Single Mode
Dump Mode
DB2 Single Mode
Structure Mode
Unstructured Mode

Layout CONTACT-REC Current 7 Total

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	0	1	2
+0	F0	F5	F5	F8	F0	C4	D4	96	96	99	85	6B	40	C	0	5	5
+10	93	89	95	85	40	40	40	C3	85	93	93	40	D7	8	1	i	n
+20	85	F1	F3	F8	60	F5	F5	60	F2	F4	F1	F0	4	e	1	3	
+30	40	40	40	40	40	F3	F4	F5	F6	40	F7	F8	F9	F			

There are lots of capabilities here.. You can try play a bit, but due to time constrains we will not cover all.

8.3.22 ►► Use Ctrl + Shift + F4 to close all opened editors.

8.4 Working with templates

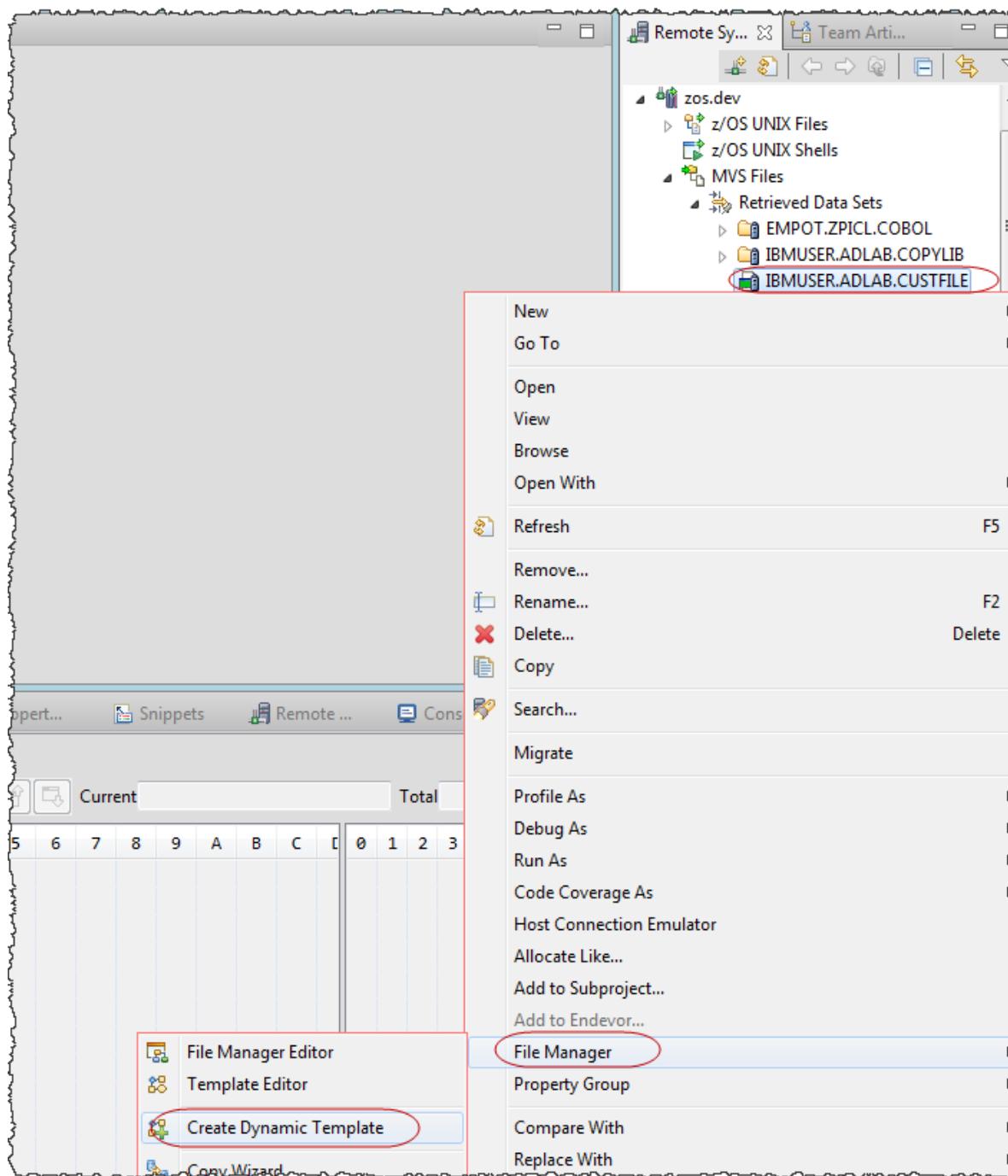
When you use the File Manager editor or viewer, you can specify a layout in the form of either a copybook or a template. If you specify a copybook, the editor/viewer can display records formatted according to the fields in the copybook. While a copybook defines the record layout, it cannot be used to select only a subset of records. A template, like a copybook, also has fields and defines the record layout. In addition, in a template you can specify:

- Record selection criteria (so that only the selected records will display)
- Field selection (so that only selected fields will display)
- Formatting of individual fields (for example, to always display a certain field in hexadecimal)
- And other formatting and data manipulation settings

You can use an existing copybook as the basis for a new template. All of the fields in the copybook are copied into the new template, and then you save the template. Templates are stored in PDS or library data sets. After you have saved a template, you can re-use it again whenever you need it. Templates can be used by other File Manager utilities other than just the editor and viewer.

For example, if you have a template that selects records, you can use it with the File Manager copy utility, and only the selected records will be copied.

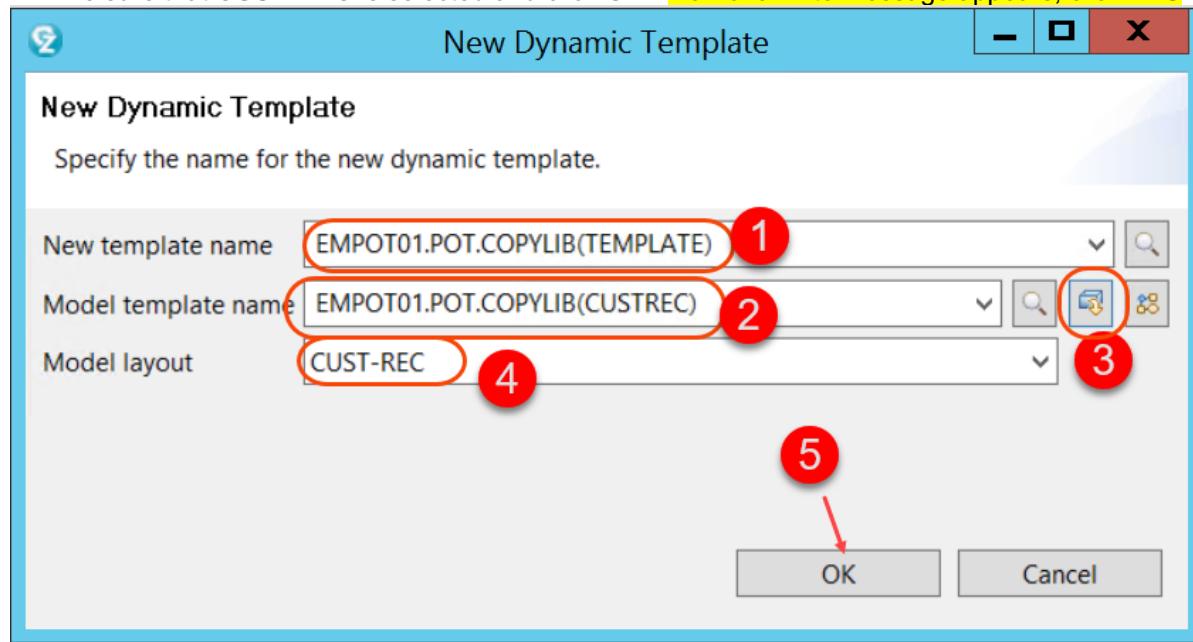
8.4.1  Go back to **z/OS Projects** Perspective and using *Remote Systems Explorer* view, right click **IBMUSER.ADLAB.CUSTFILE** file and select **File Manager -> Create Dynamic Template**



8.4.2 ► As New template name type **EMPOT01.POT.COPYLIB(TEMPLATE)**

► For Model template name type **EMPOT01.POT.COPYLIB(CUSTREC)** and click on icon  (Load model template).

► Be sure that **CUST-REC** is selected and click **OK**. If an overwrite message appears, click **YES**.

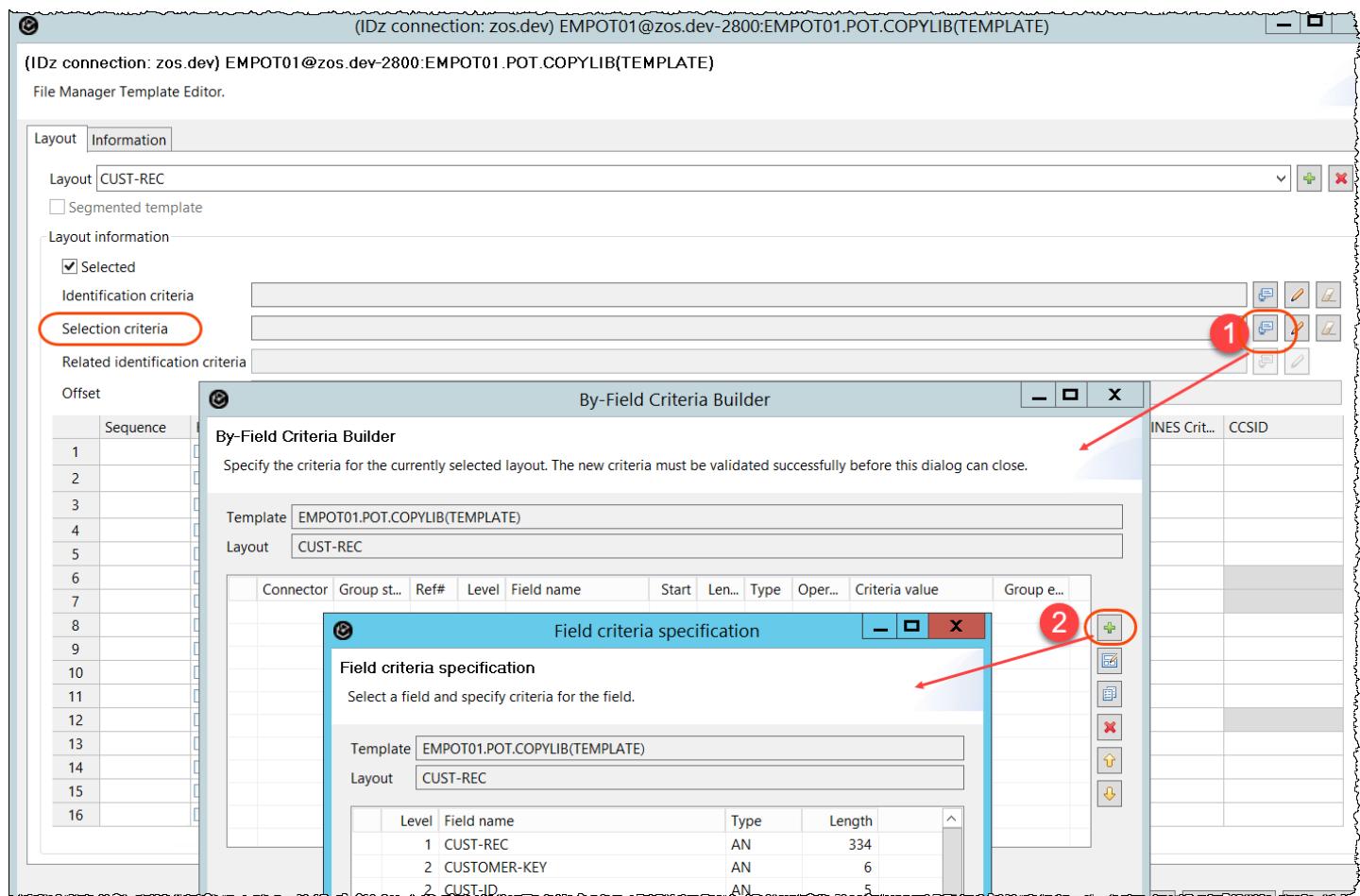


8.4.3 On the *File Manager Template Editor* you will add a selection criteria.

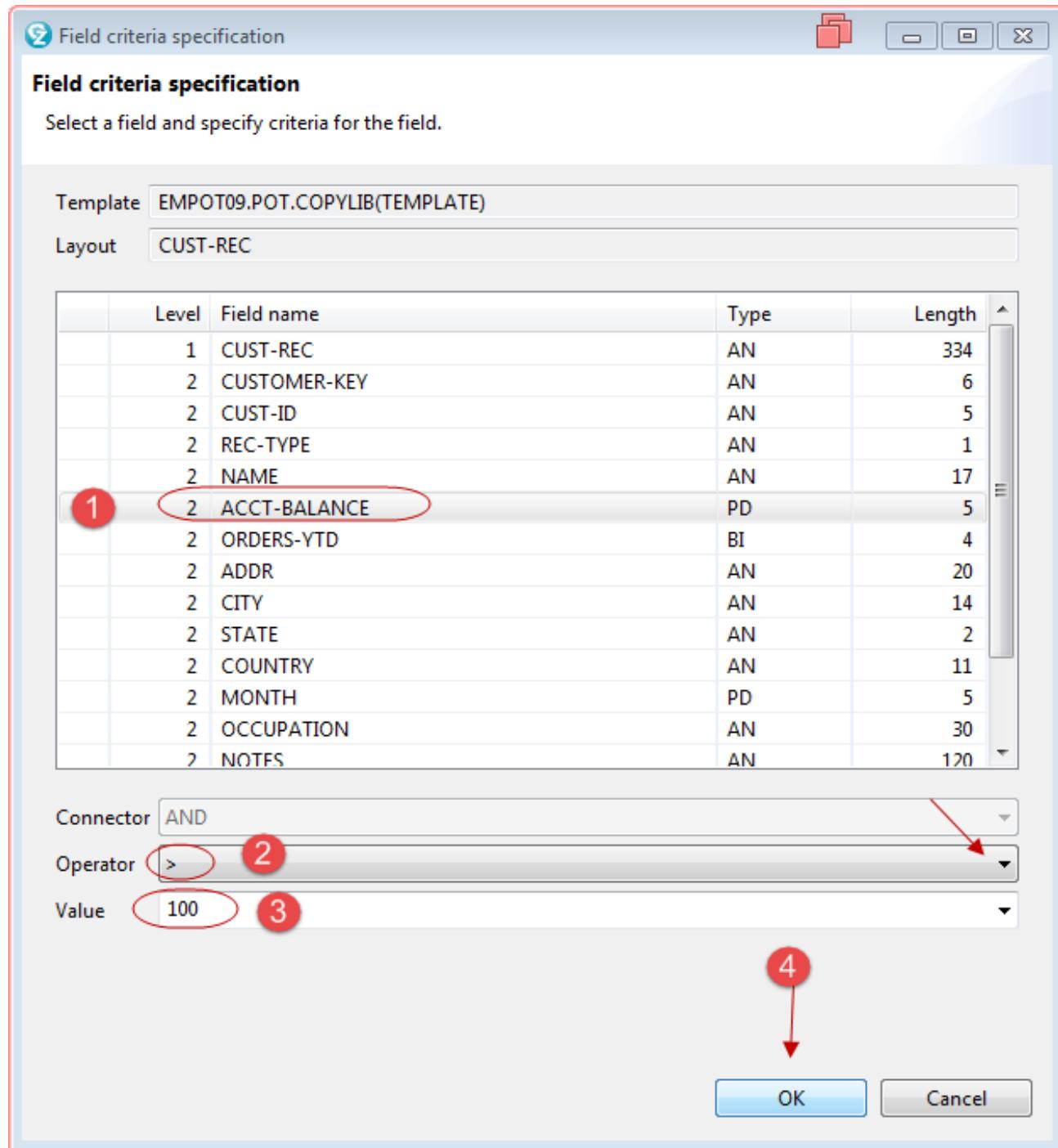
You will select all customers that have the ACCT-BALANCE **bigger than 100** and we want to display only **CUST-ID**, **NAME** and **ACCT-BALANCE**.

► 1 On **Selection criteria** click on the icon .

► 2 Using **By-Field Criteria Builder** click on the icon .

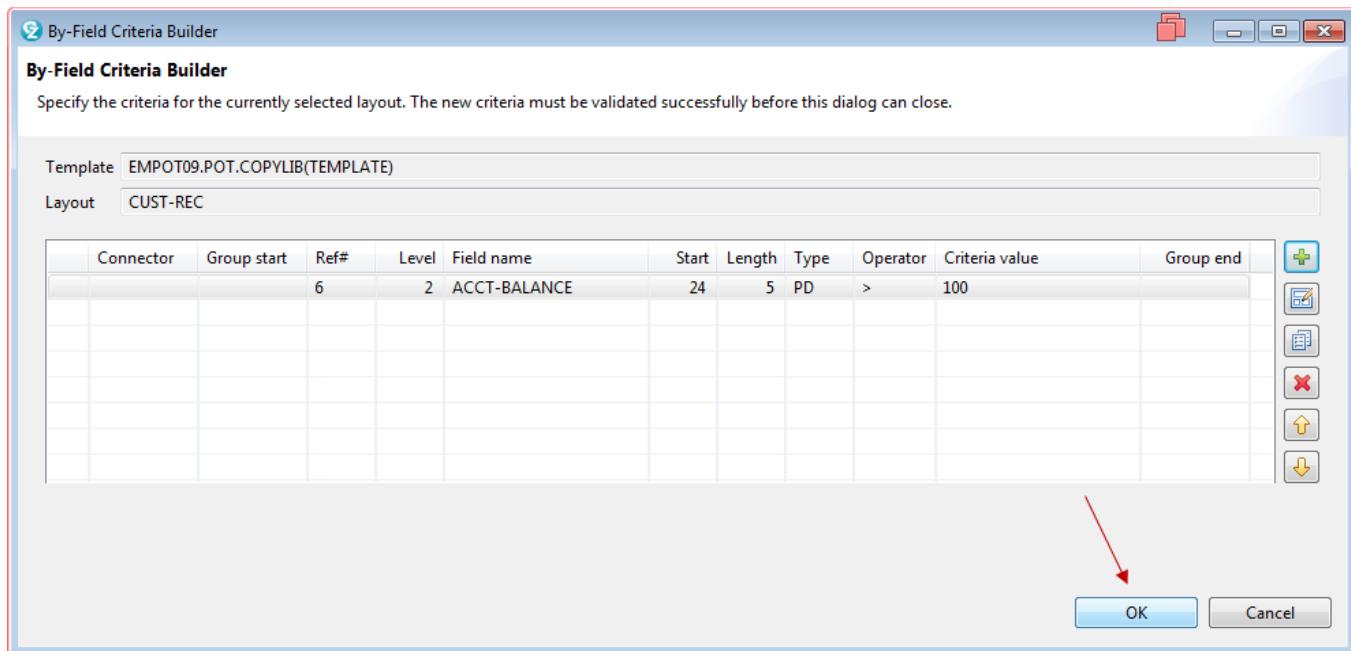


- 8.4.4 ► ① On the *Field criteria specification* select **ACCT-BALANCE**
 ► ② On the *Operator* using the drop down select **>** and ③ on the *Value* type **100** and ④ click **OK**.



8.4.5 The result is shown below..

- Click **OK**

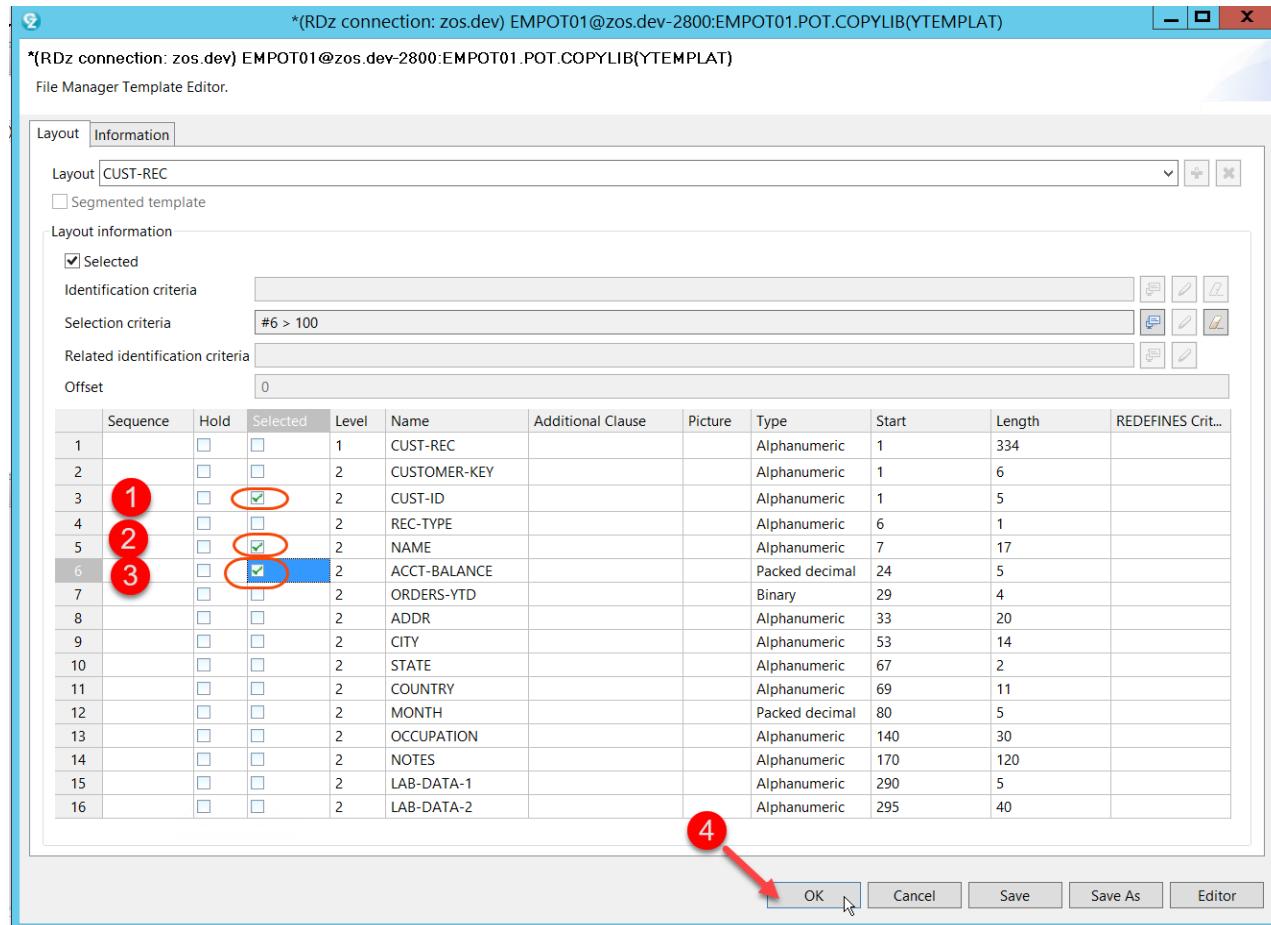


8.4.6 ► Using the column **Selected** double-click on the boxes

① CUST-ID, ② NAME and ③ ACCT-BALANCE.

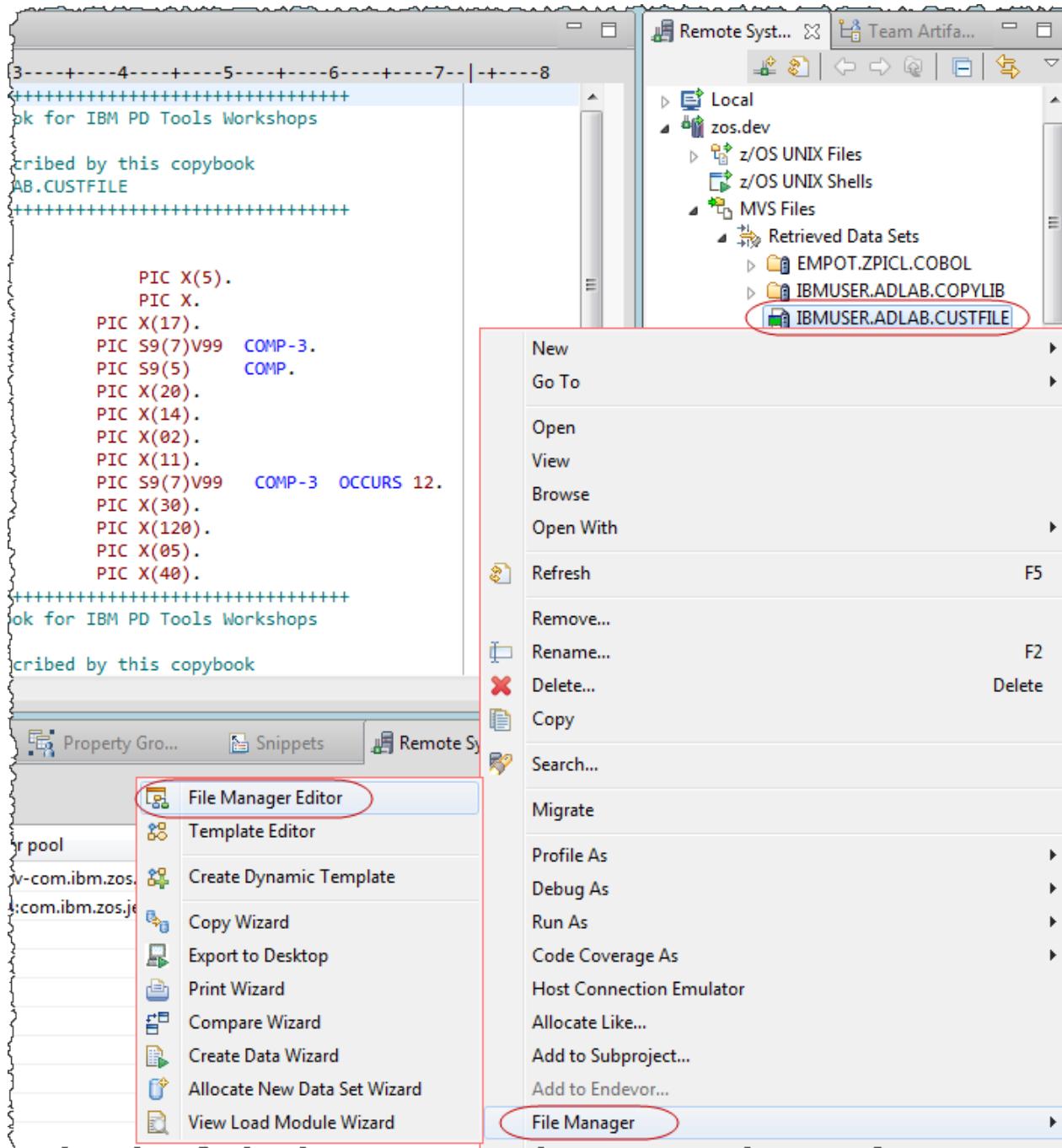
► Click OK.

The template is created. At any time, you can edit and modify it.

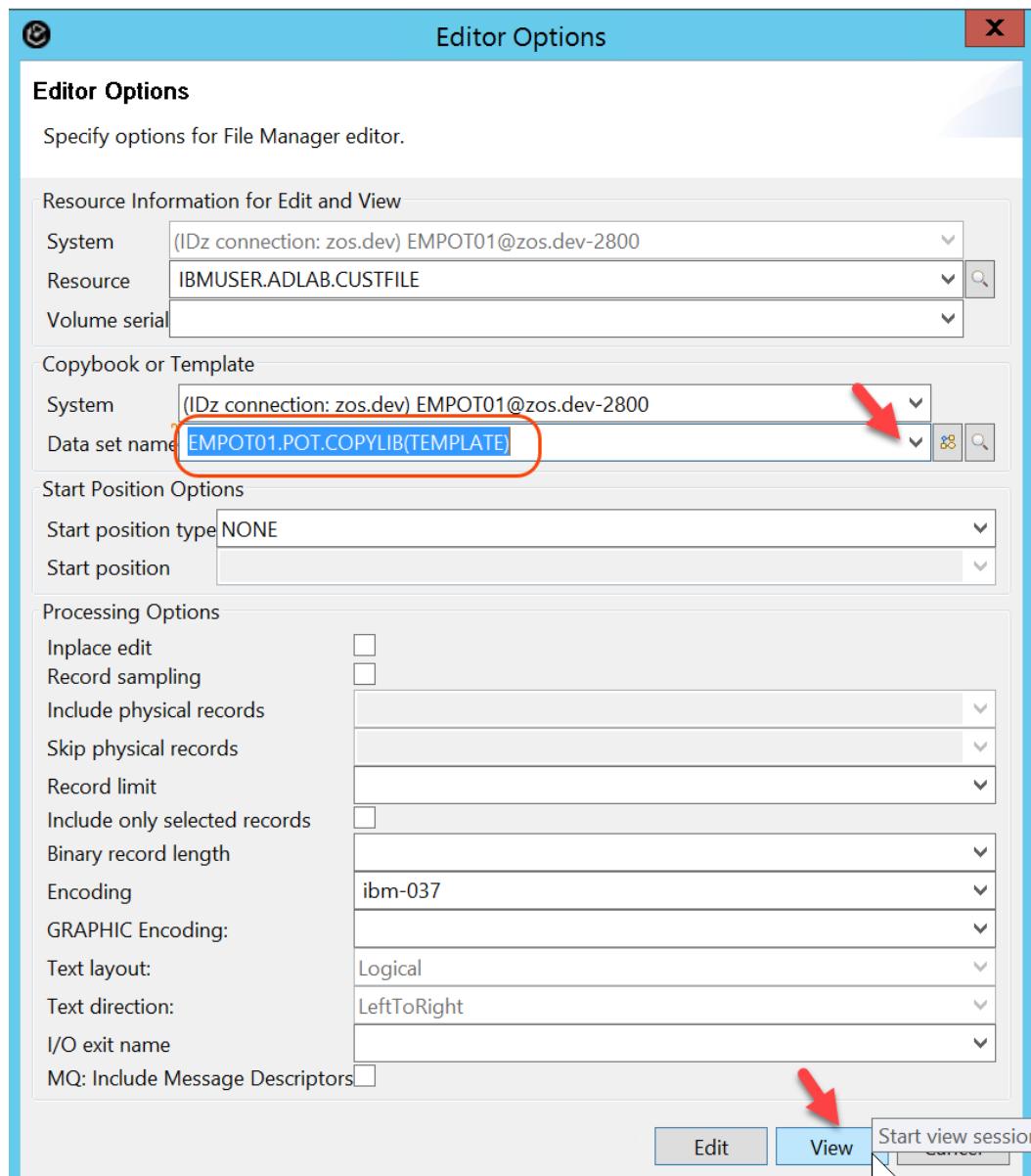


8.4.7 Now that the template is created you can use against the data set.

► Right click on **IBMUSER.ADLAB.CUSTFILE** and select **File Manager -> File Manager Editor**.



8.4.8 ► On *Data set name* type the Template that you have created.
Type **EMPOT01.POT.COPYLIB(TEMPLATE)** (or use the drop down) and click **View**.



8.4.9 You should have the results below..

Notice that only 3 fields are displayed and all have account balance higher than 100.

The screenshot shows the IBM iSeries Navigator interface with two editors open:

- CUSTFILE Editor:** Displays a list of customer records with the following data:

	CUST-ID	NAME	ACCT-BALANCE
1	03115	Graham, Holly	25453
2	05580	Moore, Adeline	49895
3	06075	Dubree, Dustin	19298
4	06927	Buchs, Jillian	9999
5	07008	Houston, Roger	29697
6	07025	Marx, Audrey	45051
7			
8			
9			
10			
11			
12			
13			
14			
- CUST-REC Editor:** Displays a detailed view of the first record with the following data:

Field	Picture	Type	Start	Length	Data
CUST-ID		AN	1	5	03115
NAME		AN	7	17	Graham, Holly
ACCT-BALANCE		PD	24	5	25453

File manager had much more capabilities. But at least you had an idea how it works with z/OS datasets

8.4.10 ➡ Use **Ctrl + Shift + F4** to close all editors. Or just click on the of each opened editor

Congratulations! You have completed the Lab 1.

LAB 2A – (OPTIONAL) Using IBM zUnit to Unit Test a COBOL CICS/DB2 program using Remote assets (60 minutes)

Updated May 28, 2021 by Regi – Created by Regi/Wilbert

Acknowledgments:

We would like to thank the following for their assistance:
Suman Gopinath & Nathan Cassata.

This lab will take you through the steps of using the automated unit testing ([zUnit](#)) capabilities of [IBM Developer for z](#) (IDz) to create a unit test case for a COBOL CICS/DB2 program. This enables the testing of just a single program within a CICS transaction without the need to run the whole transaction. This is done by stubbing out CICS calls, enabling the program to be tested without a CICS environment and without the need to deploy to CICS after a code change. This enables a developer to test early without impacting other developers that share the same CICS environment.

In this lab you will record interaction with a COBOL CICS/DB2 program (program under test) and import the recorded data into the unit test case generation process. You will build and run the unit test, make a change to the CICS COBOL program, and rerun the unit test.

Overview of development tasks

To complete this tutorial, you will perform the following tasks:

- 1. Get familiar with the application using the 3270 terminal**
→ You will start a 3270 emulation and execute a transaction named **HCAZ** to become familiar with the Application that you will be recording.
- 2. Import a z/OS project**
→ You will import a z/OS project with the required resources added to the project.
- 3. Record interaction with the application.**
→ You will record an interaction with the COBOL CICS/DB2 program.
- 4. Generate, build and run the unit test**
→ You will compile and link-edit the generated unit test program, followed by running the unit test.
- 5. Modify the program and rerun the unit test**
→ You will modify the program under test, rerun the unit test, and observe the failure of the test case.
- 6. Run the unit test from a batch JCL .**
→ You will run the unit test from a Batch JCL and observe a similar test case result.

Section 1. Get familiar with the application using the 3270 terminal

You will start a 3270 emulation and execute a transaction named **HCAZ** to become familiar with the Application that you will be interacting with.

1.1 Connect to z/OS and emulate a CICS 3270 terminal

You will use IDz to emulate a 3270 terminal to run the CICS transaction.

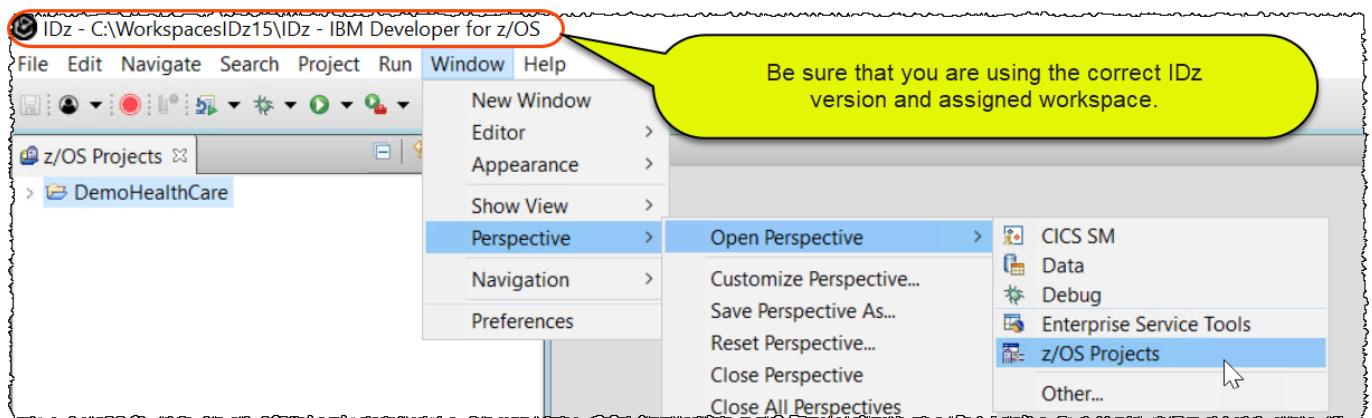
1.1.1 Start *IBM Developer for z Systems version 15* if it is not already started

- Using the desktop double click on **IDz V15** icon.
- Verify that the message indicates that it is Version **15.0.1**

IMPORTANT -> This icon will start an eclipse workspace that has already some definitions required for this lab. PLEASE DO NOT start IDz using other way than this icon.



1.1.2 ► Open the **z/OS Projects** perspective by selecting **Window > Perspective > Open Perspective > z/OS Projects**



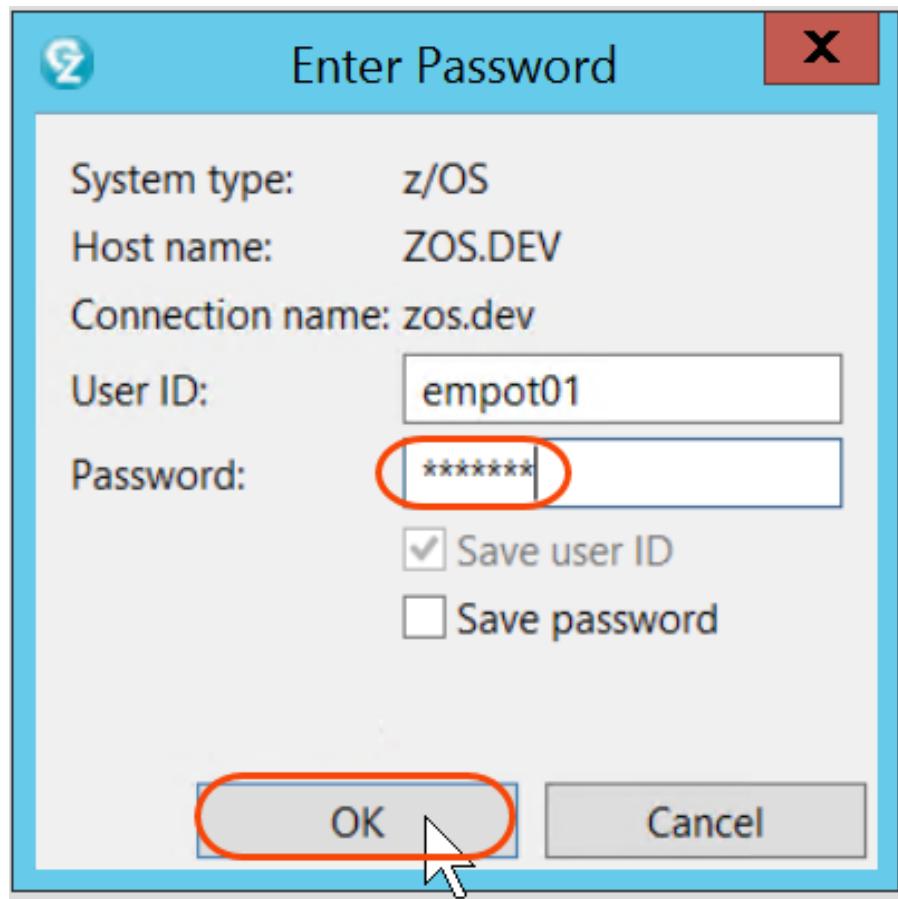
1.1.3 You will use userid **empot01**.

If you are connected as empot01, jump to step 1.1.5 Otherwise.

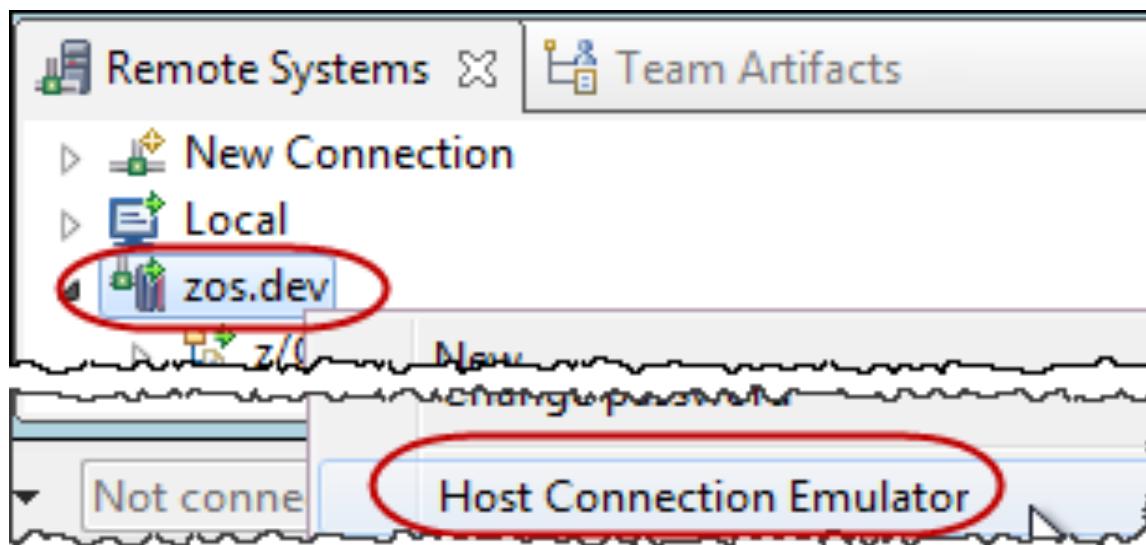
- Using *Remote Systems* view, right click on **zos.dev** and select **Connect**

(You also may need to disconnect first)

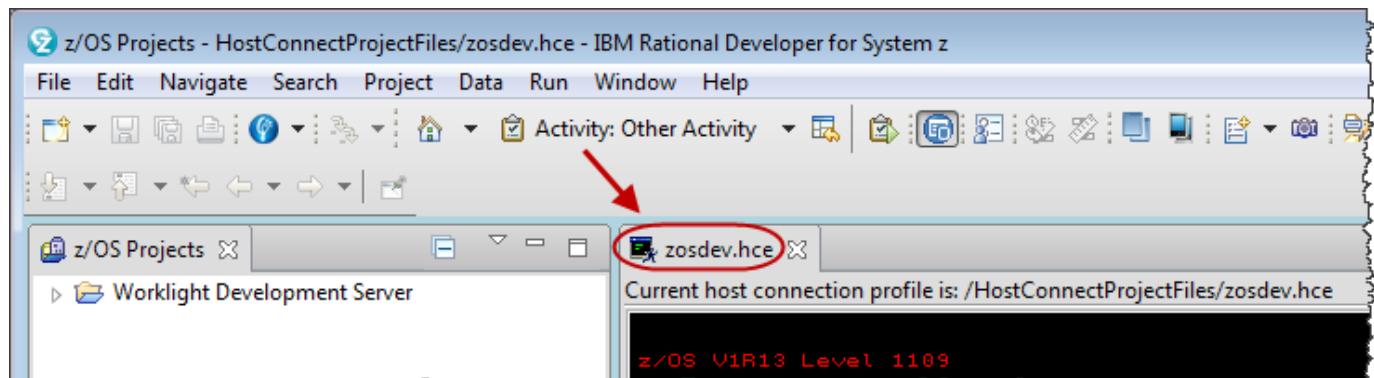
- 1.1.4 ► Type **empot01**as userid and **empot01** as password.
The userid and password can be any case; don't worry about having it in UPPER case.
Click **OK** to connect to z/OS.



- 1.1.5 ► Using the *Remote Systems* view, right click on **zos.dev** and select **Host Connection Emulator**.



- 1.1.6 ► Since you will need more space, **double-click** on the **zos.dev.hce** title



- 1.1.7 ► Type **I cicsts54**. (where "I" is the lower case of letter "L") and press **Enter key**.

```
====> Examples: "L TSO", "L CICSTS52", "L CICSTS53", "L IMS13", "L IMS14"
I cicsts54
```

The screenshot shows a terminal window with a black background. At the top, it says '====> Examples: "L TSO", "L CICSTS52", "L CICSTS53", "L IMS13", "L IMS14"'. Below that, the command 'I cicsts54' is typed in green, enclosed in a yellow oval. A yellow arrow points to the 'I cicsts54' text. The bottom of the window shows some blue and white text that is mostly illegible.

- 1.1.8 ► Logon using your z/OS user id and password (**empot01**) and press **Enter**.



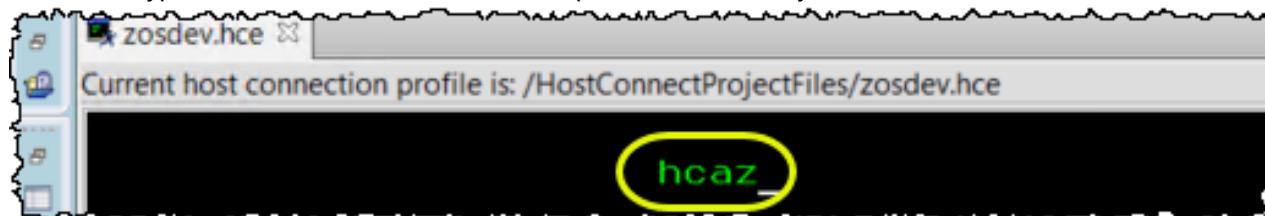
- 1.1.9 The sign-on message is displayed



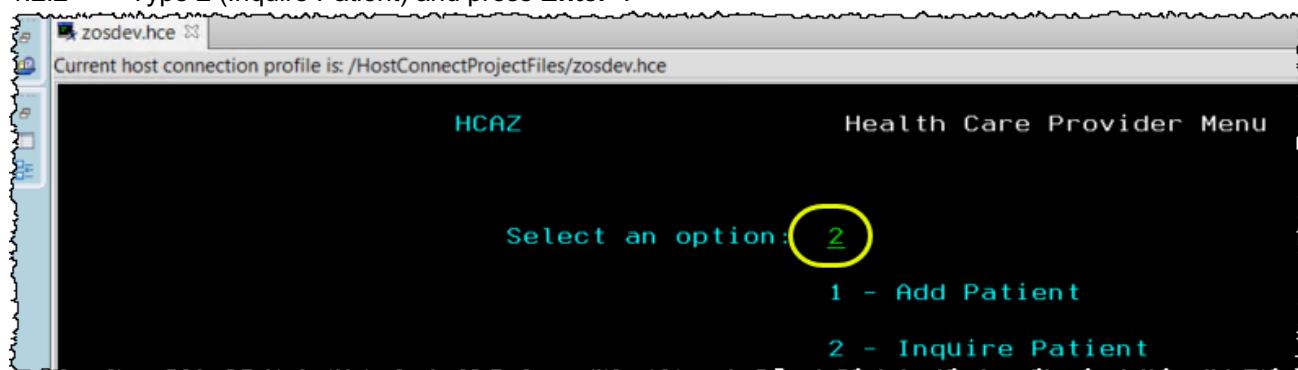
1.2 Run CICS transaction HCAZ

You should now be in the z/OS CICS region named C/CSTS54. This is the CICS instance where you will make the recording of your interaction.

- 1.2.1 ► Type the CICS transaction **hcaz** and press the **Enter** key.



- 1.2.2 ► Type 2 (Inquire Patient) and press **Enter** .



1.2.3 ► Type 1 for Patient ID and press Enter. The program will read the patient from a DB2 table and display the customer details. Remember the patient first name: **Ralph**

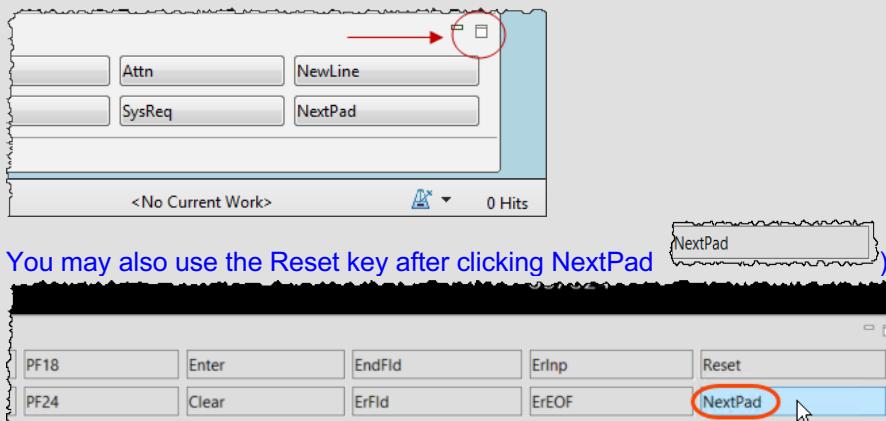
```
HCP1                                         Inquire Patient Information

Patient ID          0000000001
Name :First          Ralph
                   :Last   DALmeida
DOB                1980-07-11      (yyyy-mm-dd)
Address             34 Main Street
City               Toronto
Postcode            M5H 1T1
Phone: Mobile       077-123-9987
Email Addr          RalphD@ibm.com
Insurance Card      9627811234
User ID             ralphd
```

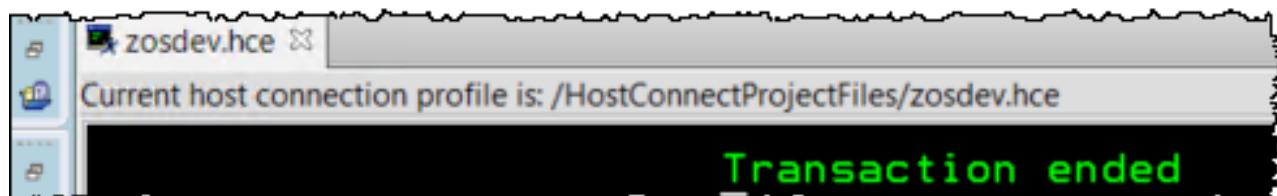
IF you need to use functions like Clear or Reset

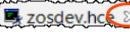
If you need to use the **clear** function use the key **Esc**.

Also you may look in the right lower corner, select this icon  . This will display possible keys, including the clear button.

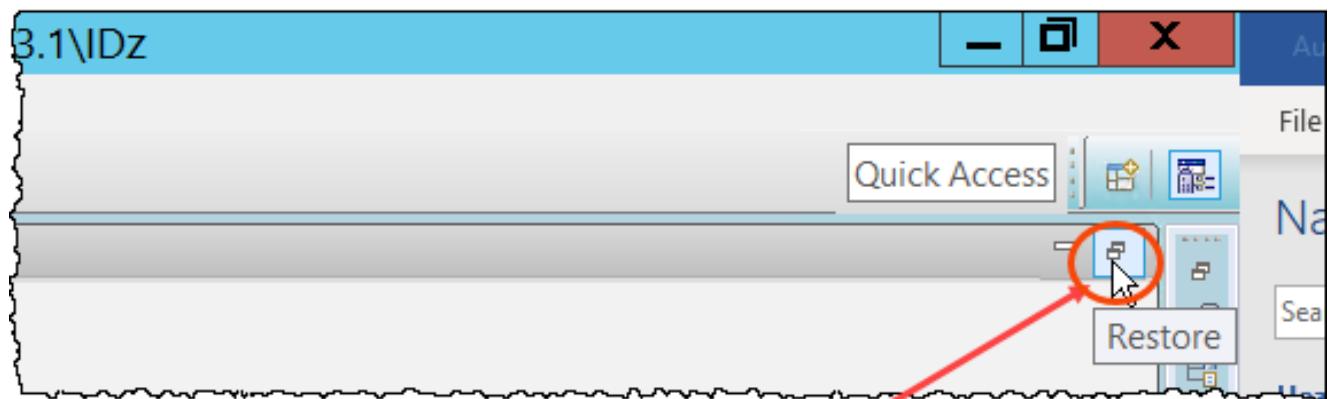


- 1.2.4  Press **F3** to end the application.



- 1.2.5  Close the terminal emulation clicking on  →  . Or pressing **CTRL + Shift + F4**.

- 1.2.6  You may need to restore the **z/OS Projects** perspective by clicking on the icon  on top right:





What have you done so far?

You emulated a 3270 terminal using IBM Developer for System z.

You also executed the CICS transaction **HCAZ** and verified a simple interaction with the Hospital application. The objective here was to show the code that you will update.

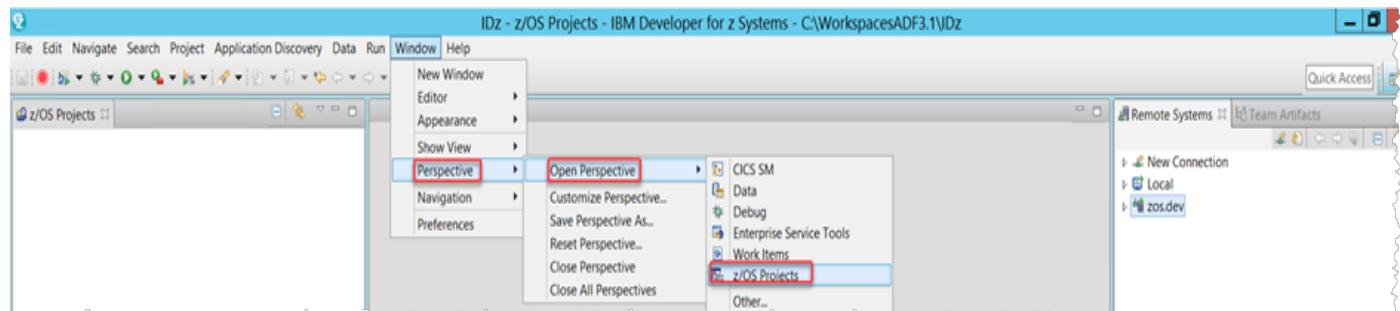
Section 2 – Import a z/OS project

You will import a z/OS Project that contains the resources needed to generate a unit test program for a COBOL CICS application.

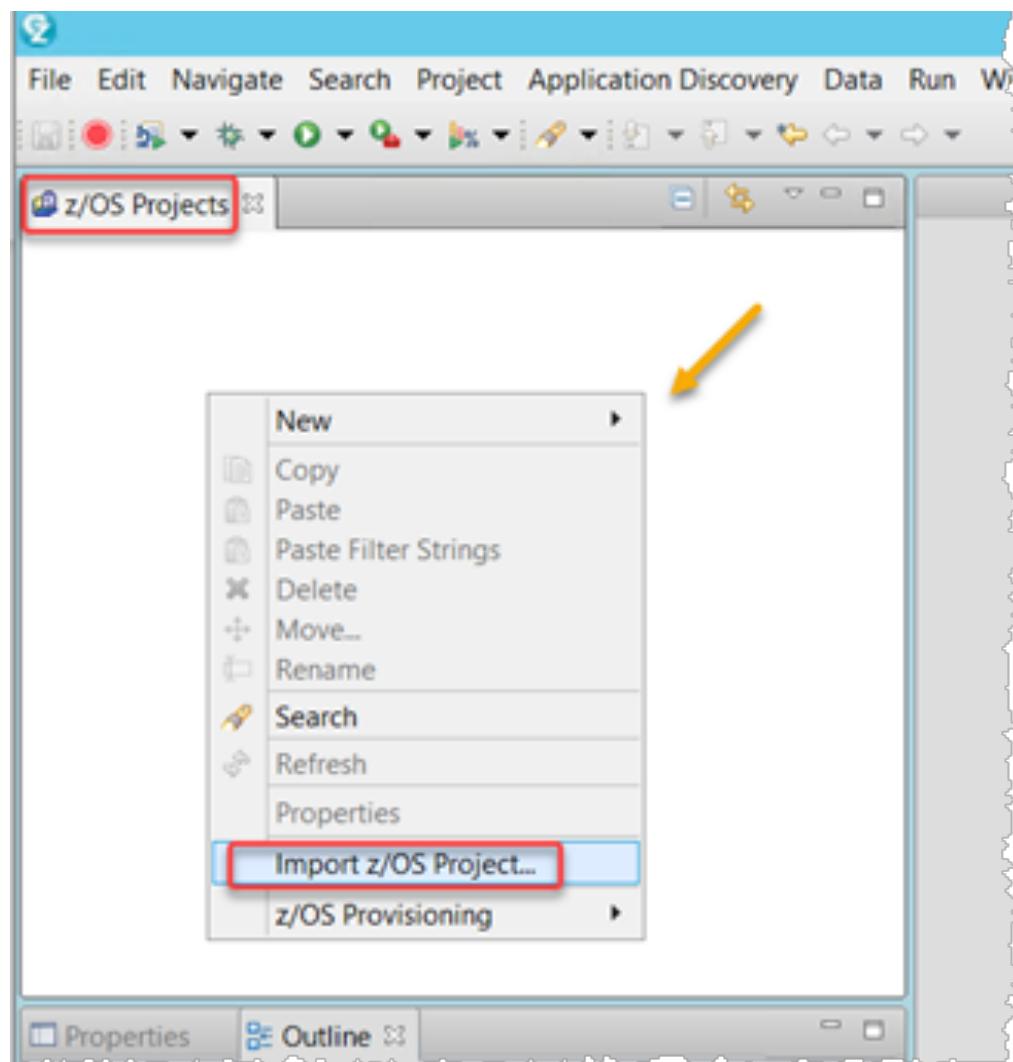
2.1 Importing the LAB6B z/OS Project

The Hospital Application used in this lab has its source code in a *PDS* that's been added to a z/OS Project. You must be connected to the *zos.dev* system (see step 1.1).

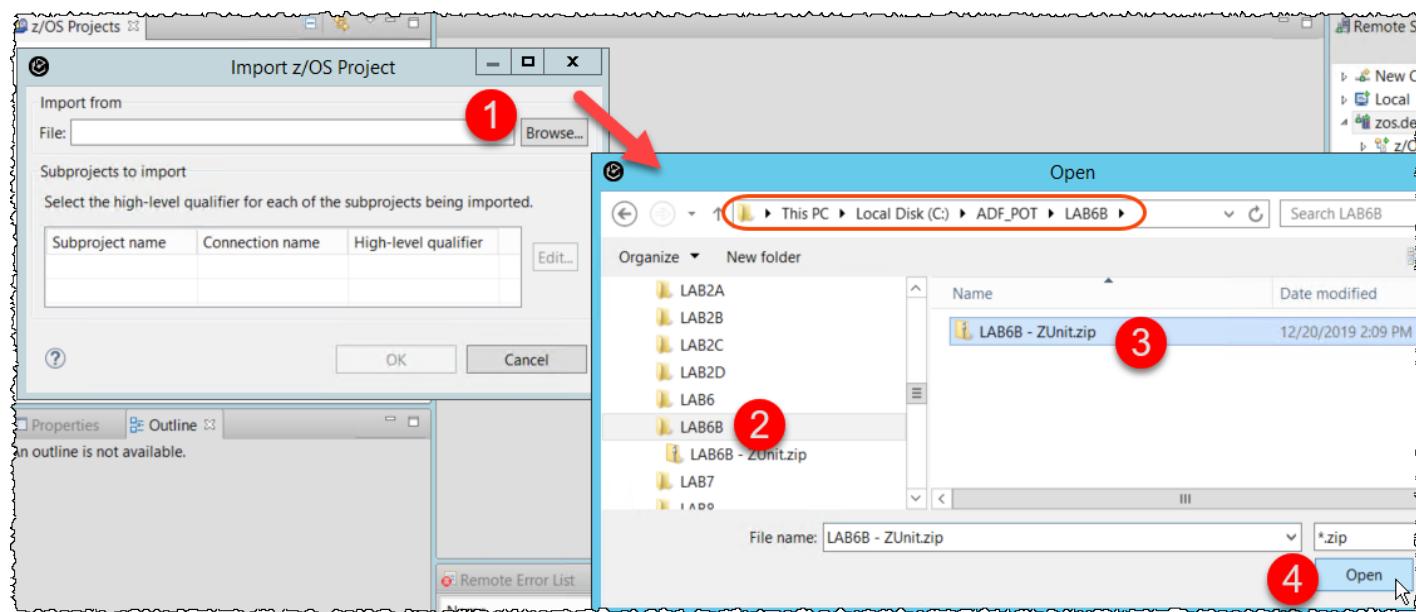
2.1.1 ► If not already in the z/OS Projects perspective, open the **z/OS Projects** perspective by selecting **Window > Perspective > Open Perspective > z/OS Projects**



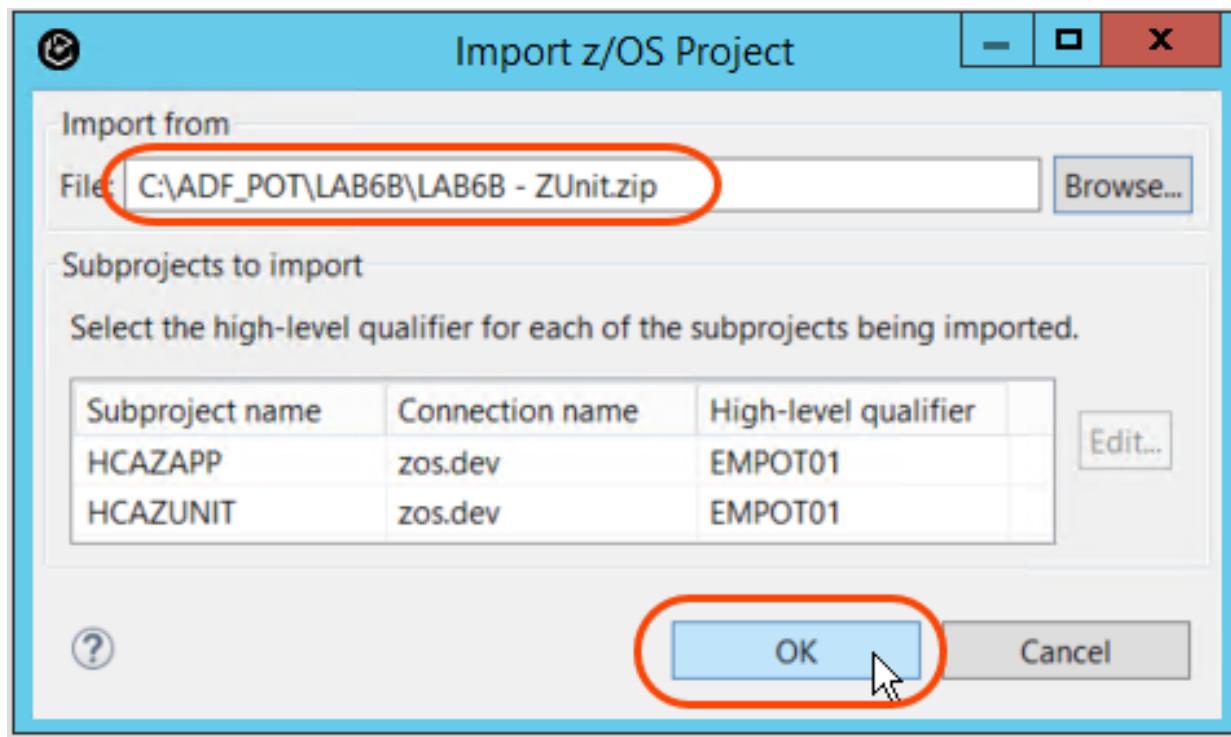
2.1.2 ► Using the **z/OS Projects** view on top left, position the cursor anywhere in the view. Right mouse click and select the **Import z/OS Project** action.



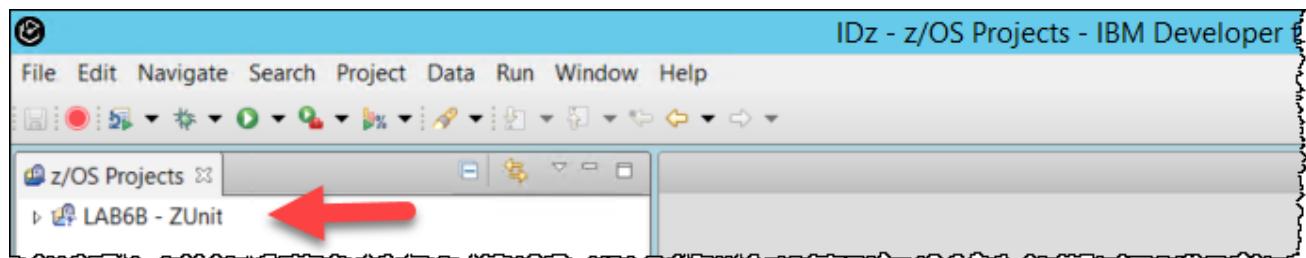
- 2.1.3 ► In the Import z/OS Project dialog, click the **Browse** button, navigate to C:\ADF_POT\LAB6B and select **LAB6B - ZUnit.zip**. Click **Open**.



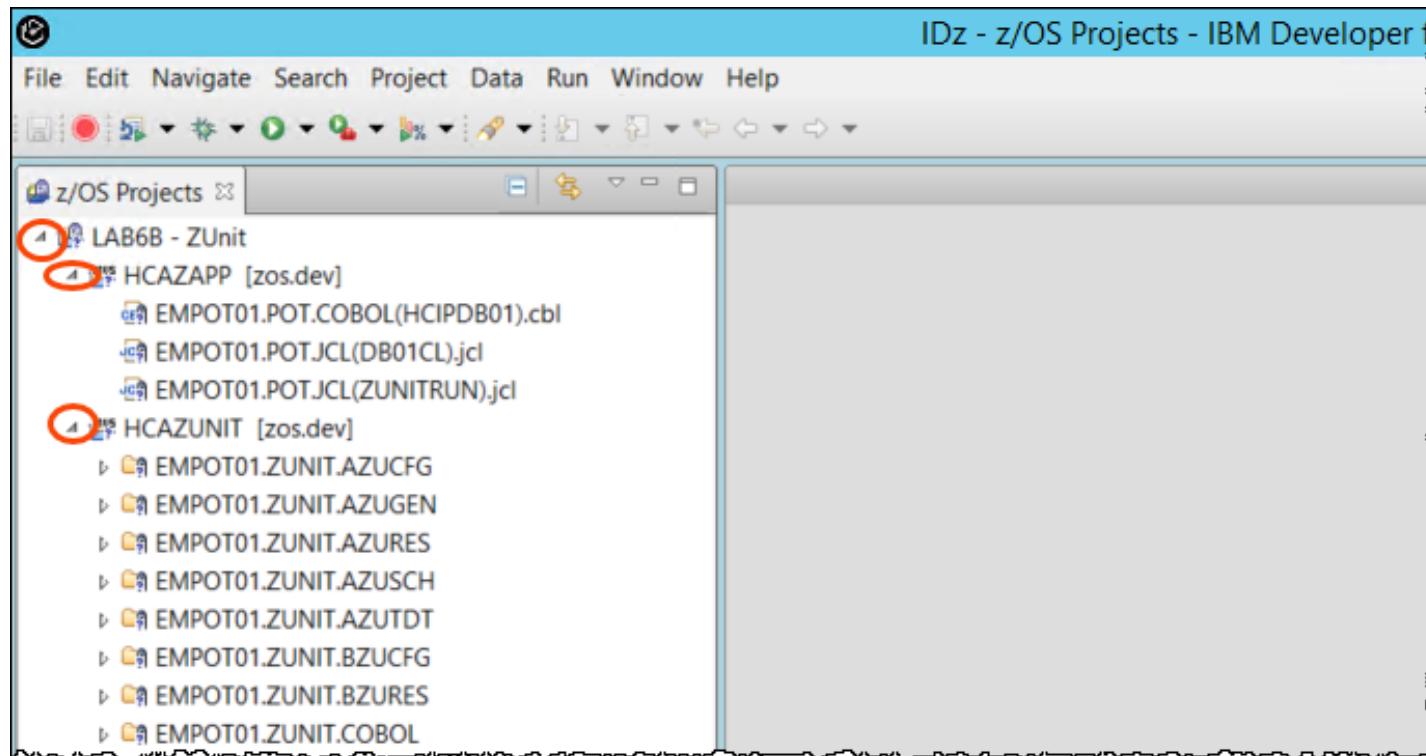
- 2.1.4 ► In the dialog that opens, click **OK**.



You should see something like this in your z/OS Projects view:



2.1.5 ► Expand the nodes by left clicking on the icon  as shown below:.



Section 3 – Record data interaction using the CICS application.

Using IDz you will record the interaction with the COBOL/DB2 program that reads a patient from a DB2 table.

3.1 Understanding the COBOL program that reads from DB2 table

The COBOL code that reads the patient from the DB2 tables is the program **HCIPDB01**

3.1.1 ► Using z/OS Projects view double click **EMPOT01.POT.COBOL(HCIPDB01).cbl** under **HCAZAPP**.

This is the program that you will update later on.

IDz - z/OS Projects - zos.dev/EMPOT01/EMPOT01.POT.COBOL/Hcipdb01.cbl - IBM Developer for z/OS - C:\Works

File Edit Source Refactor Navigate Search Project Data Run Window Help

z/OS Projects HCIPDB01.cbl

LAB6B - ZUnit
HCAZAPP [zos.dev]
EMPOT01.POT.COBOL(Hcipdb01.cbl) (highlighted with a red arrow)

HCAZUNIT [zos.dev]
EMPOT01.ZUNIT.AZUCFG
EMPOT01.ZUNIT.AZUGEN
EMPOT01.ZUNIT.AZURES
EMPOT01.ZUNIT.AZUSCH
EMPOT01.ZUNIT.AZUTDT
EMPOT01.ZUNIT.BZUCFG
EMPOT01.ZUNIT.BZURES
EMPOT01.ZUNIT.COBOL

Properties Outline

PROGRAM: HCIPDB01
IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
DATA DIVISION.

```

-----+---A-1-B---+---2---+---3---+---4---+---5---+---6---+---7---+
1 | ****
2 | * Used on CICS trx HCAZ
3 | * invoked when selecting option 2 - Inquire Patient
4 |
5 | * Look for $bug to introduce a bug
6 | ****
7 | IDENTIFICATION DIVISION.
8 | PROGRAM-ID. HCIPDB01.
9 | ENVIRONMENT DIVISION.
10 | CONFIGURATION SECTION.
11 | DATA DIVISION.
12 | WORKING-STORAGE SECTION.
13 |
14 | * Common definitions
15 |
16 | * Run time (debug) information for this invocation
17 | 01 WS-HEADER.
18 |     03 WS-EYECATCHER          PIC X(16)
19 |                         VALUE 'HCIPDB01-----WS'.
20 |     03 WS-TRANSID            PIC X(4).
21 |     03 WS-TERMINAL          PIC X(4).
22 |     03 WS-TASKNUM           PIC 9(7).
23 |     03 WS-FILLER             PIC X.

```

- 3.1.2 ► Using the Outline view on left expand **PROCEDURE DIVISION** and click on **GET-PATIENT-INFO**. This is where the patient data is read from the DB2 table. Later on you will introduce a bug in this program..

The screenshot shows the IBM Rational Application Developer interface. On the left, the 'z/OS Projects' view displays a project structure under 'LAB6B - ZUnit'. The 'Outline' view shows the structure of the COBOL program HCIPDB01. A red arrow points from the 'Outline' view to the 'GET-PATIENT-INFO' procedure in the code editor on the right. The code editor shows the following COBOL code:

```

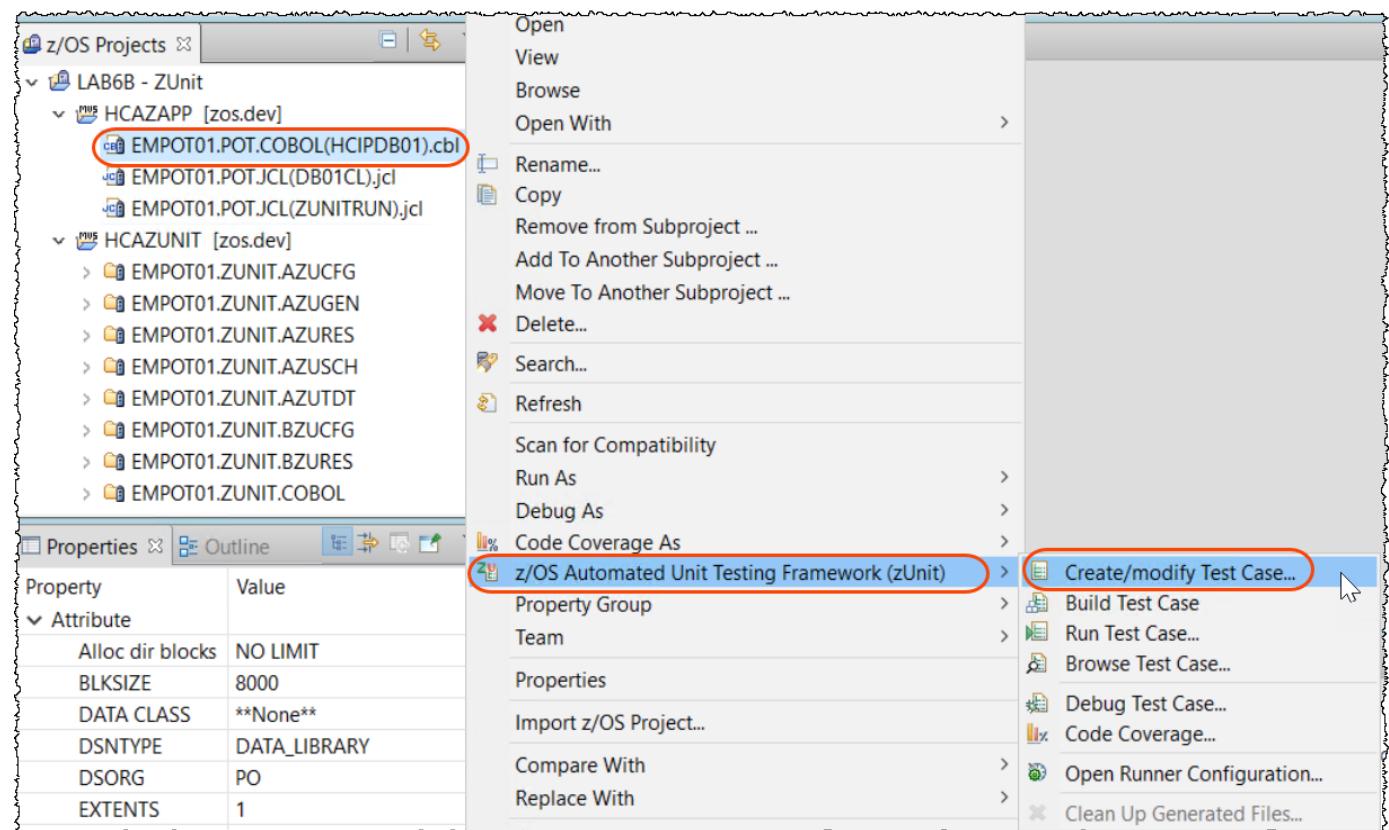
-----+--A-1-B---+--2---+--3---+--4---+--5---+
109      * END PROGRAM and return to caller
110      *
111      MAINLINE-END.
112      EXEC CICS RETURN END-EXEC.
113      MAINLINE-EXIT.
114      EXIT.
115      *
116      GET-PATIENT-INFO.
117      EXEC SQL
118      SELECT FIRSTNAME,
119          LASTNAME,
120          DATEOFBIRTH,
121          insCardNumber,
122          ADDRESS,
123          CITY,
124          POSTCODE,
125          PHONEMOBILE,
126          EMAILADDRESS,
127          USERNAME
128      INTO :CA-FIRST-NAME,
129          :CA-LAST-NAME,
130          :CA-DOB,
131          :CA-INS-CARD-NUM,

```

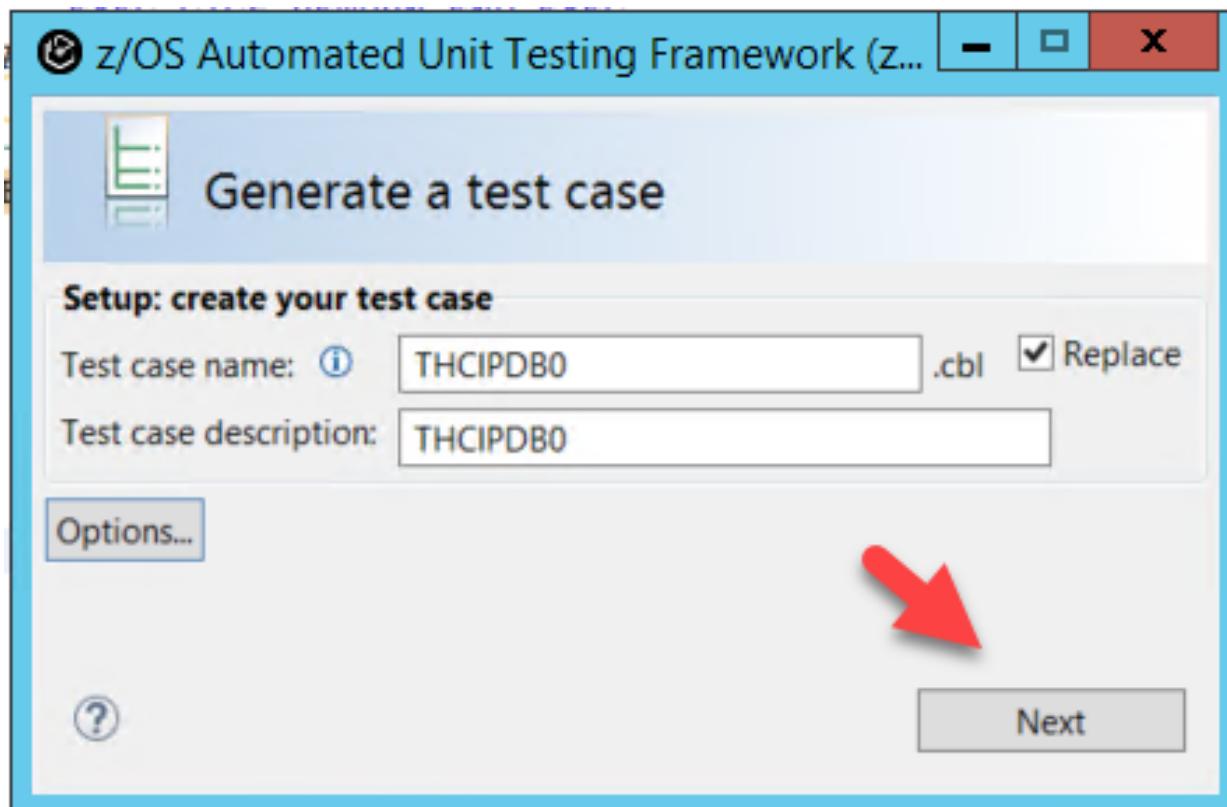
A red box highlights the SQL SELECT statement and the INTO clause. The 'Properties' and 'Remote Error List' tabs are also visible at the bottom of the interface.

3.2 Recording the COBOL program that sends the message

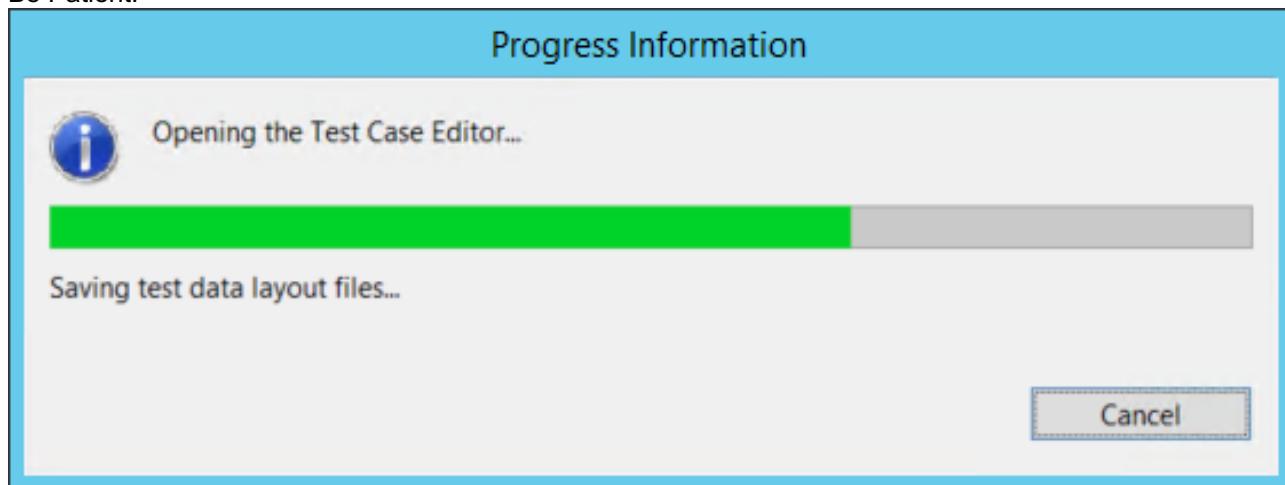
- 3.2.1 ► To start the recording, right click on **EMPOT01.POT.COBOL(HCIPDB01).cbl** and select **z/OS Automated Unit Testing Framework (zUnit)-> Create/modify Test Case..**



3.2.2 ► This opens a dialog where you can name your test case. Accept the auto-generated name and click **Next**.



3.2.3 This operation may take a while since members are being created on z/OS PDS **EMPOT01.ZUNIT.***.
Be Patient!

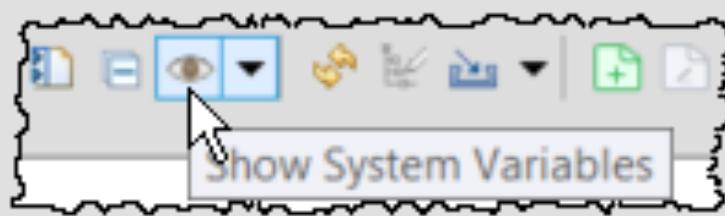


3.2.4 This will open the **Test Case Editor**, as shown below. TEST1 may or may not be on your screen. If TEST1 is there you will delete on next step.

Level	Name	PIC	USAGE	TEST1 :Input	TEST1 :Output
PROCEDURE DIVISION					
Record:Record1					
DFHEIBLK					
DFHCOMMAREA					
Layout					
1	DFHCOMMAREA				
3	CA-REQUEST-ID	X(6)	DISPLAY		
3	CA-RETURN-C...	9(2)	DISPLAY		
3	CA-PATIENT-ID	9(10)	DISPLAY		
3	REDEFINES CA-...				
3	CA-REQUEST-S...	X(3248...)	DISPLAY		
3 (redefines)	CA-PATIENT-R...				
5	CA-INS-CARD...	X(10)	DISPLAY		
5	CA-FIRST-NAME	X(10)	DISPLAY		
5	CA-LAST-NAME	X(20)	DISPLAY		
5	CA-DOB	X(10)	DISPLAY		

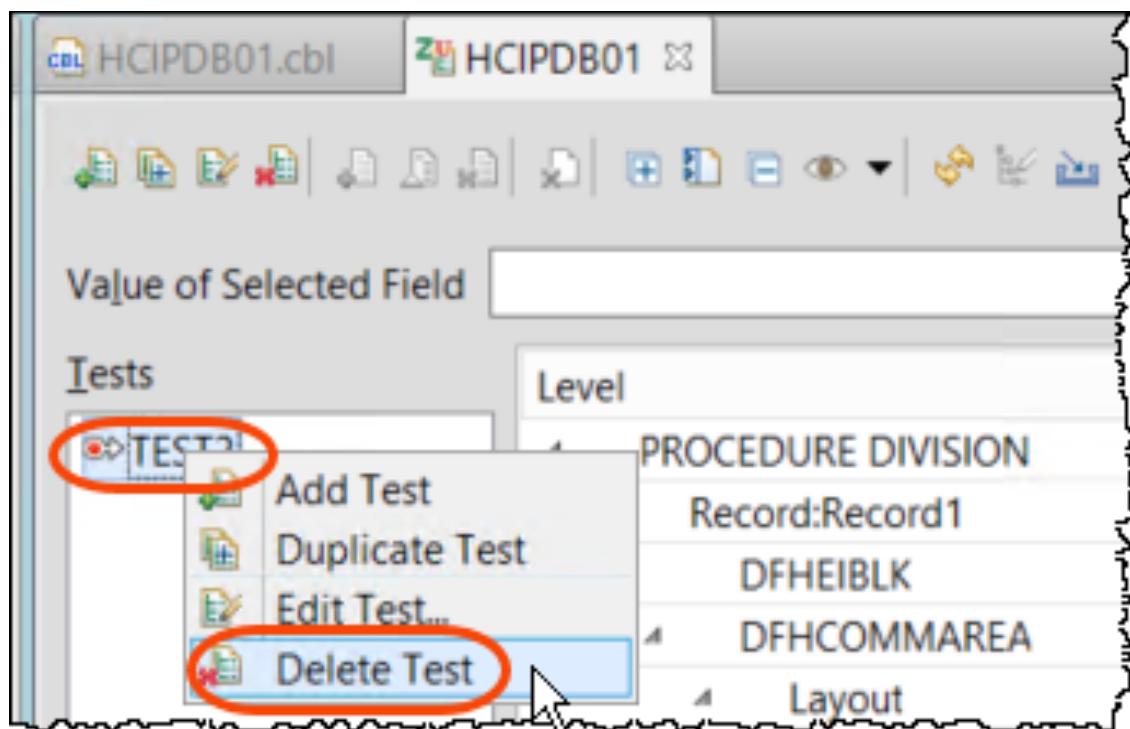
Understanding the test case editor

- The bottom left box summarizes all the input output variable structures – In this example, the program has a COMMAREA exchanged via the PROCEDURE DIVISION, five EXEC CICS statements and one EXEC SQL statement.
- The COMMAREA and the CICS statements are also accompanied by the CICS DFHEIBLK variables. To see those values you must select the icon



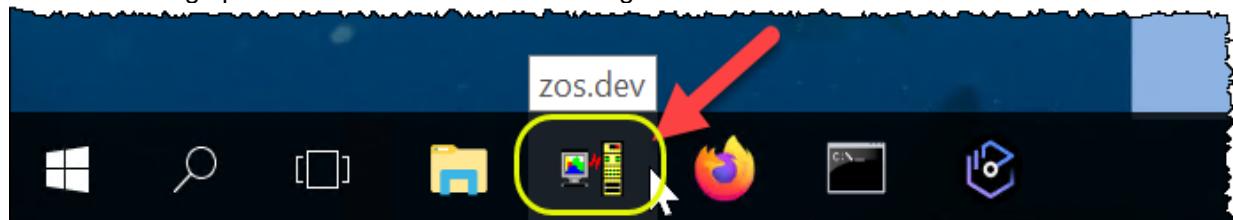
The variables that are returned by the program are the output from the program logic that a developer should be checking

3.2.5 ► Since we will be recording the test data, delete the **TESTx** entry (if it exists) by right clicking and selecting **Delete Test**



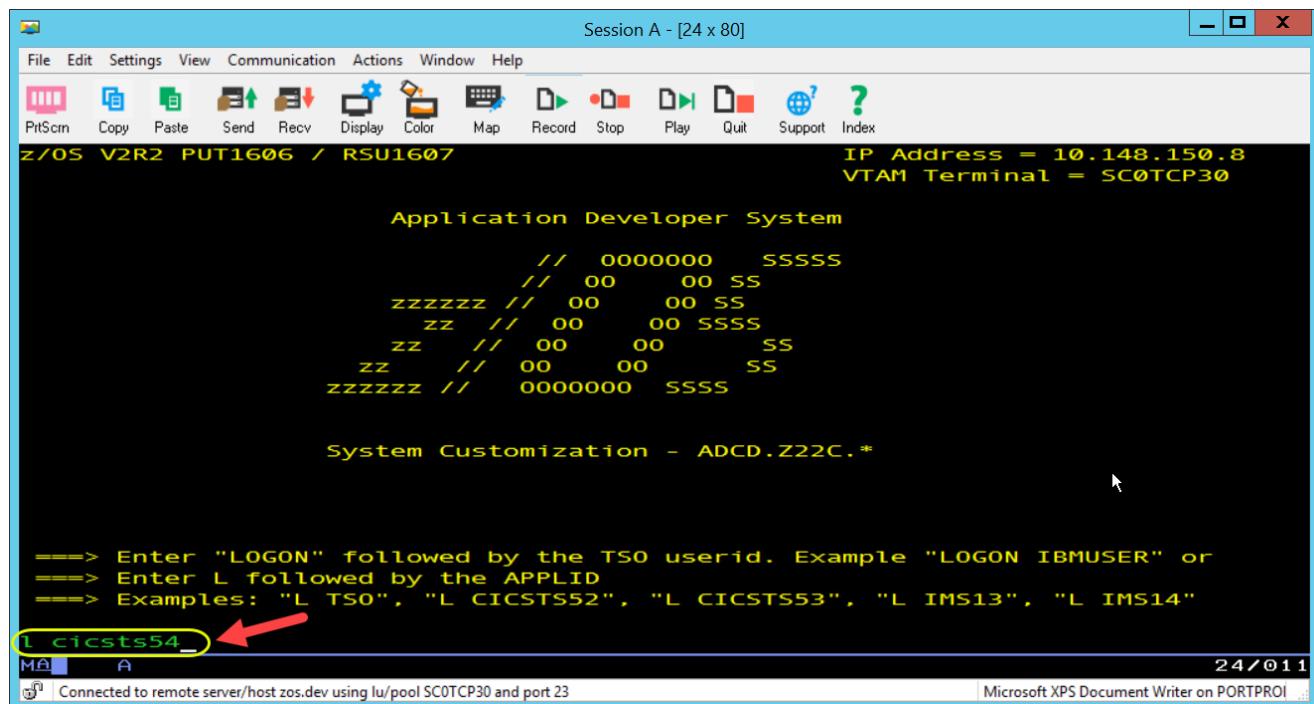
Notice that on section 1 we used the IDz emulator and now you will use the IBM PCOM emulator. Any 3270 emulator can be used.. We just show here a different way to emulate a 3270 terminal.

- 3.2.6 ► Bring up a 3270 terminal emulator clicking on the **host emulator icon** on the Windows task bar.

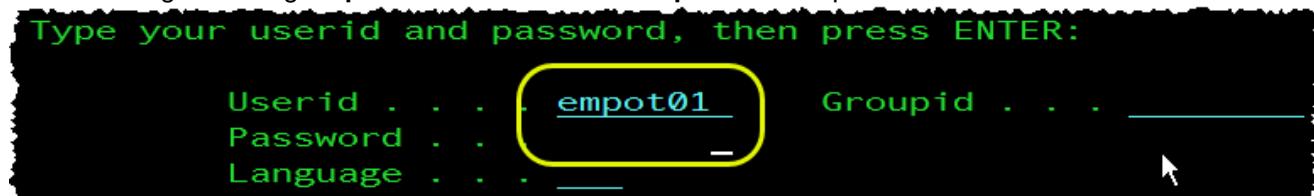


This opens the host emulator.

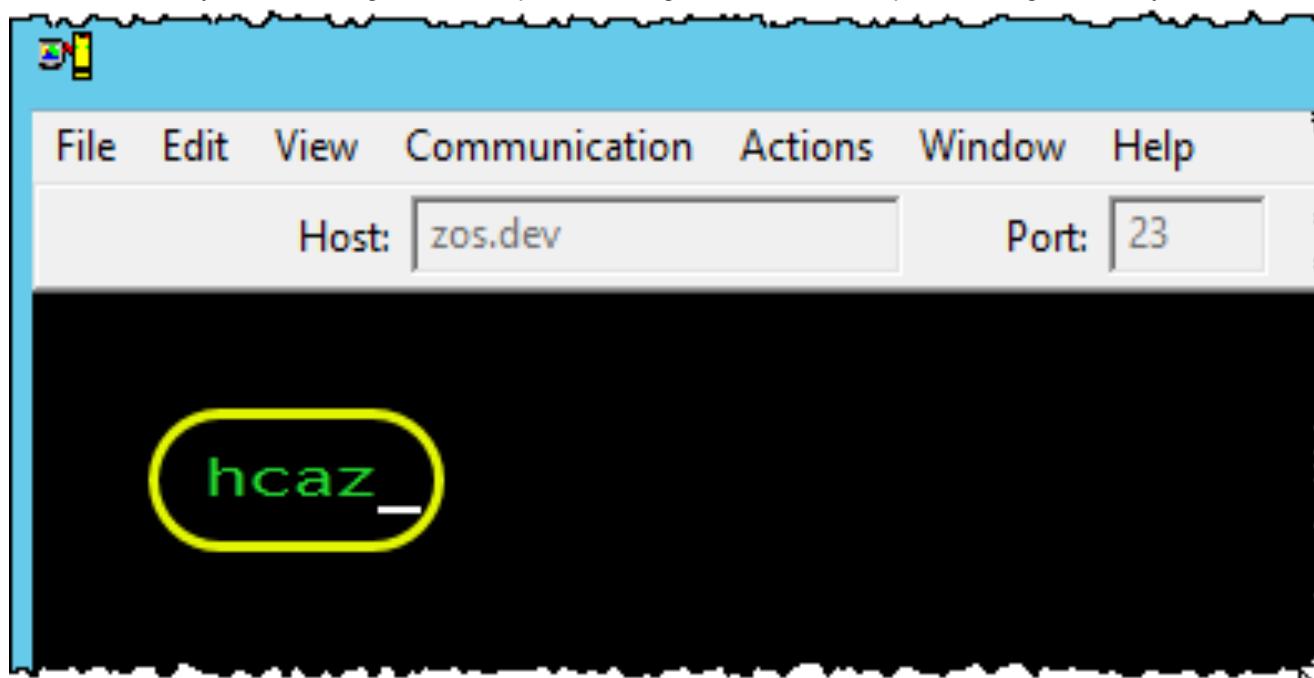
- 3.2.7 ► Type **I cicsts54.** (where I is L lower case) and press **Enter key** or the **right Ctrl key**.



- 3.2.8 ► Sign on using **empot01** as the userid and **empot01** as the password.

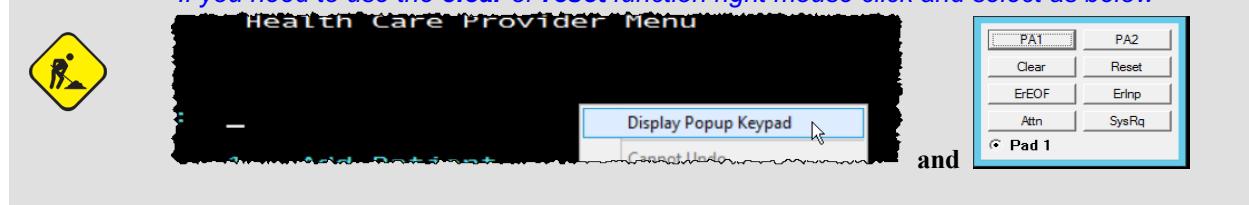


- 3.2.9 ► Once you see the sign-on is complete message, enter **hcaz** and press the right **Ctrl** key.



IF you need to use functions like Clear or Reset

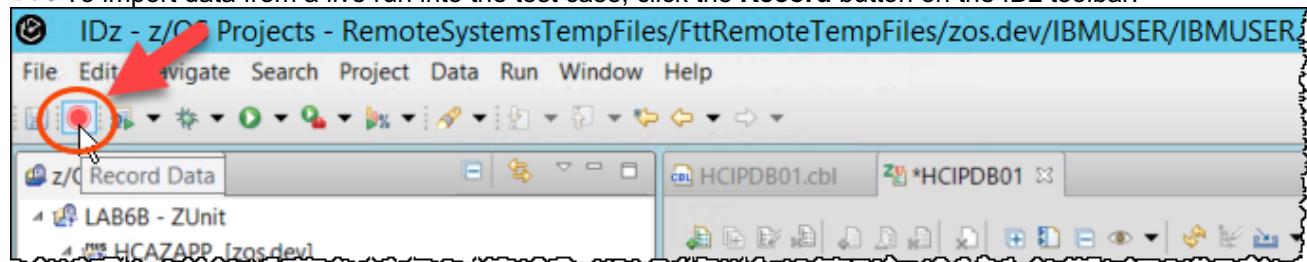
If you need to use the clear or reset function right mouse click and select as below



You are now ready to start recording and import data into the test case editor.

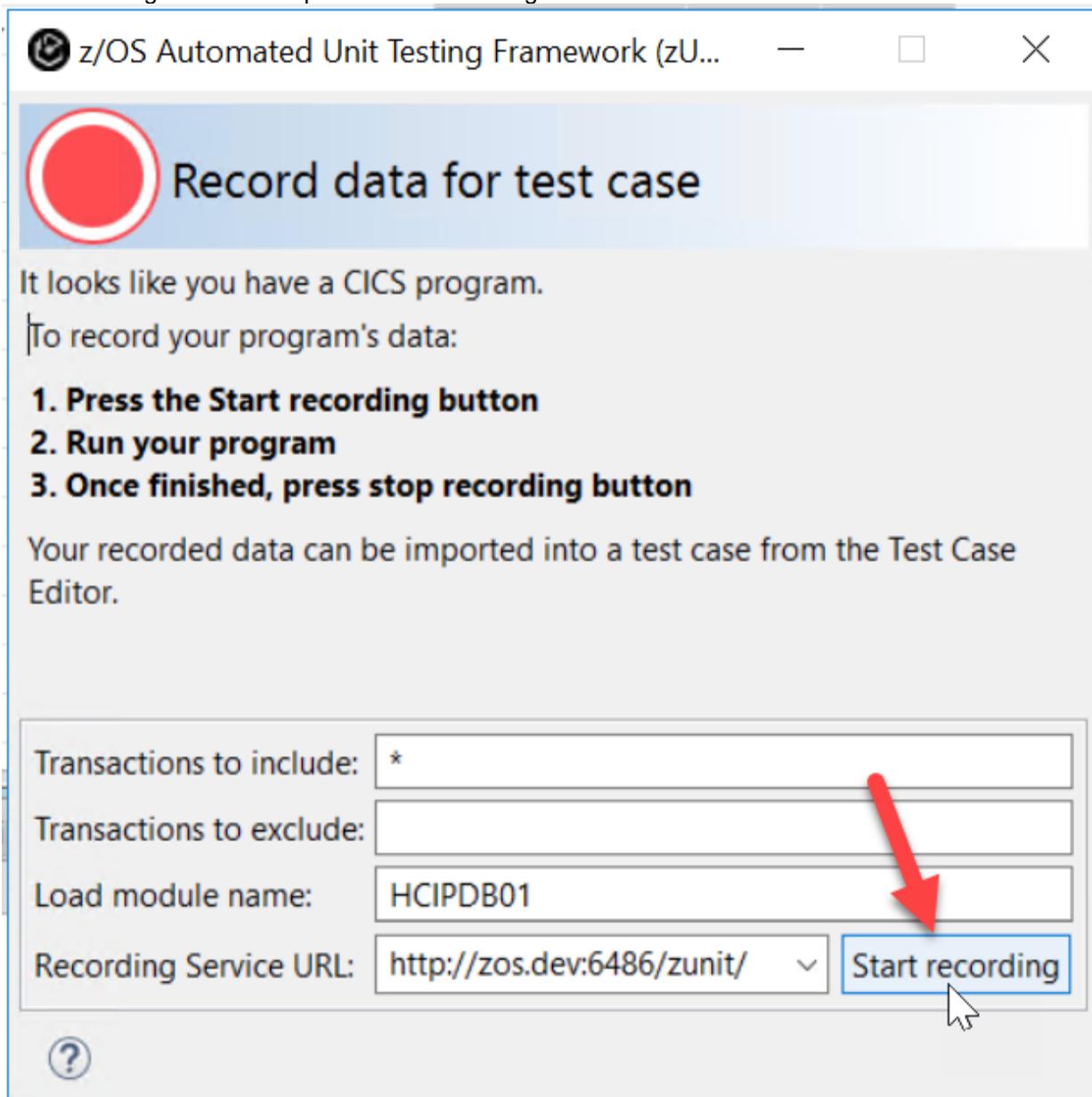
3.2.10 ► Minimize the terminal emulator and go back to IDz dialog.

► To import data from a live run into the test case, click the **Record** button on the IDz toolbar.

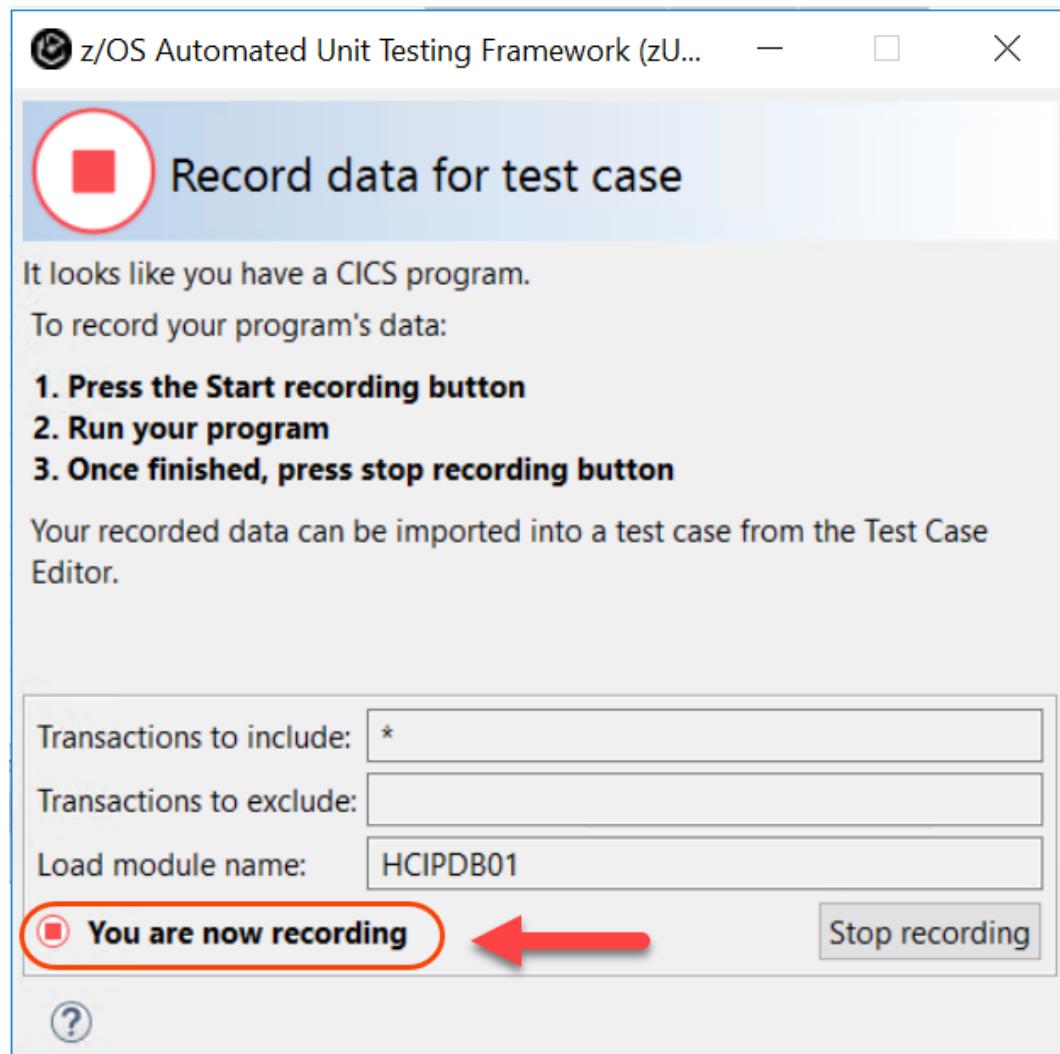


3.2.11 ► In the dialog that comes up, click on **Start recording**.

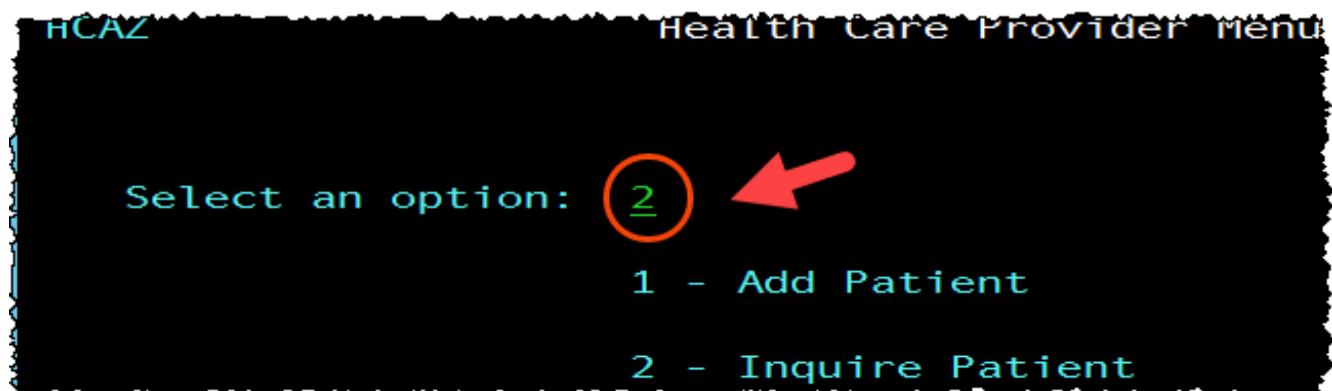
The Recording Service URL points to the CICS region where the live run is recorded.



3.2.12 When recording is turned on, the message “**You are now recording**” appears.



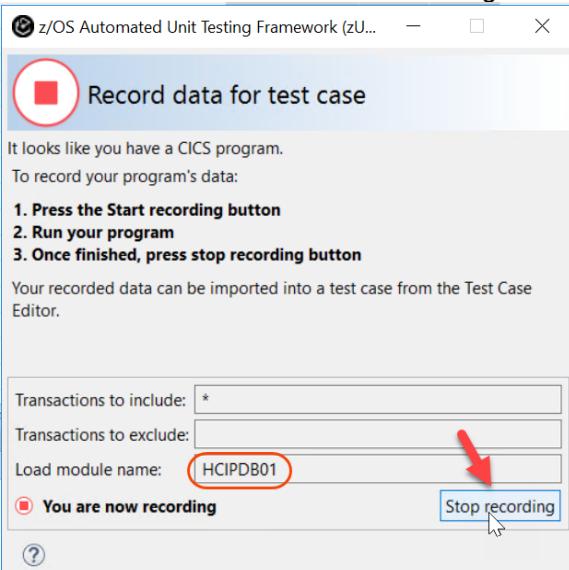
3.2.13 ➡ Switch to the 3270 terminal emulator (that is minimized), type **2** (Inquire Patient) and press **enter**.



3.2.14 ➤ Type 1 and press enter

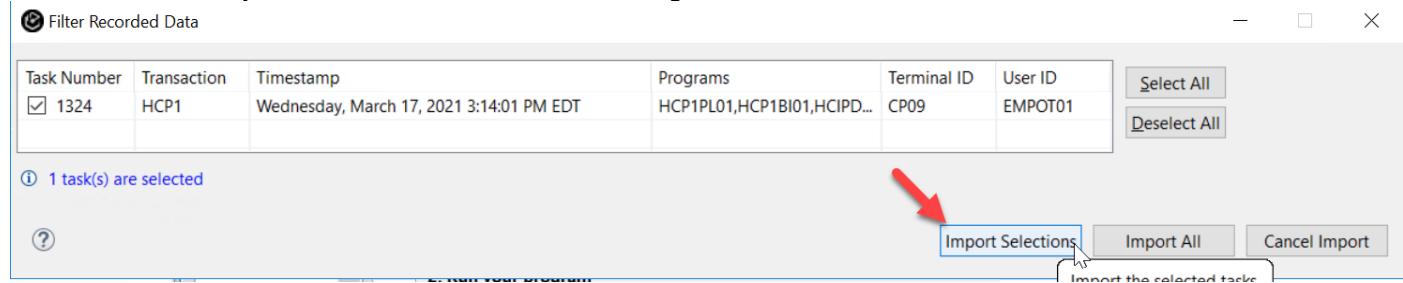


3.2.15 ►| Switch back to IDz minimizing the 3270 screen and click **Stop recording**.



The dialog Filter Recorded Data is displayed.

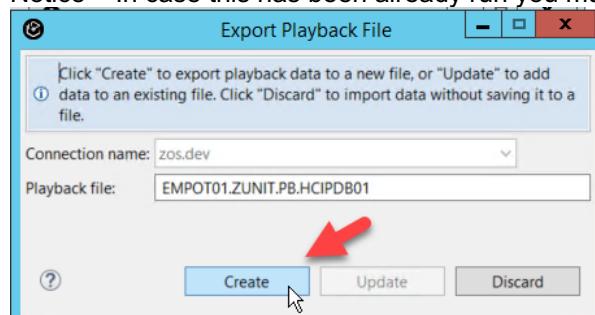
3.2.16 ►| Click **Import Selections** to dismiss the dialog.



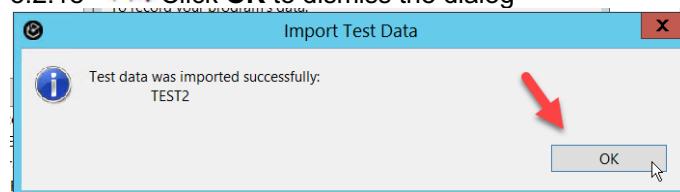
3.2.17 ►| Change the *Playback file* to **EMPOT01.ZUNIT.PB.HCIPDB01** (instead of **EMPOT05**)

►| Click **Create** to export playback data to a z/OS data set

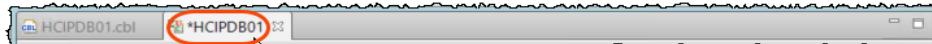
Notice – In case this has been already run you must select **Update** instead.



3.2.18 ►| Click **OK** to dismiss the dialog



3.2.19 ► Double click on the **Hcipdb01** title to enlarge the view.



3.2.20 You now see a new test created in the editor and populated with the values from the live run. Also you may notice that this program has many redefines.

You need to identify which is the area being used on this program execution. In our example is the “CA-PATIENT-REQUEST” area being redefined.

► Find “(redefines) CA-PATIENT-REQUEST” (the first REDEFINES). Right click on it and click **Select Redefines Structure**

The screenshot shows the IBM zUnit interface with the 'Hcipdb01' test selected. A context menu is open over the 'REDEFINES CA-REQUEST-SPECIFIC' entry in the hierarchy. The 'Select Redefines Structure' option is highlighted with a red circle. Other options like 'Edit', 'Select Data Type...', 'Set Test Result Format', and 'Add Test' are also visible.

3.2.21 ► Select CA-PATIENT-REQUEST and click OK

The screenshot shows the IBM zUnit interface with the 'Hcipdb01' test selected. A 'Redefine item' dialog box is open, showing the 'REDEFINES CA-REQUEST-SPECIFIC' item under 'In test: TEST2'. The 'Data item:' section contains several radio button options, with 'CA-PATIENT-REQUEST [redefines CA_REQUEST_SPECIFIC]' selected and highlighted with a red circle. Below this, there are four radio button options for applying the selection: 'Apply the selection to this record only' (selected), 'Apply the selection to all records in this test', 'Apply the selection to this record in all tests', and 'Apply the selection to all records in all tests'. At the bottom right of the dialog are 'OK' and 'Cancel' buttons.

3.2.22 Notice that now the patient data recorded is displayed at the new redefines selected.

Level	Name	PIC	USAGE	TEST2 :Inp...	TEST2 :Exp...
PROCEDURE DIVISION					
Record:Record1	DFHEIBLK				
DFHCOMMAREA					
Layout					
1	DFHCOMMAREA				
3	CA-REQUEST-ID	X(6)	DISPLAY	01IPAT	01IPAT
3	CA-RETURN-CODE	9(2)	DISPLAY	0	0
3	CA-PATIENT-ID	9(10)	DISPLAY	1	1
REDEFINES CA-REQUEST-SPECIFIC					
3 (redefine)	CA-PATIENT-REQUEST [redefines CA_REQUEST-SPECIFIC]	X(3248...)	DISPLAY	9627811234Ralph	
5	CA-INS-CARD-NUM	X(10)	DISPLAY		
5	CA-FIRST-NAME	X(10)	DISPLAY		
5	CA-LAST-NAME	X(20)	DISPLAY		
5	CA-DOB	X(10)	DISPLAY		
5	CA-ADDRESS	X(20)	DISPLAY		
5	CA-CITY	X(20)	DISPLAY		
5	CA-POSTCODE	X(10)	DISPLAY		
5	CA-PHONE-MOBILE	X(20)	DISPLAY		
5	CA-EMAIL-ADDRESS	X(50)	DISPLAY		
5	CA-USERID	X(10)	DISPLAY		

3.2.23 Press **Ctrl+S** to save any changes. Notice this save takes a while since values are saved on **z/OS**

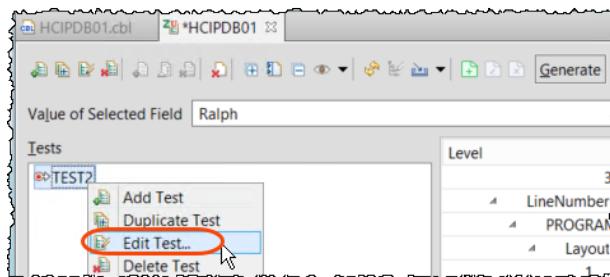
3.2.24 You now see a new test created in the editor and populated with the values from the live run.

Scroll down the editor and notice the data displayed on the screen that was captured..

- 3.2.25 ① Select **EXEC SQL SELECT INTO (PATIENT)** to position the data layout for this statement.
- ② Use the scroll bar to scroll down until the end,
- ③ Click on **Ralph** . and notice that value is displayed on the line ④ "

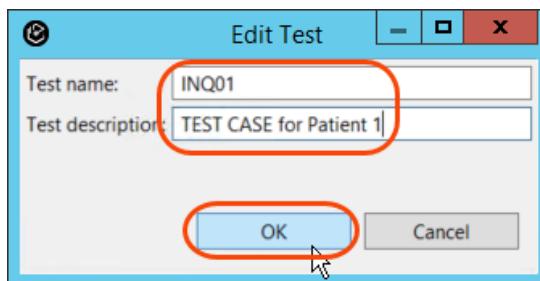
Level	Name	PIC	USAGE	TEST2 :Inp...	TEST2 :Exp...
1	HCAZERRS	X(8)	DISPLAY		
COMMAREA					
Layout					
1	CA-ERROR-MSG				
3	FILLER	X(9)	DISPLAY		
3	CA-DATA	X(90)	DISPLAY		
EXEC SQL SELECT INTO [PATI	line=117				
Record:Record1					
LineNumber=117					
INTO					
5	CA-FIRST-NAME	X(10)	DISPLAY	Ralph	3
5	CA-LAST-NAME	X(20)	DISPLAY	DALmeida	
5	CA-DOB	X(10)	DISPLAY	1980-07-11	
5	CA-INS-CARD...	X(10)	DISPLAY	9627811234	
5	CA-ADDRESS	X(20)	DISPLAY	34 Main Street ...	
5	CA-CITY	X(20)	DISPLAY	Toronto	
5	CA-POSTCODE	X(10)	DISPLAY	M5H 1T1	
5	CA-PHONE-M...	X(20)	DISPLAY	077-123-9987 ...	
5	CA-EMAIL-AD...	X(50)	DISPLAY	RalphD@ibm.c...	
5	CA-USERID	X(10)	DISPLAY	ralphd	
WHERE					
3	DB2-PATIENT-ID	S9(9)	BINARY	1	
SQLCA					

3.2.26 ► Right click on **TEST2** and select **Edit Test**



3.2.27 It is a good practice give a name for the test case.

► Use a name like **INQ01** and a description like “**TEST CASE for Patient 1**”: . Click **OK**



3.2.28 ► Press **Ctrl+S** to save any changes.

3.2.29 ► Double click again on the **HCIPDB01** title to shrink the view.

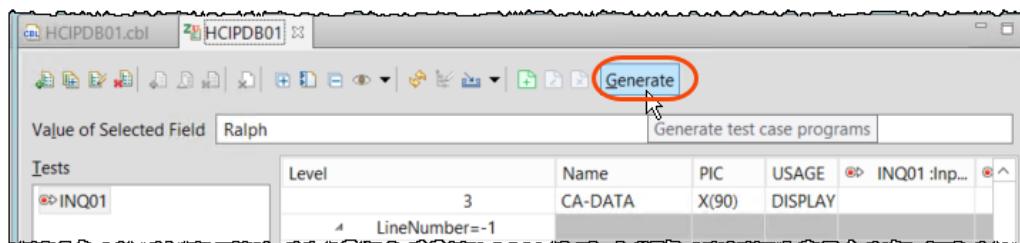


Section 4. Generate, build and run the unit test.

You will generate, build, and run the unit test.

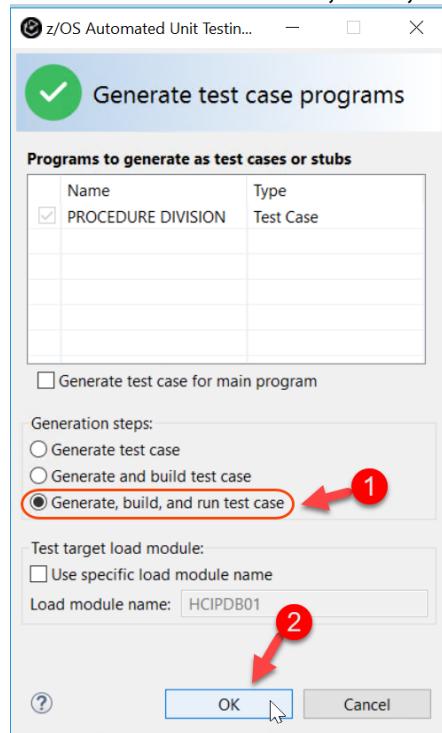
4.1 Generating, building and running the test case program.

4.1.1 ► Now that the expected input and output values have been set in the test case editor from the recorded run, click on **Generate** to generate the test case program.



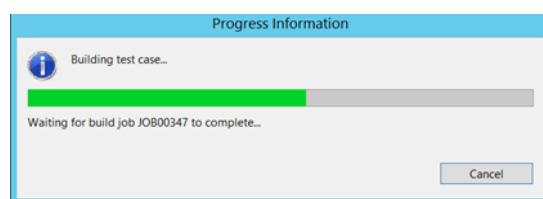
On the **Generate test case programs** dialog,

4.1.2 ► Select **Generate, build, and run test case** and then click **OK**.

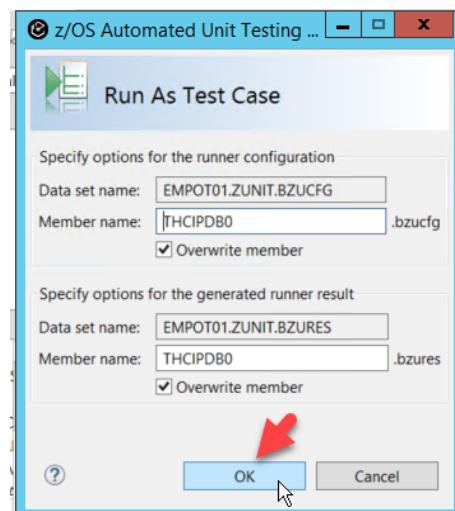


A **Progress information** dialog on the building will open.

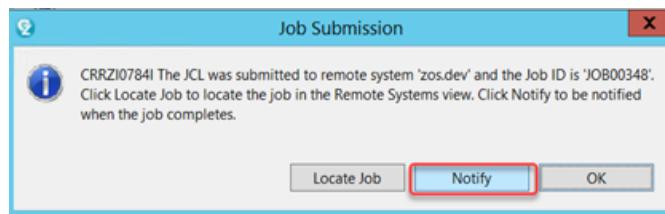
This generation will take a while and a job is submitted to z/OS for execution. The COBOL test cases programs were compiled/link edited and are ready to run.



4.1.3 ► Once the building is complete, a **Run As Test Case** dialog will open, click **OK** to continue.



4.1.4 ► A Job Submission dialog opens, click **Notify** to be notified of job completion.



#1. Had a return code 12 as below?



An error occurred while building the test case
Reason:
CRRZT0083E JES job with job name EMPOT011 and job ID JOB00390 failed on remote system zos.dev. Return code: CC 0012.
OK

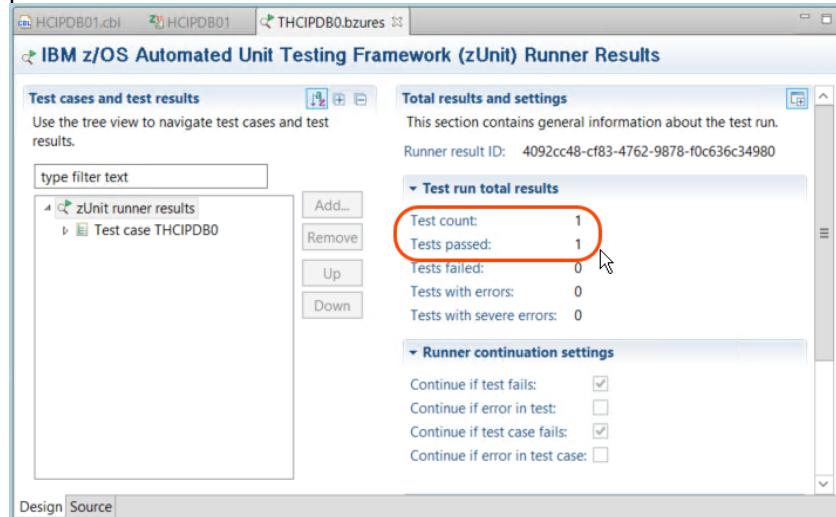
*You might be using a wrong IDz Workspace...
Be sure you are using the workspace below and IDz V 15.0.1*

IDz_review - C:\Workspaces\IDz15\IDz_review - IBM Developer for z/OS
File Edit Navigate Search Project Run Window Help

*Call the instructors and
1. Close all opened editors.
2. Delete the data set EMPOT01.ZUNIT.PB.HCIPDB01 and go back to record and try again*

#2. Had a “strange Test fail” ?
*Call the instructor..
1. Delete the data set EMPOT01.ZUNIT.PB.HCIPDB01
2. All members from EMPOT01.ZUNIT.* need to be deleted and the record started again.*

4.1.5 Once the unit test run has completed, the results screen is displayed showing test cases ran and passed.



You have now created a test case with data imported from a recorded run and have been successful in testing a CICS program without the need of a CICS environment.

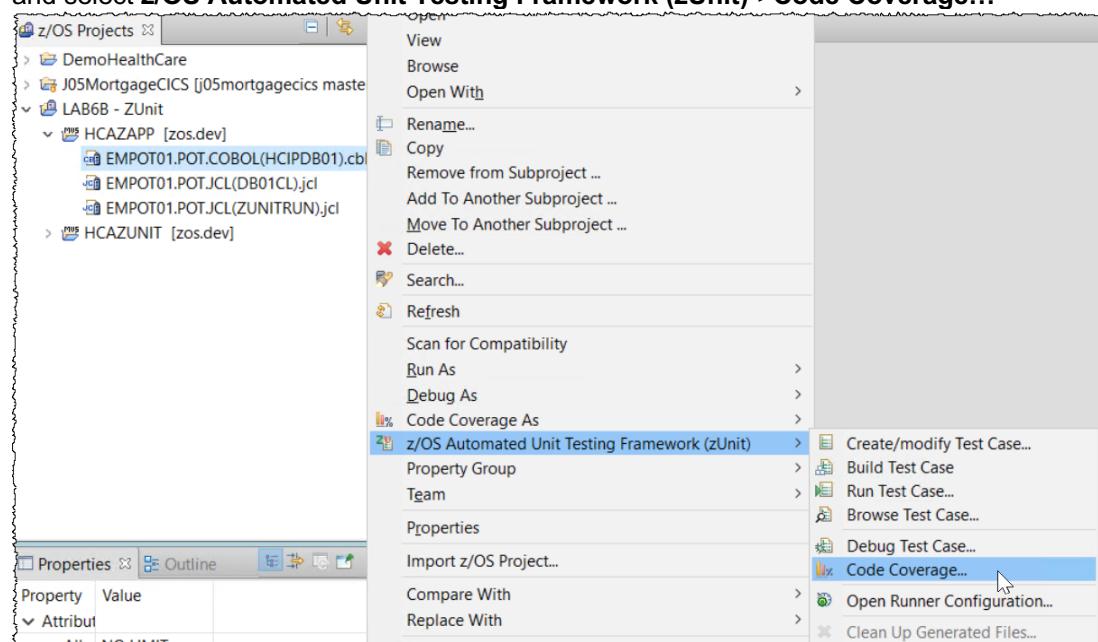
- 4.1.6 ►| Use **Ctrl + Shift + F4** to close all opened editors.

4.2 Running zUnit test case with Code Coverage

Code coverage analyzes a running program and generates a report of lines that are executed, compared to the total number of executable lines. Sometimes you need to be sure that your test is covering all capabilities of your program and that you are having the correct test cases.

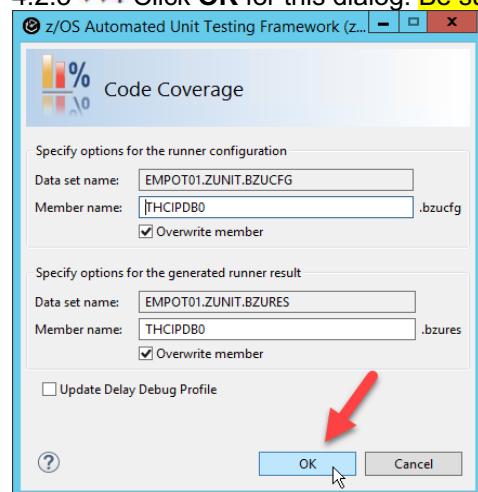
Notice that you are doing the code coverage using a batch job without need to have CICS and DB2 active, what is very handy..

- 4.2.1 ►| Right click on **EMPOT01.POT.COBOL(HCIPDB01).cbl** and select **z/OS Automated Unit Testing Framework (zUnit)->Code Coverage...**



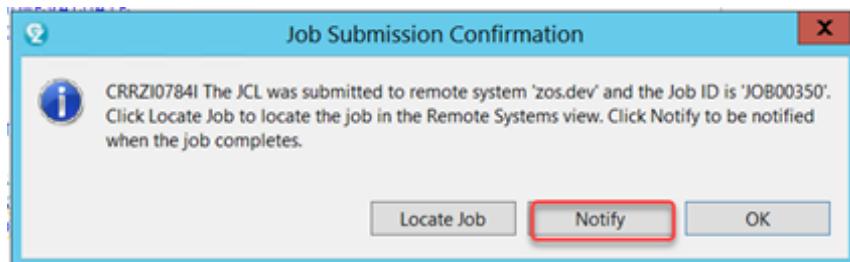
- | If you get a message that files are updated click **Yes** to continue

- 4.2.3 ►| Click **OK** for this dialog. Be sure that *Update Delay Debug Profile* is **NOT** checked.

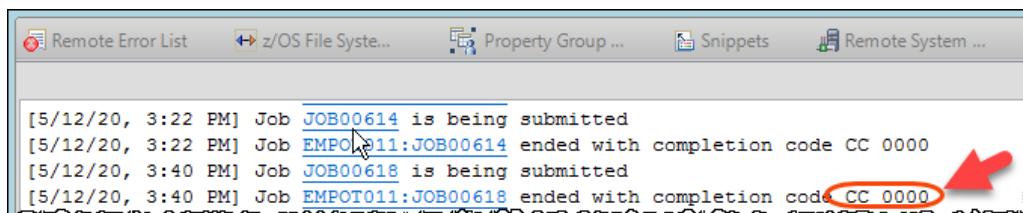


4.2.4 A Job Is submitted for batch execution..

- Click **Notify** on the Job Submission Confirmation dialog.



- Make sure the job completes with completion code of **0**.



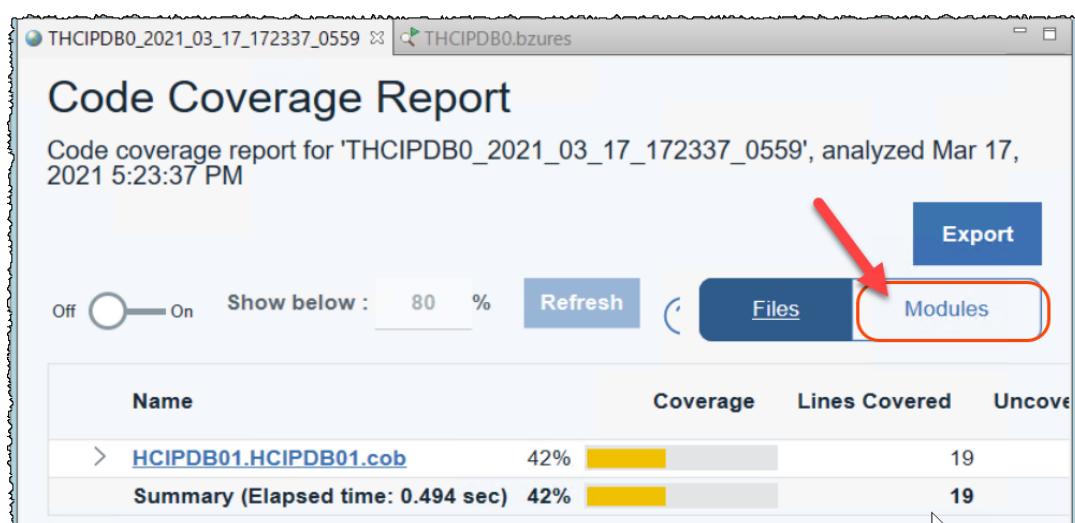
4.2.5 Once again the zUnit Results are displayed and the Test passed.

- Click on **THCIPDB0_yyyy_mm_dd_xxxxxxx** tab to see the *Code Coverage report*



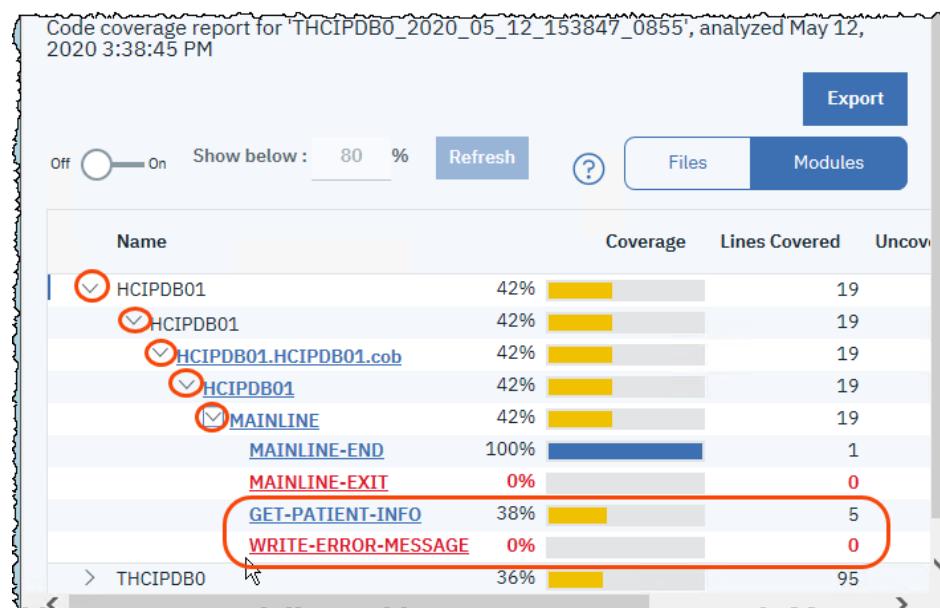
4.2.6 The code coverage report shows all COBOL programs executed for this specific test case. It shows the coverage of the COBOL programs generated for zUnit to perform the test as well our *HCIPDB01* COBOL program..

- Since we are just interested in the code coverage of program being tested click on **Modules**:

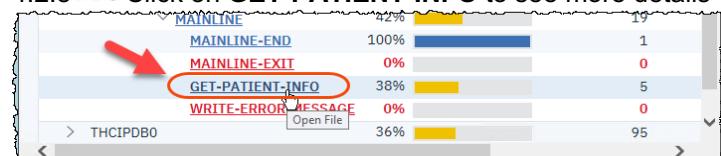


4.2.7 ► Expand the nodes as below to see the COBOL paragraphs.

Notice that **GET-PATIENT-INFO** has 38% of coverage and that **WRITE-ERROR-MESSAGE** was not covered at all on this test case.



4.2.8 ► Click on **GET-PATIENT-INFO** to see more details



4.2.9 ► Scroll down and move the mouse to the colored areas on left.

The lines covered by this test case are in **green** and the lines not covered are in **red**.

Also from the previous image we can see the **WRITE-ERROR_MESSAGE** paragraph is not covered at all.

The screenshot shows a COBOL source code editor with the file **HCIPDB01.HCIPDB01.cob**. The code is displayed with line numbers on the left. A yellow speech bubble labeled "lines covered" points to a green vertical bar on the far left of the code area, indicating covered lines. A red arrow points to a red vertical bar on the far left, indicating uncovered lines. A callout box at the bottom left of the code area states "Lines 259-266 not covered." The code itself contains several COBOL statements like **MOVE**, **EVALUATE**, and **PERFORM**.

4.2.10 From this report we can see that the test cases created for this program are not sufficient ..
The developer could go back to the test cases editor (step 3.2.19 above) and manually add test cases that cover other paragraphs. Due the lack of time we will not cover that here, but if you have extra time you can try it. Ask the instructor for guidance.

Section 5. Introduce a bug in the program and rerun the unit test.

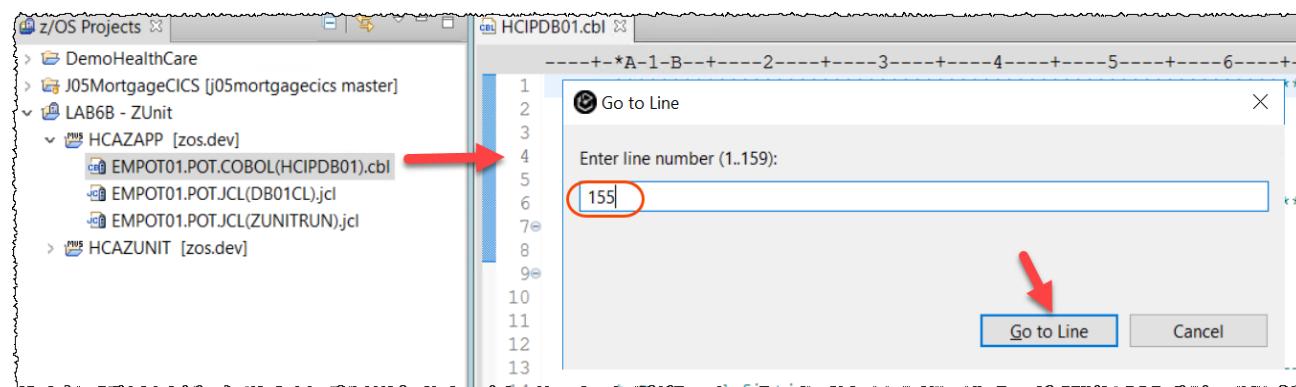
Using IDz you will modify the program and introduce a bug. Then you will rerun the unit test created in the previous section.

5.1 Modifying the program and introduce a bug

5.1.1 ► On IDz, close all active windows by pressing **Ctrl+Shift+F4**

► Open **EMPOT01.POT.COBOL (HCIPDB01).cbl** by double clicking on it in the z/OS Projects view.

5.1.2 ► On the editor, go to line 155 by pressing **Ctrl+L** and entering **155** and clicking **Go to Line**.



5.1.3 ► Change the line 155 removing the * from the statement that moves "BAD NAME",

► Press **Ctrl+S** to save the changes.

```

138     FROM PATIENT
139     WHERE PATIENTID = :DB2-PATIENT-ID
140     END-EXEC.
141     Evaluate SQLCODE
142     When 0
143       MOVE '00' TO CA-RETURN-CODE
144     When 100
145       MOVE '01' TO CA-RETURN-CODE
146     When -913
147       MOVE '01' TO CA-RETURN-CODE
148     When Other
149       MOVE '90' TO CA-RETURN-CODE
150       PERFORM WRITE-ERROR-MESSAGE
151       EXEC CICS RETURN END-EXEC
152     END-Evaluate.
153   * %bug -- the line below will introduce a BUG
154   *
155   * MOVE "BAD NAME" to CA-FIRST-NAME
156   *
157   EXIT.
158   *

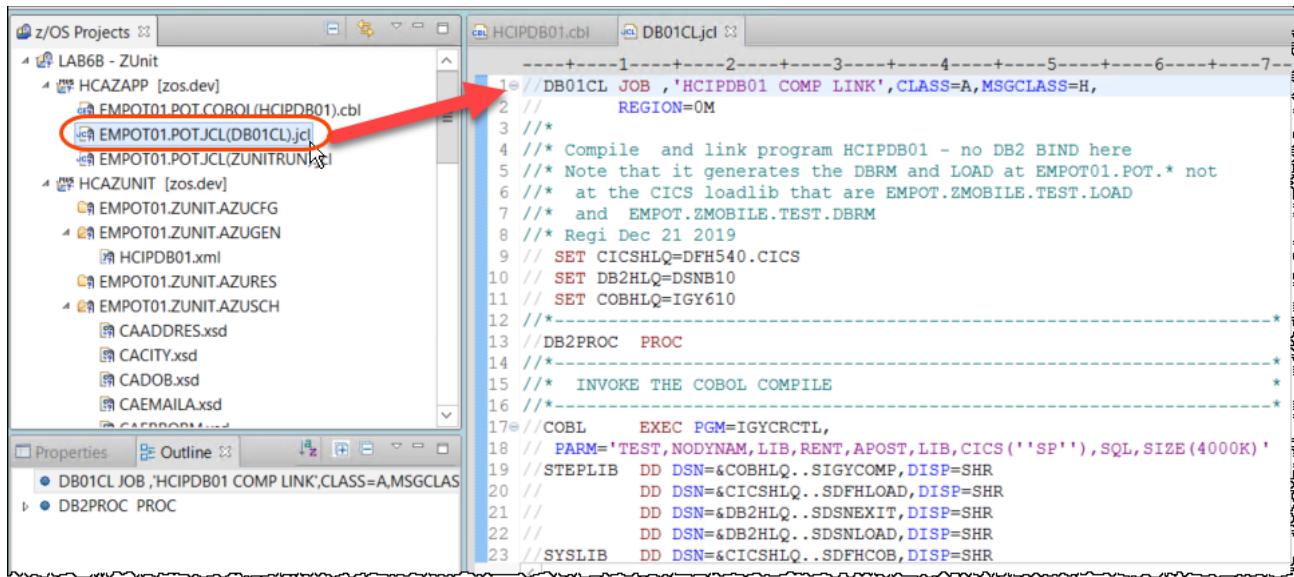
```

5.2 Rebuilding the changed program without deploying to CICS

- .5.2.1 ► On the z/OS Projects view, double click **EMPOT01.POT.JCL (DB01CL).jcl**, to edit .

This JCL will compile and link the changed COBOL program *HCIPDB01* using a working PDS to store the load module.

Notice that DB2 bind is not necessary since DB2 SQL calls will be intercepted when running the generated zUnit test cases.



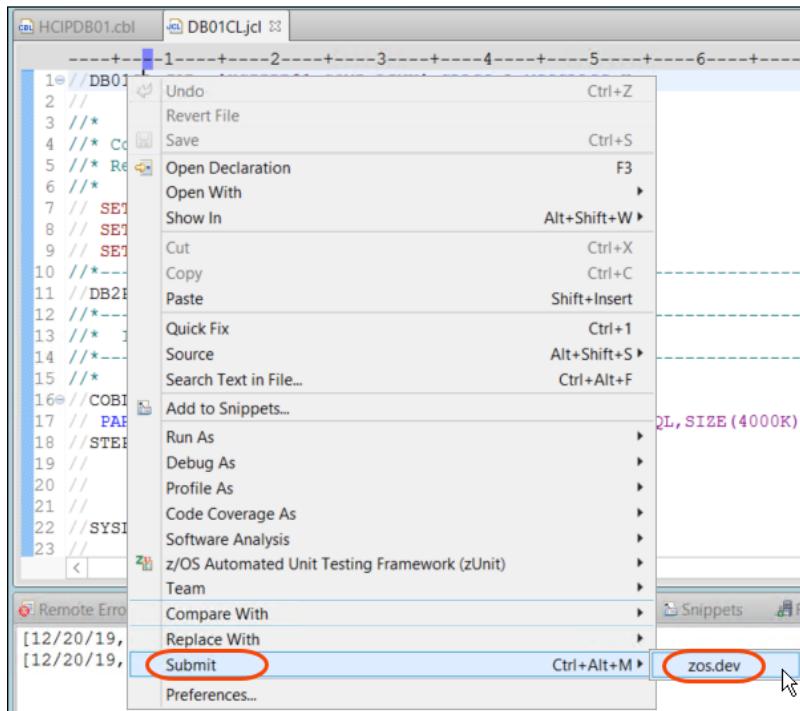
```

z/OS Projects : LAB68 - ZUNIT
  + HCAZAPP [zos.dev]
    + EMPOT01.POT.COBOL(HCIPDB01).cbl
      + EMPOT01.POT.JCL(DB01CL).jcl (highlighted)
      + EMPOT01.POT.JCL(ZUNITRUNK) &
  + HCAZUNIT [zos.dev]
    + EMPOT01.ZUNIT.AZUCFG
    + EMPOT01.ZUNIT.AZUGEN
    + HCIPDB01.xml
    + EMPOT01.ZUNIT.AZURES
  + EMPOT01.ZUNIT.AZUSCH
    + CAADDRES.xsd
    + CACITY.xsd
    + CADOB.xsd
    + CAEMAILA.xsd
    + CAEPD01.xsd

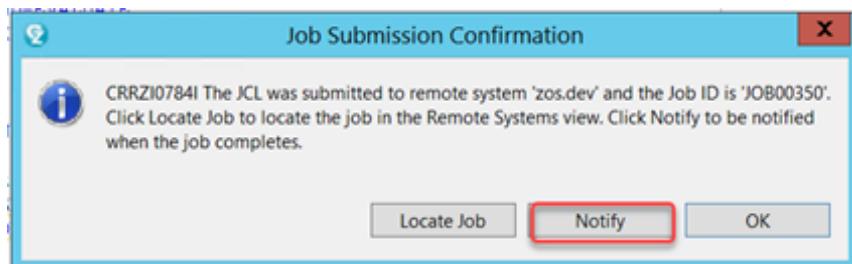
Properties : Outline : z/OS Projects : HCIPDB01.cbl : DB01CL.jcl
  + DB01CL.JOB , 'HCIPDB01 COMP LINK', CLASS=A, MSGCLASS=H,
  + REGION=0M
  + /*
  +  * Compile and link program HCIPDB01 - no DB2 BIND here
  +  * Note that it generates the DBRM and LOAD at EMPOT1.POT.* not
  +  * at the CICS loadlib that are EMPOT.ZMOBILE.TEST.LOAD
  +  * and EMPOT.ZMOBILE.TEST.DBRM
  +  * Regi Dec 21 2019
  +  * SET CICSHLQ=DFH540.CICS
  +  * SET DB2HLQ=DSNB10
  +  * SET COBHLQ=IGY610
  +  */
  + /*
  +  * DB2PROC PROC
  +  */
  + /*
  +  * INVOKE THE COBOL COMPILE
  +  */
  + /*
  +  * COBL EXEC PGM=IGYCRCTL,
  +  * PARM='TEST,NODYNAM,LIB,RENT,APOST,LIB,CICS(''SP''),SQL,SIZE(4000K)'
  +  * STEPLIB DD DSN=&COBHLQ..SIGYCOMP,DISP=SHR
  +  * DD DSN=&CICSHLQ..SDFHLOAD,DISP=SHR
  +  * DD DSN=&DB2HLQ..SDSNEXIT,DISP=SHR
  +  * DD DSN=&DB2HLQ..SDSNLOAD,DISP=SHR
  +  */
  + /*
  +  * SYSLIB DD DSN=&CICSHLQ..SDFHCOB,DISP=SHR
  +  */

```

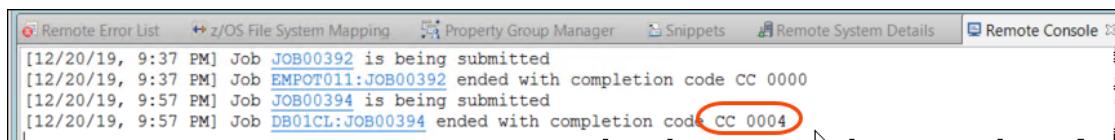
- .5.2.2 ► Right click and select **submit > zos.dev** to submit this job for execution



5.2.3 ► Click **Notify** on the Job Submission Confirmation dialog.

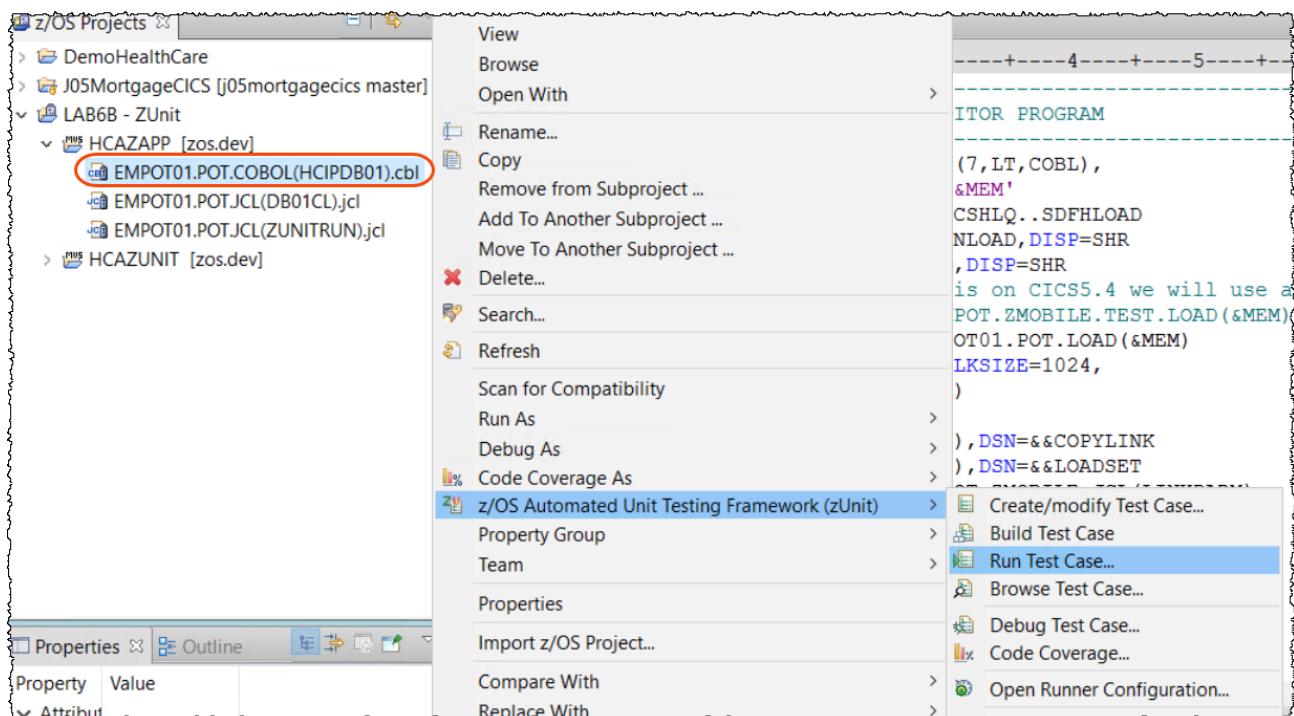


Make sure the job completes with completion code of 4.

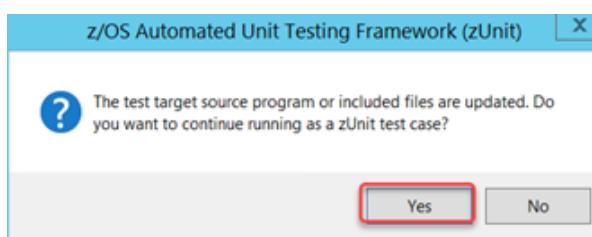


5.3 Rerunning the test case

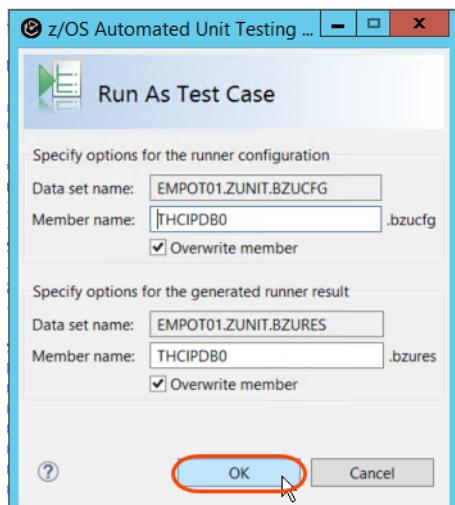
5.3.1 ► On the z/OS Projects view, select **EMPOT01.POT.COBOL(HCIPDB01).cbl**, right mouse click, and select the **z/OS Automated Unit Testing Framework (zUnit)**->**Run Test Case..** action.



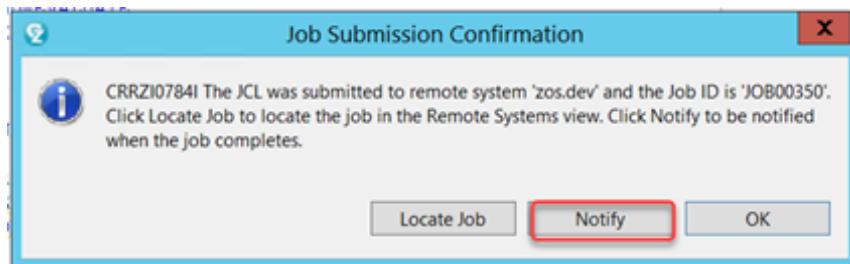
5.3.2 ► Click **Yes** to the following dialog to continue running the test case.



5.3.3 ► Click **OK** to the **Run As Test Case** dialog.



5.3.4 ► Click **Notify** on the Job Submission Confirmation dialog.

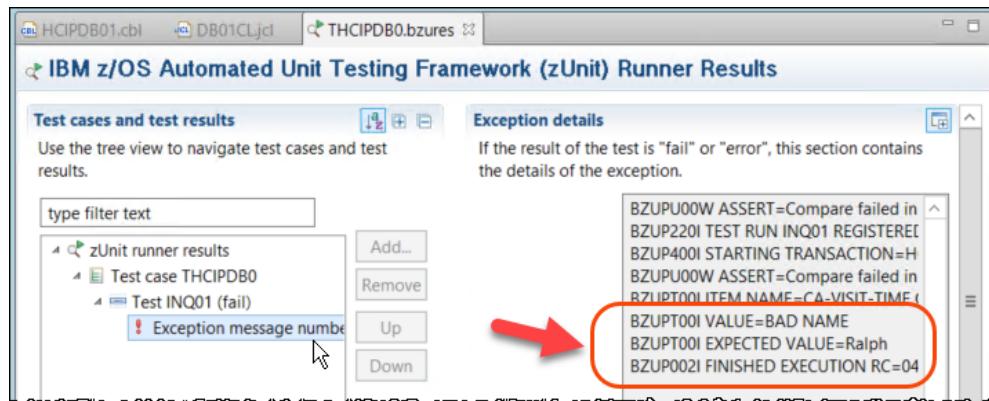


5.3.5 When the test run completes, the test results will be displayed, and it showed 1 test was ran and it failed.

► Click **Test case THCIPDB0**. The failure is expected because the expected value of the error message is different from the actual value.

Test case details
This section contains information about the test case.
Test case ID: e30e513c-140b-4176-b4f1-0b497de49f44
Module name: THCIPDB0
Test case name: THCIPDB0
Result: fail
Test count: 1
Tests passed: 0
Tests failed: 1
Tests with errors: 0
Tests with severe errors: 0

- 5.3.6 ► Expand **Test case THCPDB0, Test INQ01 (fail)** and click **Exception message number** to verify the value received versus the expected value and verify the failure



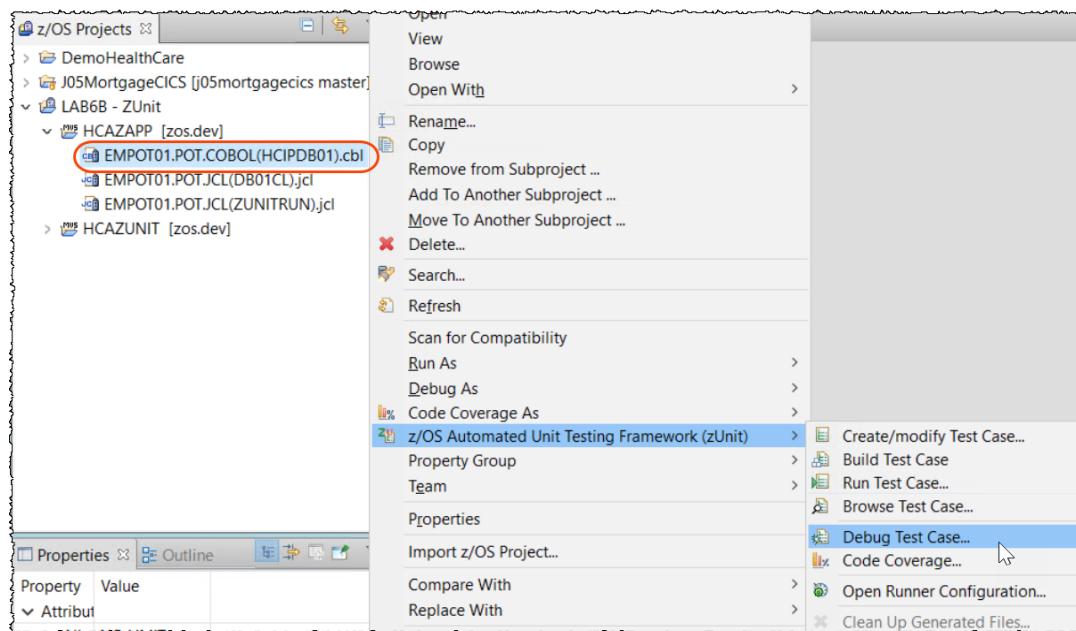
5.4 Running zUnit test case with debugging

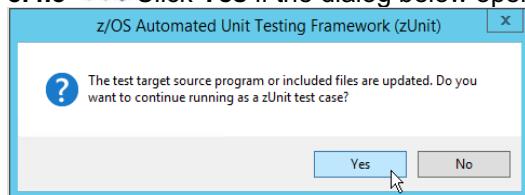
The Debug dialog will debug the test case. Note that will debug the zUnit programs, the generated COBOL and also the COBOL program that you are testing. Below one example.

Notice that you are debugging a batch job without need to have CICS and DB2 active, what is very handy.

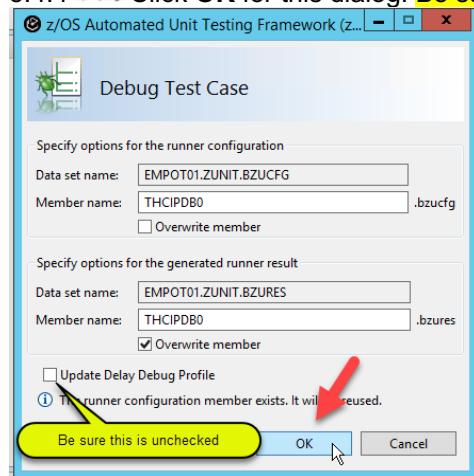
- 5.4.1 ► Close all active windows by pressing **Ctrl+Shift+F4**

- 5.4.2 ► Right click on **EMPOT01.POT.COBOL(HCIPDB01).cbl** and select **z/OS Automated Unit Testing Framework (zUnit) > Debug Test Case...**

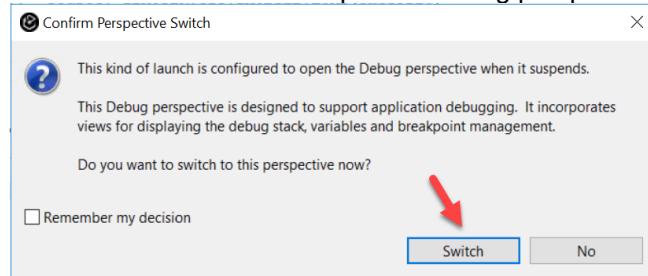


5.4.3 ► Click Yes if the dialog below opens

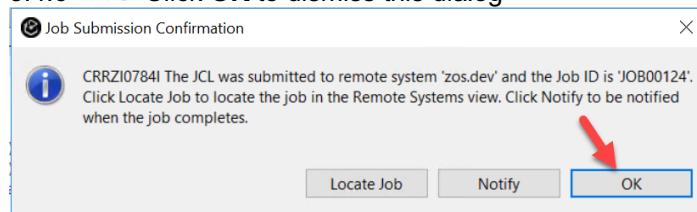
5.4.4 ► Click **OK** for this dialog. Be sure that Update Delay Debug Profile is un-checked.



5.4.5 ► Click **Switch** to open the debug perspective.

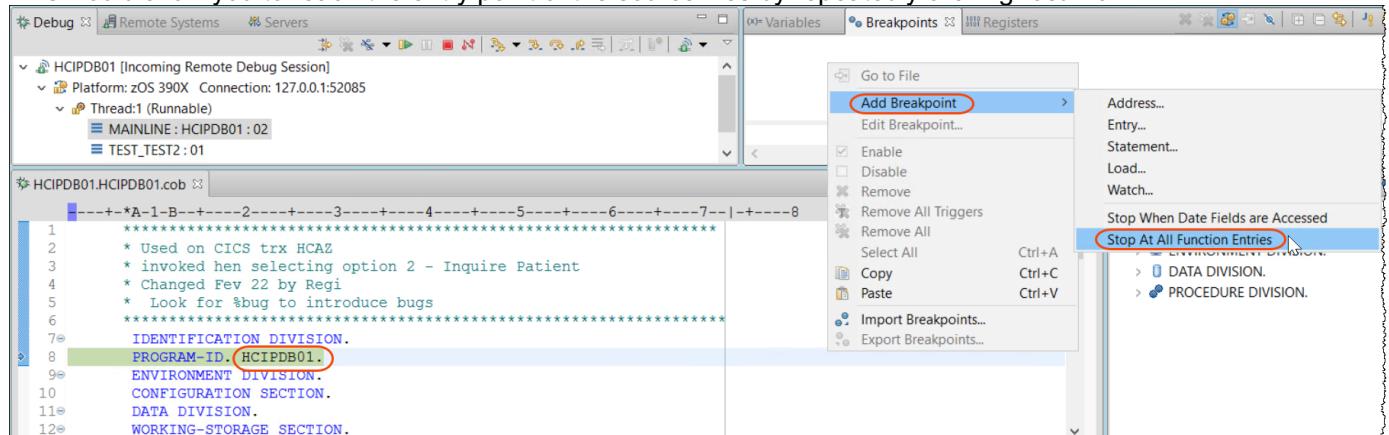


5.4.6 ► Click **OK** to dismiss this dialog

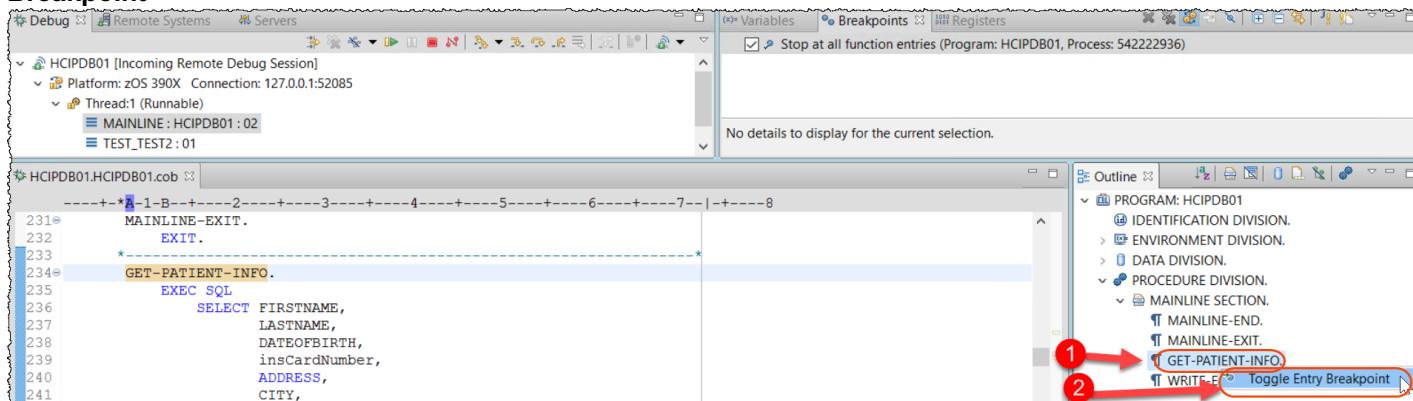


5.4.7 Notice that the first program being debugged is the program that you introduced the bug. (**HCIPDB01**) The execution is stopped BEFORE that program starts. You can add some breakpoints before running.

► Using the Breakpoints view, right click and select **Add Breakpoints > Stop At All Functions Entries**. This would allow you to reach the entry point of the source files by repeatedly clicking resume.



5.4.8 ► Using the Outline view to navigate to **GET-PATIENT-INFO** and right click and select **Toggle Entry Breakpoint**



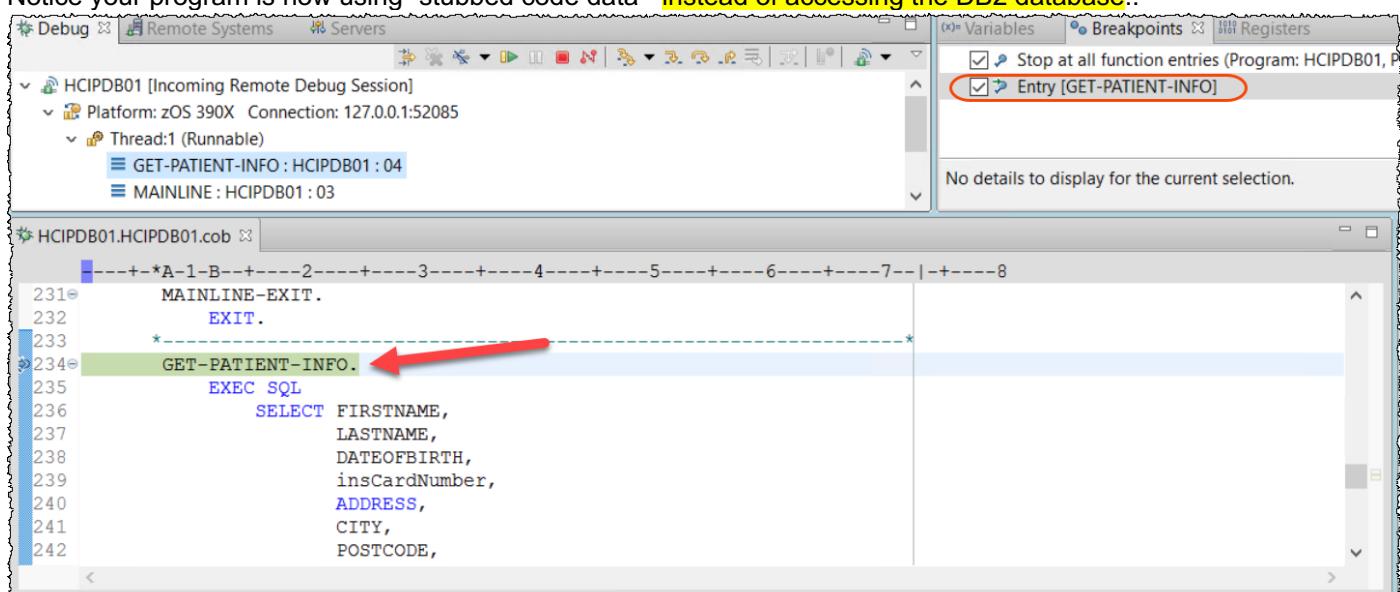
5.4.9 A breakpoint is created on the **EXEC SQL SELECT** statement. When the program runs it will stop there.

```
HCIPDB01.HCIPDB01.cob
-----+*A-1-B---2---+---3---+---4---+
231  MAINLINE-EXIT.
232      EXIT.
233 *
234  GET-PATIENT-INFO.
235      EXEC SQL
236          SELECT FIRSTNAME,
237              LASTNAME,
238              DATEOFBIRTH,
239              insCardNumber,
240              ADDRESS,
241              CITY,
```

5.4.10 ► Click on icon or press **F8** to resume the execution



5.4.11 The execution will halt at the **SQL SELECT** statement due the breakpoint you added.
Notice your program is now using “stubbed code data” instead of accessing the DB2 database..



5.4.12 ► Click on icon or press F5 to Step Into the code until you find the statement below.

► Move the mouse to : DB2-PATIENT-ID and see its contents

Again, notice that we get DB2 data without going to the database, but using the stub recorded earlier (Play Back file).

```

*--A-1-B---2---+---3---+---4---+---5---+---6---+---7---|-----8
      FROM PATIENT
      WHERE PATIENTID = :DB2-PATIENT-ID
      END-EXEC.

Evaluate SQLCODE
  When 0
    MOVE '00' TO CA-RET
  When 100
    MOVE '01' TO CA-RET
  When -913
    MOVE '01' TO CA-RET
  When Other
    MOVE '90' TO CA-RET

```

5.4.13 ► Keep clicking on icon or press F5 to Step Into the code until you see the bug that you had introduced.

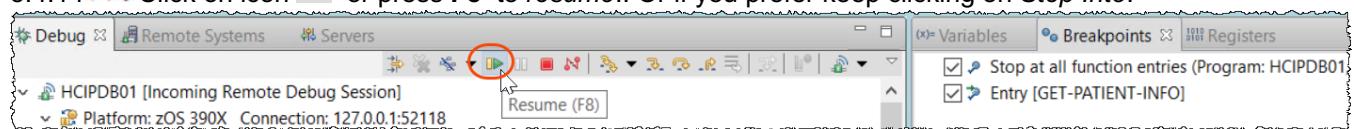
► You also can see the BUG that you introduced. Move the mouse to CA-FIRST-NAME to see the field content (**Ralph**), which will be replaced by "**BAD NAME**".

```

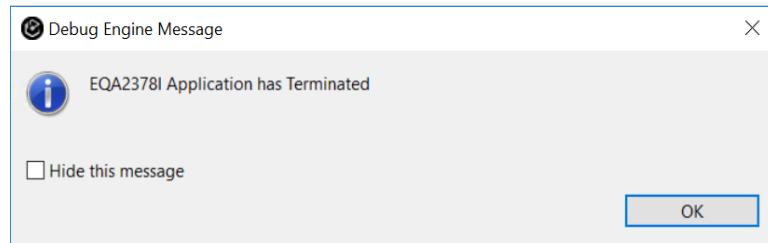
*--A-1-B---2---+---3---+---4---+---5---+---6---+---7---|-----8
      END-Evaluate.
      *%bug -- the line below will introduce a BUG
      *-----
      MOVE "BAD NAME" to CA-FIRST-NAME
      *----- EXIT.
      *----- COPY HCERRSPD.
      *-----
      * Procedure to write error m
      *   message will include Dat
      *   Medication Id and SQLCOD

```

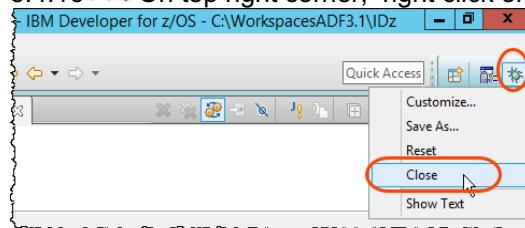
5.4.14 ► Click on icon or press **F8** to resume.. Or if you prefer keep clicking on *Step Into*.



5.4.15 ► You will see that the debug ends when you have the dialog below. Click **OK**



5.4.16 ► On top right corner, right click on icon and select **Close** to close the debug perspective



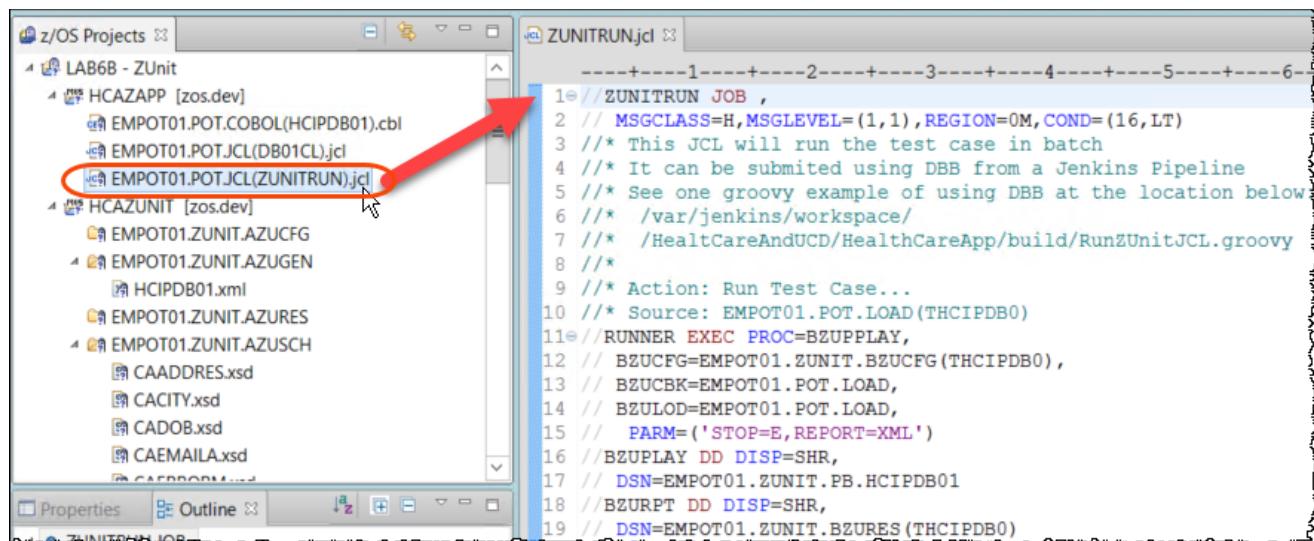
Section 6. Run the unit test from a batch JCL.

Once a test case has been generated and built, it can be executed using a batch run via JCL. This would enable it to be ran as part of an automated process or pipeline..

6.1 Running the unit test from a batch JCL

6.1.1 ► Close any active windows by pressing **Ctrl+Shift+F4**.

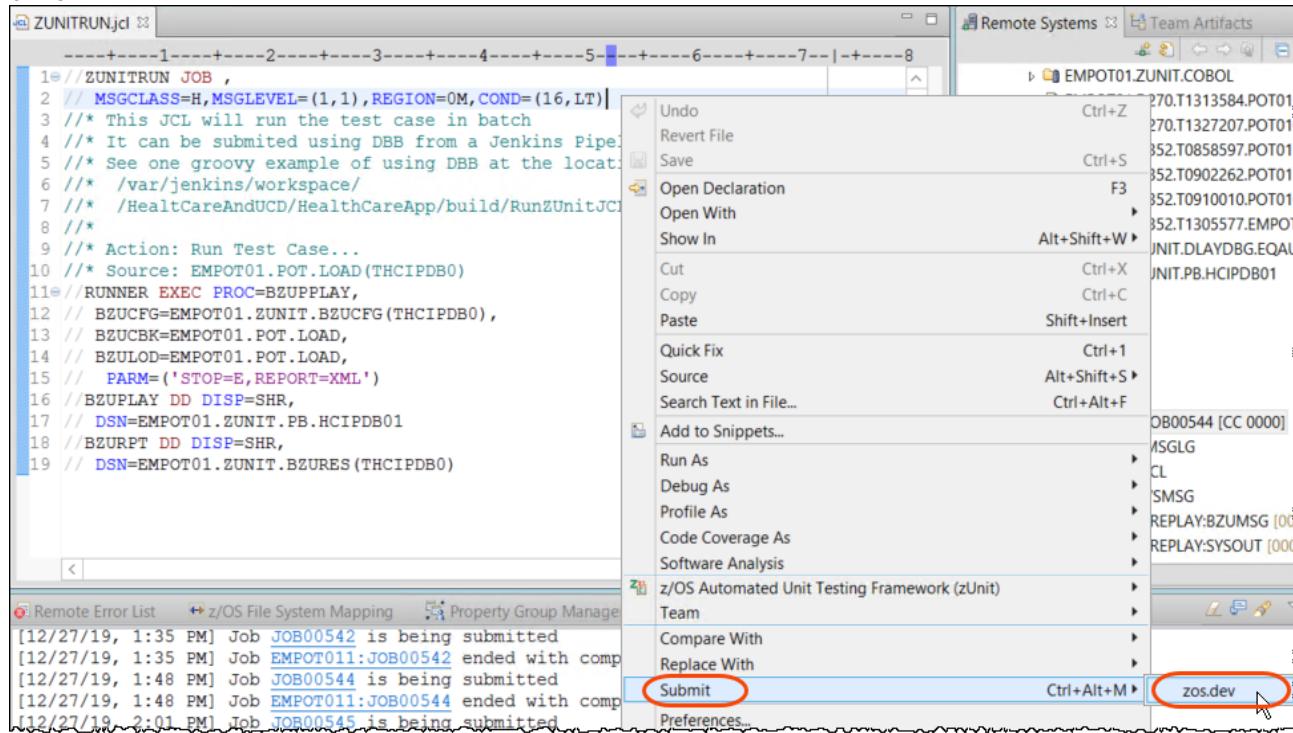
6.1.2 ► On the IDz **z/OS Projects** view, double click **EMPOT01.POT.JCL(ZUNITRUN).jcl** to open it. This JCL will run the test case that you created using a batch job.



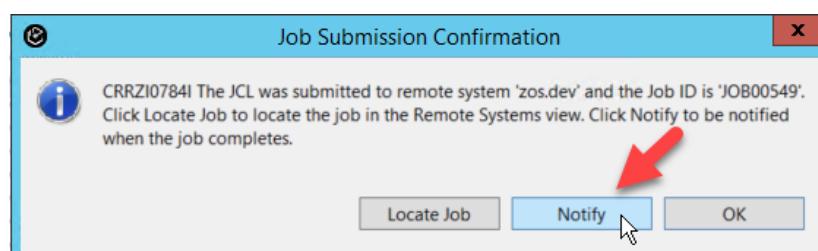
The screenshot shows the IBM Rational Application Developer interface. On the left, the 'z/OS Projects' view displays several projects and their contents. A red arrow points from the project tree to the 'ZUNITRUN.jcl' file in the main editor window. The 'ZUNITRUN.jcl' editor shows the following JCL code:

```
1 //ZUNITRUN JOB ,  
2 // MSGCLASS=H,MSGLEVEL=(1,1),REGION=0M,COND=(16,LT)  
3 //** This JCL will run the test case in batch  
4 //** It can be submitted using DBB from a Jenkins Pipeline  
5 //** See one groovy example of using DBB at the location below  
6 //** /var/jenkins/workspace/  
7 //** /HealtCareAndUCD/HealthCareApp/build/RunZUnitJCL.groovy  
8 //**  
9 //** Action: Run Test Case...  
10 //** Source: EMPOT01.POT.LOAD(THCIPDB0)  
11 //RUNNER EXEC PROC=BZUPPLAY,  
12 // BZUCFG=EMPOT01.ZUNIT.BZUCFG(THCIPDB0),  
13 // BZUCBK=EMPOT01.POT.LOAD,  
14 // BZULOD=EMPOT01.POT.LOAD,  
15 // PARM='STOP=E,REPORT=XML'  
16 //BZUPLAY DD DISP=SHR,  
17 // DSN=EMPOT01.ZUNIT.PB.HCIPDB01  
18 //BZURPT DD DISP=SHR,  
19 // DSN=EMPOT01.ZUNIT.BZURES(THCIPDB0)
```

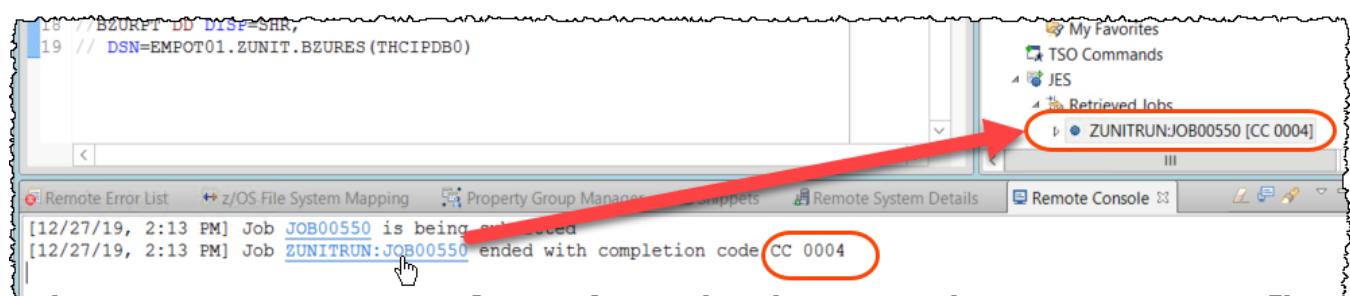
6.1.3



► Right click the editor and select **Submit > zos.dev**.

6.1.4 ► Click **Notify** on the Job Submission Confirmation dialog.6.1.5 You MUST have **004** as return code.

► Click on **ZUNITRUN:JOB00xxx**



6.1.6 Expand **ZUNITRUN:JOB00xxx** and verify the results double clicking on **RUNNER:REPLAY:SYSOUT**.

```

JCL ZUNITRUN.jcl EMPOT01.ZUNITRUN.JOB00183.D000... EMPOT01.ZUNITRUN.JOB00183.D000...
1 TEST_INQ01 STARTED...
2 CALL HCIPDB01
3 DB2_INPT ...
4 DB2_OUTP ...
5 CICS_OE08_HCIPDB01 CHECK VALUES...
6 EXEC CICS RETURN X'0000' L=00227
7 AREA ALLOCATED FOR RECORD COUNT:00000048
8 CICS_OE08_HCIPDB01 SUCCESSFUL.
9 ****
10 AZU2001W THE TEST "INQ01" FAILED DUE TO AN ASSERTION.
11 AZU1101I COMPARE FAILED IN PROCEDURE DIVISION.
12 DATA ITEM NAME : CA-FIRST-NAME OF CA-PATIENT-REQUEST OF DFHCOMMAREA
13 VALUE : BAD NAME
14 EXPECTED VALUE: Ralph
15 ****
16 TEST_INQ01 SUCCESSFUL.
17

```

6.1.7 This JCL could be executed using DBB and part of a Jenkins Pipeline.
You may see an example of a groovy script used by DBB at the USS file below:
`/var/jenkins/workspace/HealtCareAndUCD/HealthCareApp/build/ RunZUnitJCL.groovy`

Under **zos.dev** and **z/OS UNIX Files** look for filter groovy_samples

```

RunZUnitJCL.groovy
1 import com.ibm.dbb.build.CopyToHFS
2 import com.ibm.dbb.build.DBBConstants
3 import com.ibm.dbb.build.JCLEExec
4 ****
5 * Changed Dec 27, 2019 by Regi
6 * The following sample shows how to use JCLEexec API to execute a ZUnit and
7 * display the results in the console.
8 * This sample assumes that user has setup the ZUnit and a JCL to execute the
9 * ZUnit.
10 * This sample requires:
11 *   1. The data set contains the JCL.
12 *   2. The data set contains the output of the ZUnit result.
13 * Sample output:
14 *   Running ZUnit in JCL 'IBMUSER.ZUNIT.JCL(ZUNIDB01)'
15 *   The JCL Job completed with Max-RC CC 0004
16 *   **** Module [J05CMORT] ****
17 * zUnit Test Runner 2.0.0.1 started at 2019-11-06T13:57:22.885...
18 * Test count: 1
19 * Tests passed: 1
20 * Tests failed: 0
21 * Tests in error: 0
22 *
23 ****
24
25/* DBB_CONF must be set for running JCLEexec */
26/* def confDir = System.getenv("DBB_CONF") */
27/* added by regi - was above statement only before */
28def confDir = "/var/dbb/1.0.7/conf"
29
30/* The data set contains the ZUnit JCL */
31/* For example: IBMUSER.ZUNIT.JCL */
32def jcldataset = "IBMUSER.ZUNIT.JCL"

```

Notice that this capability allows to invoke zUnit using pipelines like Jenkins.

Congratulations! You have completed the Lab 2A.

LAB 2B – (OPTIONAL) Using IBM zUnit to Unit Test a COBOL/DB2 batch program (60 minutes)

Updated June 25, 2021 by Regi –(reviewed by Wilbert Kho)

This lab will take you through the steps of using the automated unit testing ([zUnit](#)) capabilities of [IBM Developer for z](#) (IDz) to create a unit test case for a COBOL/DB2 batch program. This enables the testing of just a single program using a JCL being executed. This is done by stubbing out DB2 calls, enabling the program to be tested without a DB2 environment being active. This enables a developer to test early without using DB2 and necessary bindings.

In this lab you will record interaction with a COBOL/DB2 program via JCL execution and import the recorded data into the unit test case generation process. You will build and run the unit test, make a change to the COBOL/DB2 program, and rerun the unit test.

Overview of development tasks

To complete this tutorial, you will perform the following tasks:

PART #1 – Unit test on COBOL/DB2 program DB2BATCH and introduce a bug

1. **Run the COBOL/DB2 batch program using JCL**
→ You will submit a JCL to execute a COBOL/DB2 program that calls a COBOL subprogram to print a small report.
2. **Use zUnit to record the batch execution.**
→ You will record the batch execution using the COBOL/DB2 program and subprograms.
3. **Generate, build, and run the unit test generated program**
→ You will compile and link-edit the generated unit test program, followed by running the unit test.
4. **Modify the COBOL/DB2 program (introduce a bug) and rerun the unit test**
→ You will modify the COBOL/DB2 program introduce a bug, rerun the unit test, and observe the failure of the test case.

PART #2 – Fix COBOL/DB2 program DB2BATCH and re-run the Unit test

5. **Run the batch program using the provided JCL and verify the bug .**
→ You will run the Batch JCL and observe the bug on the printed report..
6. **Use IDz to fix the bug, recompile the COBOL/DB2 program**
→ The bug is fixed using IDz, program fixed is rebuilt
7. **Rerun the zUnit and verify that the bug is eliminated**
→ When the zUnit test case is executed you can verify that the program is fixed.

PART #1 – Unit test on program DB2BATCH and introduce a bug

Section 1. Run the COBOL/DB2 batch program using JCL

You will submit a provided JCL to execute on z/OS and verify the report produced by the second program that is invoked by a dynamic call.

Below it is showing the COBOL program DB2BATCH invoking the program DB2PRT that prints the report

The screenshot shows three windows:

- DB2BATCH.cbl:** Contains COBOL code for performing an SQL query on the RBAROSA.EMPL table to calculate departmental averages and calling the DB2PRT program to print the results.
- DB2PRT.cbl:** Contains COBOL code for opening a report file, printing header information, writing report lines, and closing the report file.
- IBMUSER.BATCHRUN.JOB00097.D000106?.spool:** Shows the printed report output, which is a table of departmental performance statistics.

Scenario: COBOL Batch/DB2

DEPT	PERF-AVG	PERF-MIN	PERF-MAX	HOURS-AVG	HOURS-MIN	HP
ACC	.00	8.00	8.00	15.99	15.99	1
FIN	.00	3.00	9.00	22.69	32.45	1
MKT	.00	1.00	3.00	22.82	32.41	1
R&D	.00	1.00	1.00	22.82	32.41	1
REG	.00	9.00	9.00	26.75	26.75	1
N/A	.00	.00	.00	35.43	.00	1
8						

1.0 Connect to z/OS using IDz

1.0.1 Start *IBM Developer for z Systems* version 15 if it is not already started otherwise go to step 1.1.1

- ▶ Using the desktop double click on **IDz V15** icon.
- ▶ Verify that the message indicates that it is Version **15.0.1**

IMPORTANT -> This icon will start an eclipse workspace that has already some definitions required for this lab. PLEASE DO NOT start IDz using other way than this icon.



1.1 Submit a provided JCL for execution

You will use IDz to submit a provided JCL .

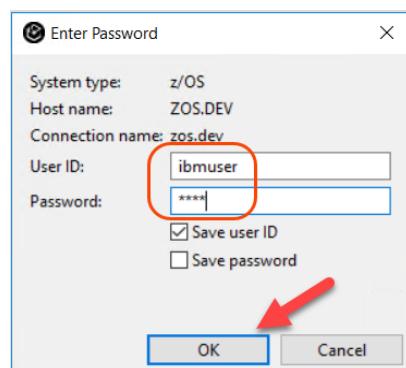
- 1.1.1 ► Open the **z/OS Projects** perspective by selecting
Window > Perspective > Open Perspective > z/OS Projects

Nothing will happen you are already at this [perspective].

- 1.1.2 On this lab you will use userid **ibmuser** . and password **sys1**.
If you are connected as **ibmuser**, jump to step 1.1.4 otherwise.

► Using *Remote Systems* view, right click on **zos.dev** and select **Disconnect** and then right click on **zos.dev** again and select **Connect**

- 1.1.3 ► Type **ibmuser** as userid and **sys1** as password.
The userid and password can be any case; don't worry about having it in UPPER case.
Click **OK** to connect to z/OS.



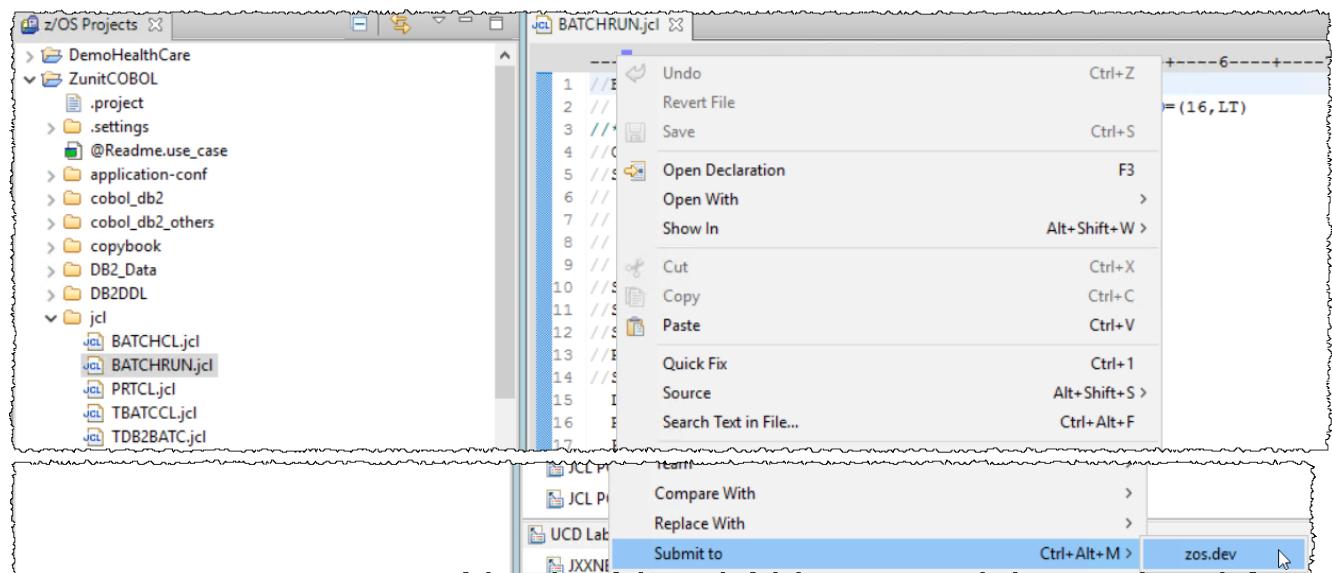
- 1.1.4 ► Under **ZunitCOBOL** expand **jcl** and double click on **BATCHRUN.jcl** to edit the JCL that will be submitted for execution.

```

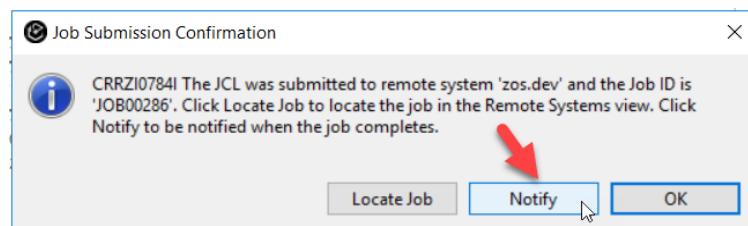
//> BATCHRUN JOB,
//  MSGCLASS=H,MSGLEVEL=(1,1),TIME=1440,REGION=70M,COND=(16,LT)
///* Execute DB2BATCH Batch program
//CURSRAV4 EXEC PGM=IKJEFT01,DYNAMNBR=20
//STEPLIB DD DSN=IBMUUSER.POT.LOAD,DISP=SHR
//          DD DSN=DSNB10.SDSNLOAD,DISP=SHR
//          DD DSN=DSNB10.DBBG.RUNLIB.LOAD,DISP=SHR
//          DD DISP=SHR,DSN=B2U100.SBZULOAD
//          DD DISP=SHR,DSN=EQAE10.SEQAMOD
//SYSPRINT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//SYSPRINT DD SYSOUT=**
//RPTPRINT DD SYSOUT=*,DCB=(RECFM=FBA,BLKSIZE=1330)
//SYSIN DD *
DSN SYSTEM(DBBG)
RUN PROGRAM(DB2BATCH) -
PLAN(DB2BATCH)
END
/*

```

- 1.1.5 ► Right click on the JCL edited and select **Submit to > zos.dev**

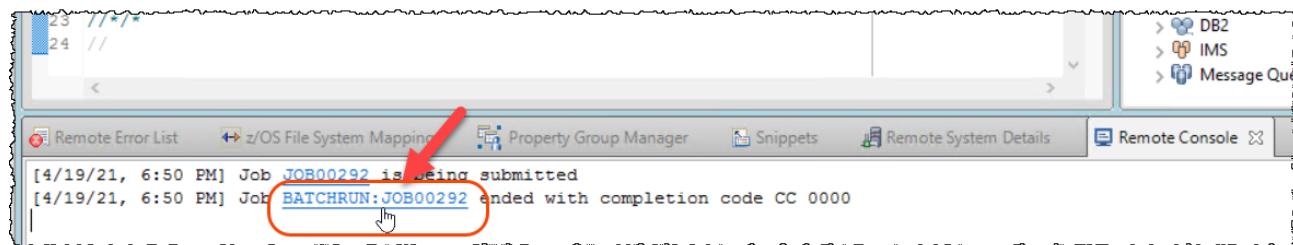


- 1.1.6 ► Click **Notify** to be notified when the execution is complete.



1.1.7 Under **Remote Console**, you will be notified when execution is completed.

► Once the execution ends, click on the link **BATCHRUN:JOB00xxx** (where 00xxx is a number) to point to the Job output. This number vary depending on z/OS.



1.1.8 ► Under **Remote Systems** view scroll down, expand **JES > Retrieved Jobs > BATCHRUN:JOB00xxx** and double click **CURSRAV4::RPTPRINT** step and you will see the report produced by the **DB2PRT** called COBOL subprogram.

Tip: If there is no jobs under "Retrieved Jobs", is because you did not click on link as stated at 1.1.7. You can also see this output once you right click on "My Jobs" under **JES** and select **Refresh**.

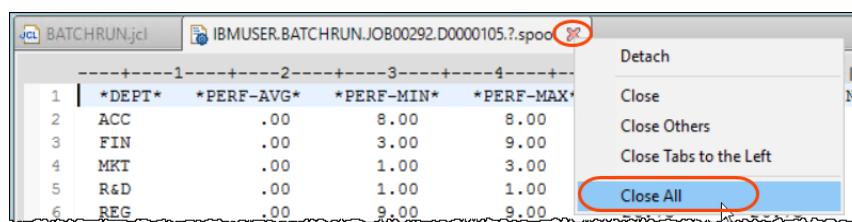
The screenshot shows the 'Remote Systems' interface. On the left, a terminal window displays a spool file named 'BATCHRUN.jcl' with the following data:

	DEPT	*PERF-AVG*	*PERF-MIN*	*PERF-MAX*	*HOURS-AVG*	*HOURS-MIN*	*HOURS-MAX*
1	*DEPT*	.00	8.00	8.00	15.99	15.99	15.99
2	ACC	.00	8.00	8.00	15.99	15.99	15.99
3	FIN	.00	3.00	9.00	22.69	32.45	8.89
4	MKT	.00	1.00	3.00	22.82	32.41	13.23
5	R&D	.00	1.00	1.00	22.82	32.41	13.23
6	REG	.00	9.00	9.00	26.75	26.75	26.75
7	N/A	.00	9.00	9.00	35.45	35.45	35.45

A yellow callout bubble points to the first row with the text: "Notice on first line the DEPT value of ACC. When you introduce a bug this value will be different."

On the right, the 'Retrieved Jobs' tree view shows the job 'BATCHRUN:JOB00292 [CC 0000]' expanded, with the 'CURSRAV4::RPTPRINT' step selected and highlighted with a red box.

1.1.9 ► Close all the opened editors using **Ctrl + Shift + F4**, Or right click on the and choose **Close all**.



What have you done so far?



You submitted a JOB to be executed under batch. This job uses two subprograms being invoked. One of the programs invoked dynamically prints a small report from data retrieved from a DB2 table.

Section 2 – Use zUnit to record the batch execution.

Using IDz you will record the COBOL/DB2 batch execution via JCL.

The main COBOL program (**DB2BATCH**) reads from a DB2 table and pass some data to be printed by another dynamically called COBOL subprogram (**DB2PRT**).

Notice that the main COBOL program also invokes a third COBOL program (**REGI0C**) using a static call.

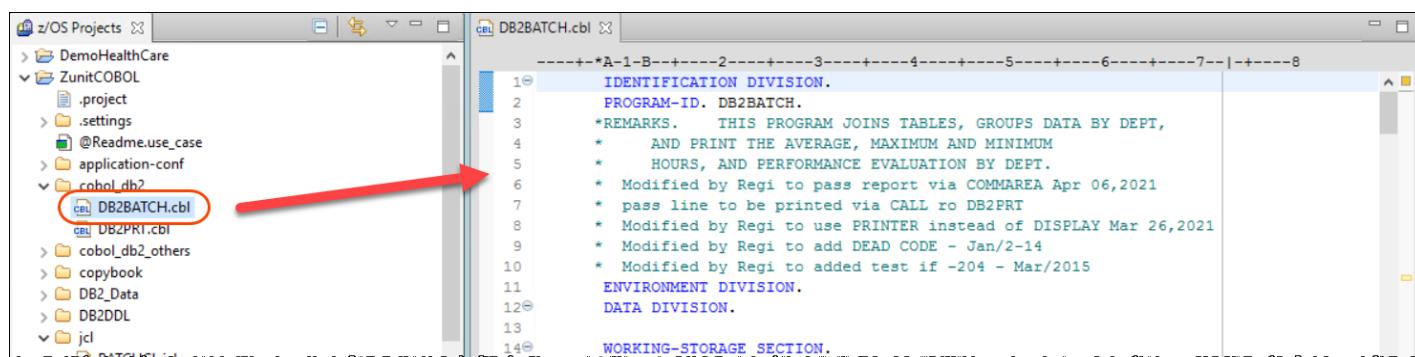
2.1 Understanding the main COBOL program that reads from DB2 table

The main COBOL code that reads the DB2 tables is the program **DB2BATCH**

2.1.1 Using z/OS Projects view

double click on **DB2BATCH.cbl** under **ZunitCOBOL/cobol_db2**.

This is the program that you will update later on..



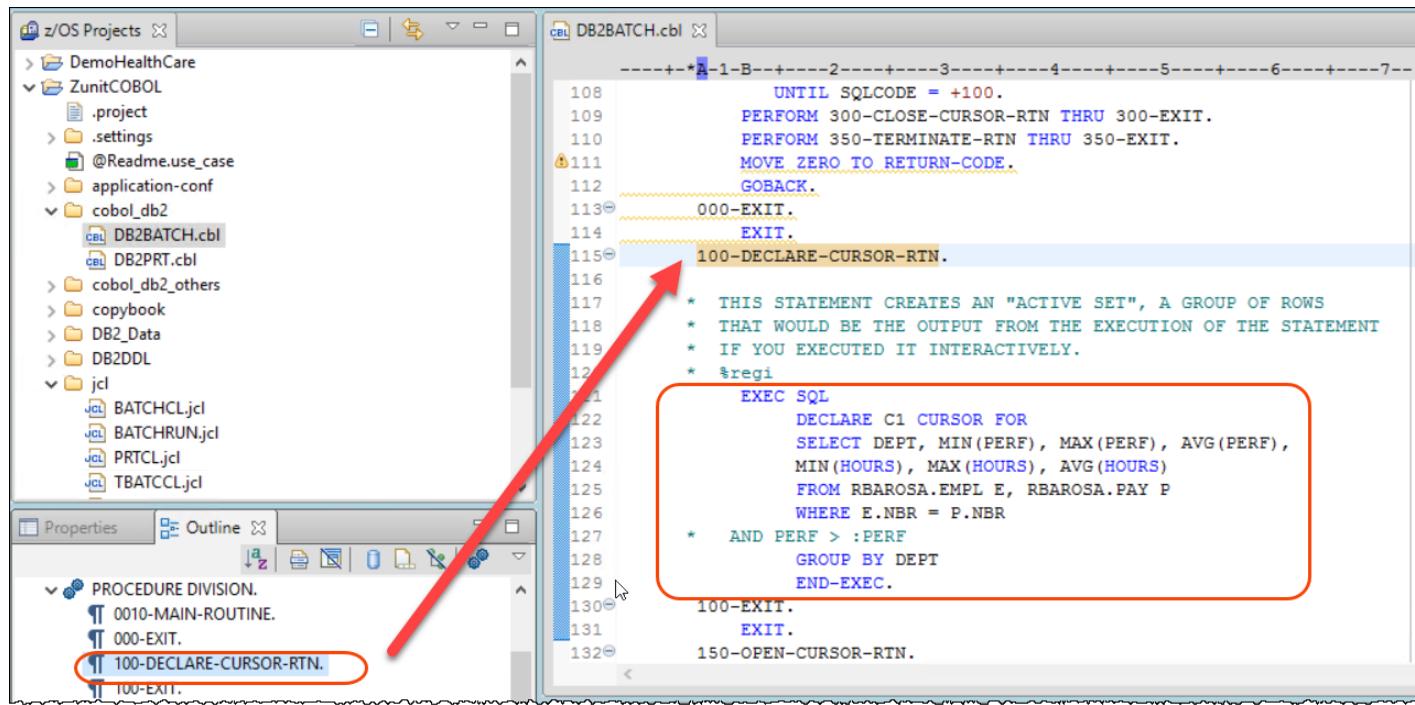
```

1  IDENTIFICATION DIVISION.
2  PROGRAM-ID. DB2BATCH.
3  *REMARKS.  THIS PROGRAM JOINS TABLES, GROUPS DATA BY DEPT,
4  *      AND PRINT THE AVERAGE, MAXIMUM AND MINIMUM
5  *      HOURS, AND PERFORMANCE EVALUATION BY DEPT.
6  *      Modified by Regi to pass report via COMMAREA Apr 06,2021
7  *      pass line to be printed via CALL ro DB2FRT
8  *      Modified by Regi to use PRINTER instead of DISPLAY Mar 26,2021
9  *      Modified by Regi to add DEAD CODE - Jan/2-14
10 *      Modified by Regi to added test if -204 - Mar/2015
11 ENVIRONMENT DIVISION.
12* DATA DIVISION.
13*
14* WORKING-STORAGE SECTION.

```

2.1.2 Using the Outline view on left expand PROCEDURE DIVISION and click on 100-DECLARE-CURSOR-RTN.

This is where the DB2 select statement is defined. Notice that the program will execute a DB2 table join and scan the rows resulting from that join. Later on you will introduce a bug in this program.



The screenshot shows the IBM Rational Application Developer interface. On the left, the 'z/OS Projects' view displays a project structure for 'DemoHealthCare' under 'ZunitCOBOL'. The 'Outline' view on the right shows the structure of the 'DB2BATCH.cbl' program, specifically the 'PROCEDURE DIVISION' section.

A red arrow points from the '100-DECLARE-CURSOR-RTN.' entry in the 'Outline' view to the corresponding code line in the main editor window.

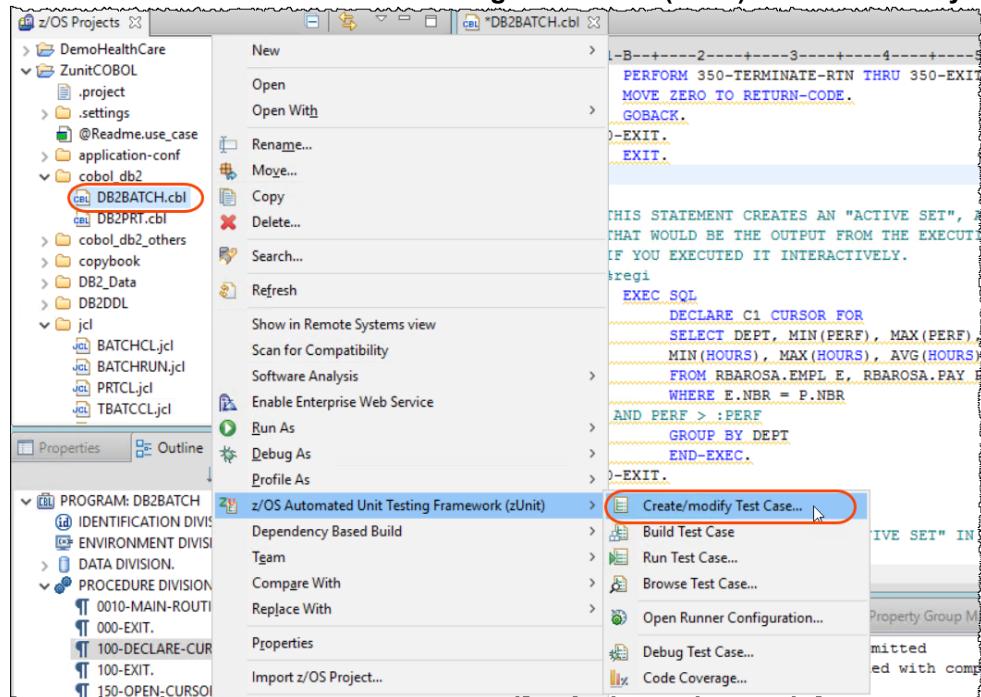
```

-----+-----1-----2-----3-----4-----5-----6-----7-----
108      UNTIL SQLCODE = +100.
109      PERFORM 300-CLOSE-CURSOR-RTN THRU 300-EXIT.
110      PERFORM 350-TERMINATE-RTN THRU 350-EXIT.
111      MOVE ZERO TO RETURN-CODE.
112      GOBACK.
113      000-EXIT.
114      EXIT.
115      100-DECLARE-CURSOR-RTN.
116
117      * THIS STATEMENT CREATES AN "ACTIVE SET", A GROUP OF ROWS
118      * THAT WOULD BE THE OUTPUT FROM THE EXECUTION OF THE STATEMENT
119      * IF YOU EXECUTED IT INTERACTIVELY.
120      * %regi
121      EXEC SQL
122          DECLARE C1 CURSOR FOR
123              SELECT DEPT, MIN(PERF), MAX(PERF), AVG(PERF),
124                  MIN(HOURS), MAX(HOURS), AVG(HOURS)
125                  FROM RBAROSA.EMPL E, RBAROSA.PAY P
126                      WHERE E.NBR = P.NBR
127
128          * AND PERF > :PERF
129          GROUP BY DEPT
130          END-EXEC.
131
132      100-EXIT.
133      EXIT.
134      150-OPEN-CURSOR-RTN.

```

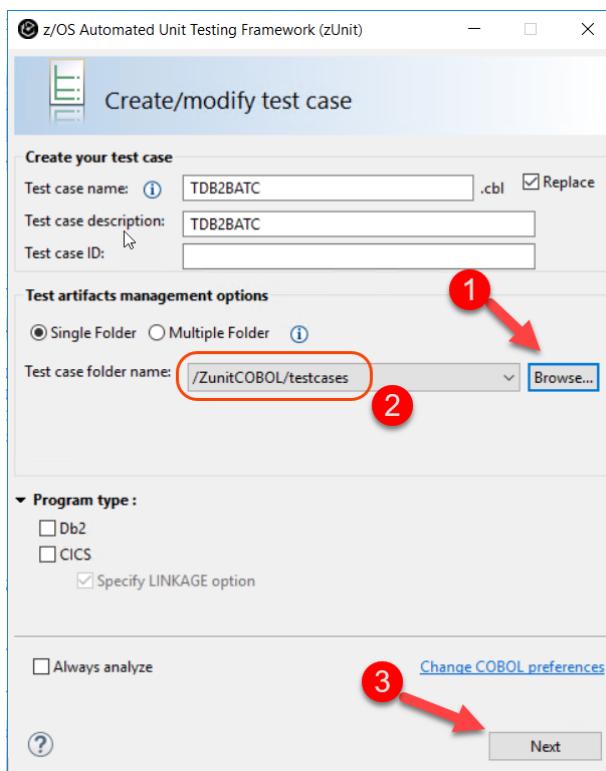
2.2 Recording the batch JCL execution

2.2.1 To start the recording, right click on **DB2BATCH.cbl** and select **z/OS Automated Unit Testing Framework (zUnit)-> Create/modify Test Case..**

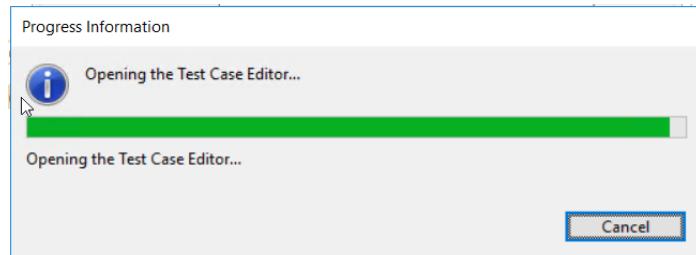


2.2.2 This opens a dialog where you can name your test case.

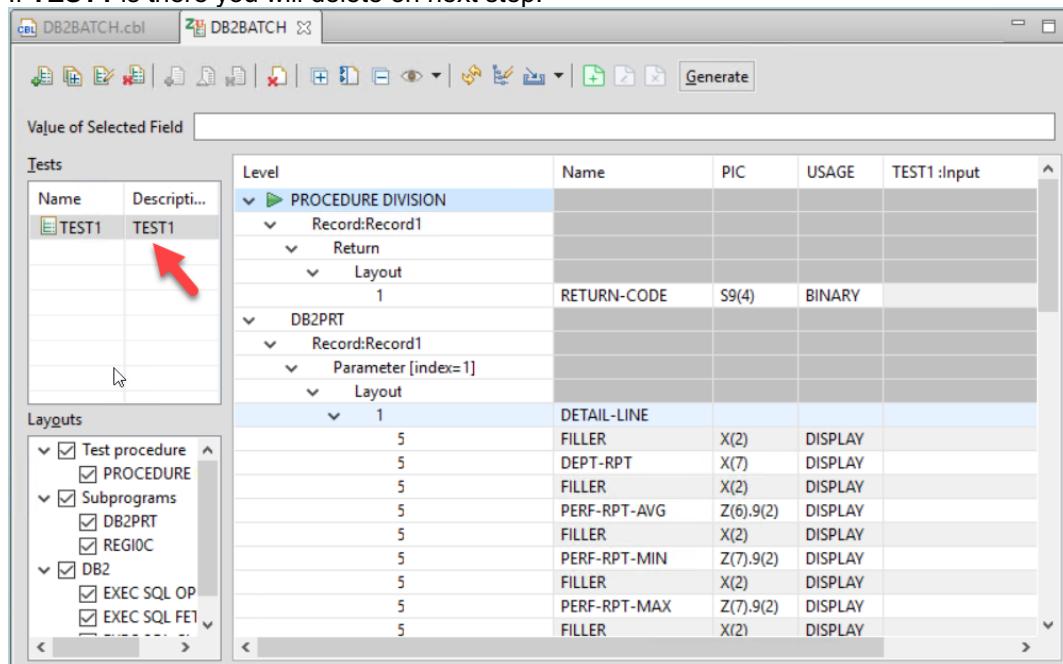
Using the Browse button select **/ZunitCOBOL/testcases** as folder name and click **Next**.



2.2.3 This operation will generate the test data layout on the local workspace and open the *Test Case Editor*.



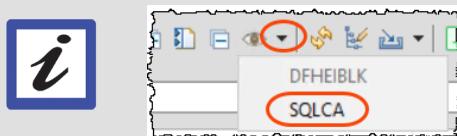
2.2.4 The *Test Case Editor*, as shown below. TEST1 may or may not be on your screen. If TEST1 is there you will delete on next step.



Understanding the test case editor

8. The bottom left box summarizes all the input output variable structures – In our exercise, the main COBOL program(DB2BATCH) has 2 Subprograms (DB2PRT and REGI0C) and the LINKAGE Section of those programs are displayed. Also the areas used by DB2 statements are listed.

9. The DB2 SQLCA area can be displayed once is selected as below.

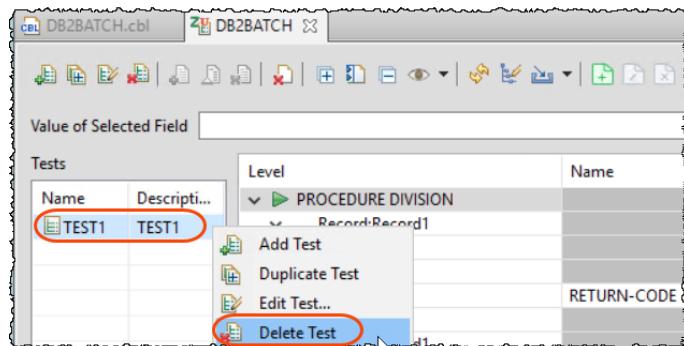


The test entry Input and Expected output columns represent the flow of data into and out of the main program, that is, the program being tested by zUnit. When data is added to these columns in a subroutine, the flow of data is:

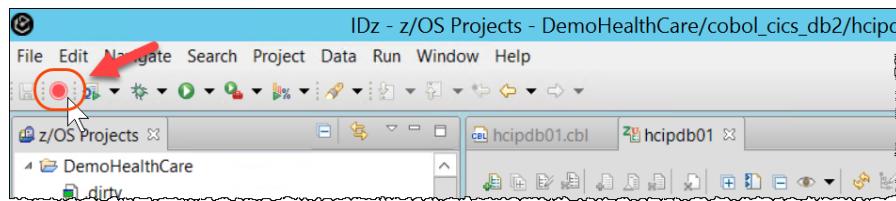
Input: data passed into the main program: that is, what is passed to the main program when the subroutine completes execution.

Expected output: data passed out of the main program: that is, what is passed to the subroutine when it is called by the main program.

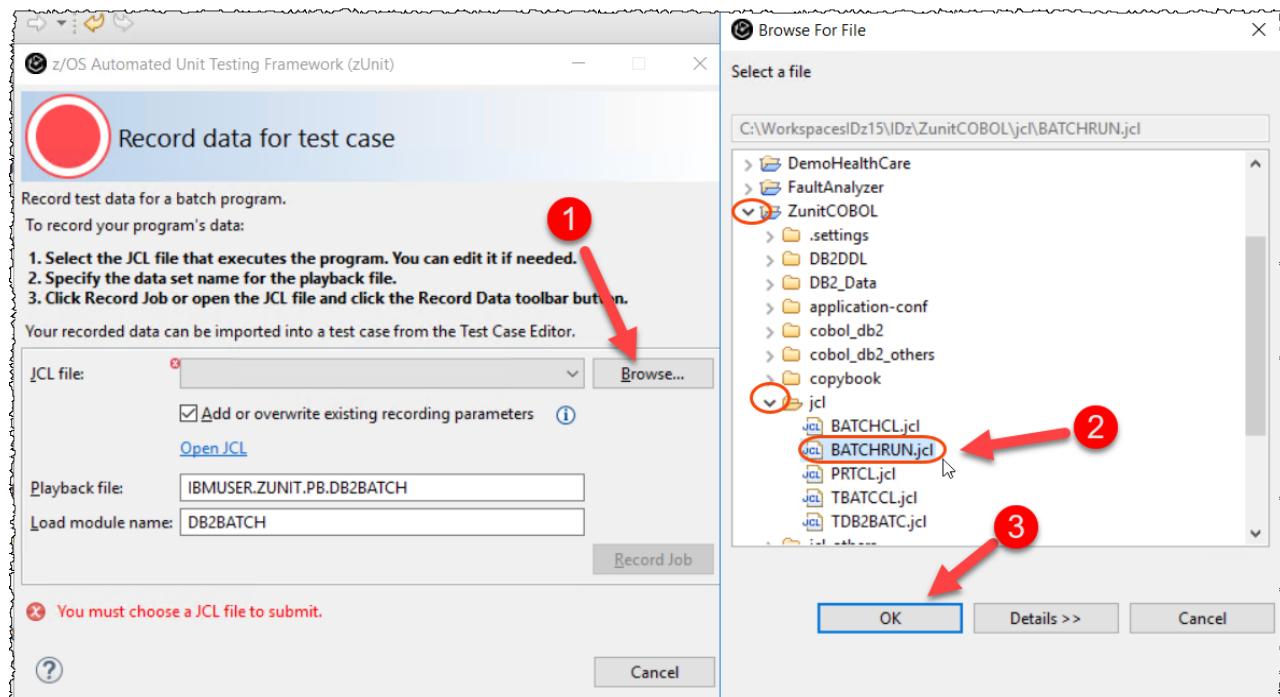
2.2.5 ► Since we will be recording the program execution, delete the **TEST1** or any other entry (if it exists) by right clicking and selecting **Delete Test**



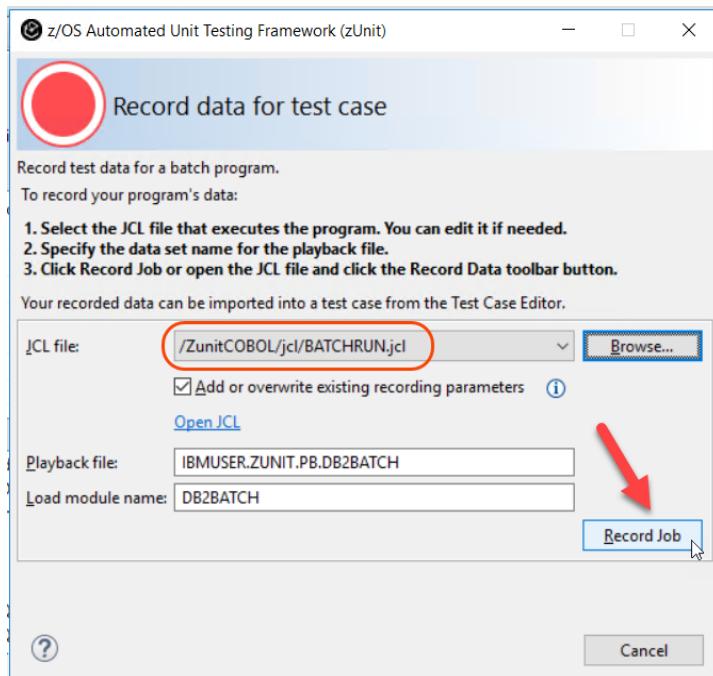
2.2.6 ► To record from a JCL batch execution into the test case, click the **Record** button on the IDz toolbar.



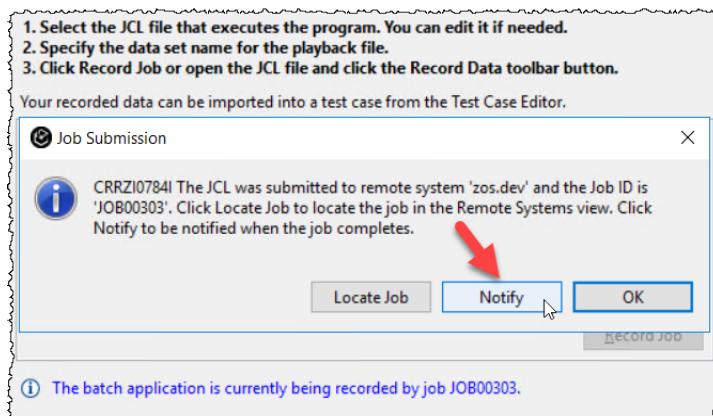
2.2.7 ► In the dialog that comes up, use the **Browse** button to select **BATCHRUN.jcl** under **ZunitCOBOL/jcl** and click **OK**.



2.2.8 ► Click on **Record Job** to submit the JCL for execution.

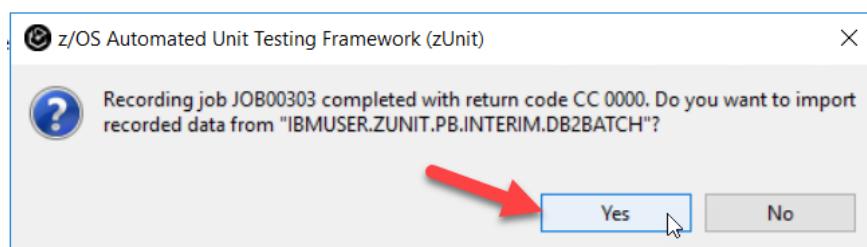


2.2.9 ► Click **Notify** for the dialog below. The JCL will be submitted to run on z/OS.

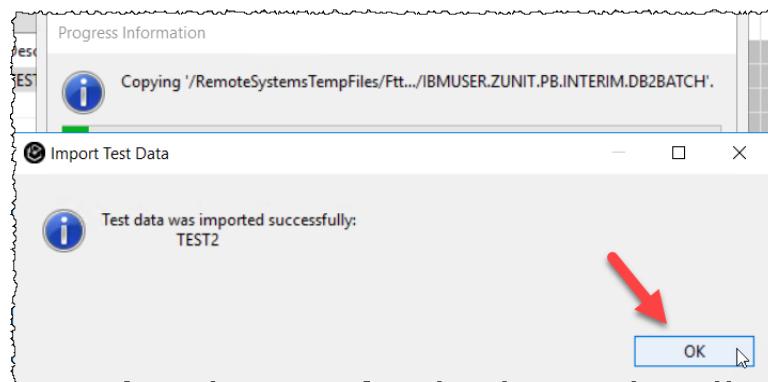


2.2.10 When the job is completed the dialog below is displayed.

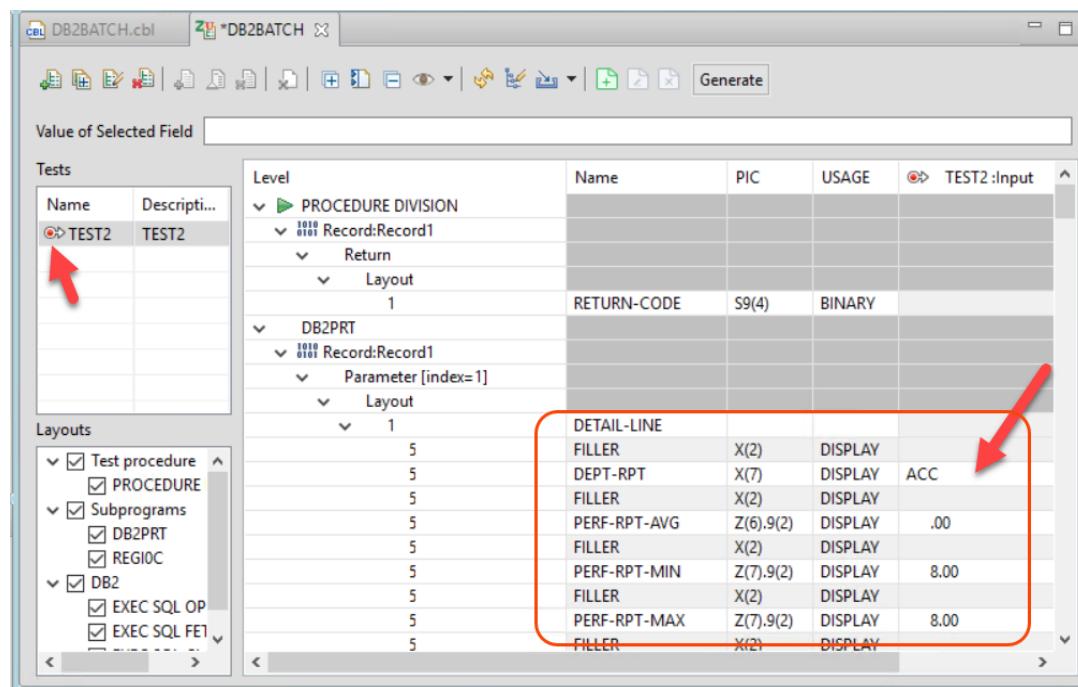
► Click **Yes** to create the playbackfile and import the test data.



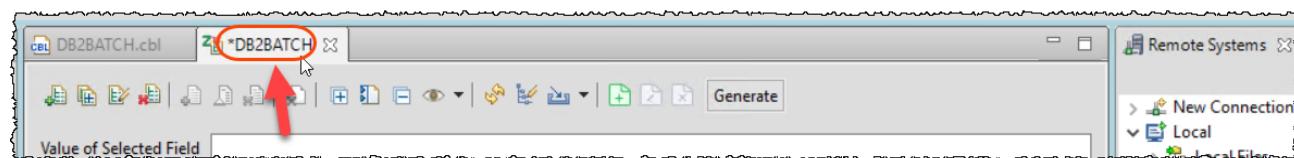
2.2.11 ► Click **OK** after the data is successfully imported to the test data and playback file.



2.2.12 The test case editor is populated with the data recorded



2.2.13 ► Double click on the **DB2BATCH** title to enlarge the view.



2.2.14 You now see a new test created in the editor and populated with the values from the live run.

► Scroll down the editor and notice the 6 rows of data passed to the **DB2PRT** subprogram.

Level	Name	PIC	USAGE	TEST2:Input	TEST2:Execution
1	DETAIL-LINE				
5	FILLER	X(2)	DISPLAY		
5	DEPT-RPT	X(7)	DISPLAY	REG	REG
5	FILLER	X(2)	DISPLAY		
5	PERF-RPT-AVG	Z(6),9(2)	DISPLAY	.00	.00
5	FILLER	X(2)	DISPLAY		
5	PERF-RPT-MIN	Z(7),9(2)	DISPLAY	9.00	9.00
5	FILLER	X(2)	DISPLAY		
5	PERF-RPT-MAX	Z(7),9(2)	DISPLAY	9.00	9.00
5	FILLER	X(2)	DISPLAY		
5	HOURS-RPT-AVG	Z(7),9(2)	DISPLAY	26.75	26.75
5	FILLER	X(2)	DISPLAY		
5	HOURS-RPT-MAX	Z(7),9(2)	DISPLAY	26.75	26.75
5	FILLER	X(5)	DISPLAY		
5	HOURS-RPT-MIN	Z(7),9(2)	DISPLAY	26.75	26.75
RECEIVED-FROM-C...	9(2)	DISPLAY	0	0	
1	DETAIL-LINE				
5	FILLER	X(2)	DISPLAY		
5	DEPT-RPT	X(7)	DISPLAY	N/A	N/A
5	FILLER	X(2)	DISPLAY		
5	PERF-RPT-AVG	Z(6),9(2)	DISPLAY	.00	.00
5	FILLER	X(2)	DISPLAY		
5	PERF-RPT-MIN	Z(7),9(2)	DISPLAY	.00	.00
5	FILLER	X(2)	DISPLAY		
5	PERF-RPT-MAX	Z(7),9(2)	DISPLAY	.00	.00
5	FILLER	X(2)	DISPLAY		

2.2.15 ► 1 Select **EXEC SQL FETCH (C1)** to position the data layout for this statement.

2 Click on **ACC** . and notice that value is displayed on the 3 **Value of Selected Field**

Level	Name	PIC	USAGE	TEST2:Input	TEST2:Execution
EXEC SQL OPEN [C1]	line=135				
Record:Record1					
LineNumber=135	SQCA				
EXEC SQL FETCH [C1]	line=199				
Record:Record1					
LineNumber=199	INTO				
5	DEPT-TBL	X(3)	DISPLAY	ACC	
5	DEPT-NULL	S9(4)	BINARY	0	
5	PERF-TBL-MIN	S9(4)	BINARY	8	
5	PERF-NULL	S9(4)	BINARY	0	
5	PERF-TBL-MAX	S9(4)	BINARY	8	
5	PERF-NULL	S9(4)	BINARY	0	
5	PERF-TBL-AVG	S9(5)V9(2)	PACKED...	8.00	
5	PERF-NULL	S9(4)	BINARY	0	
5	HOURS-TBL-MIN	S9(5)V9(2)	PACKED...	15.99	
5	HOURS-NULL	S9(4)	BINARY	0	
5	HOURS-TBL-MAX	S9(5)V9(2)	PACKED...	15.99	
5	HOURS-NULL	S9(4)	BINARY	0	
5	HOURS-TBL-AVG	S9(5)V9(2)	PACKED...	15.99	
5	HOURS-NULL	S9(4)	BINARY	0	
SQLCA					
Record:Record2					
LineNumber=199	INTO				
5	DEPT-TBL	X(3)	DISPLAY	FIN	
5	DEPT-NULL	S9(4)	BINARY	0	
5	PERF-TBL-MIN	S9(4)	BINARY	3	
5	PERF-NULL	S9(4)	BINARY	0	
5	PERF-TBL-MAX	S9(4)	BINARY	9	
5	PERF-NULL	S9(4)	BINARY	0	

2.2.16 ►► Press **Ctrl+S** to save any changes.

2.2.17 ►► Double click again on the **DB2BATCH** title to shrink the view.

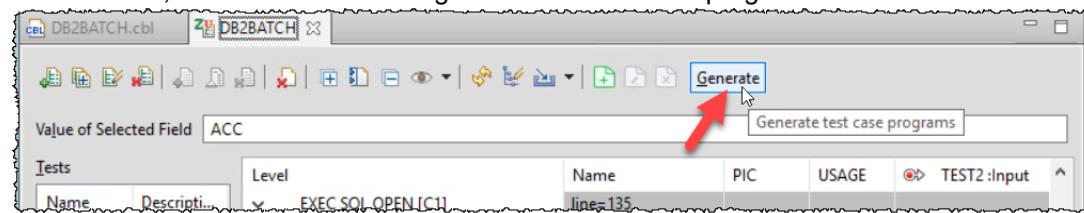


Section 3. Generate, build, and run the unit test generated program.

You will generate, build, and run the unit test for the test case created.

3.1 Generating the COBOL test case programs.

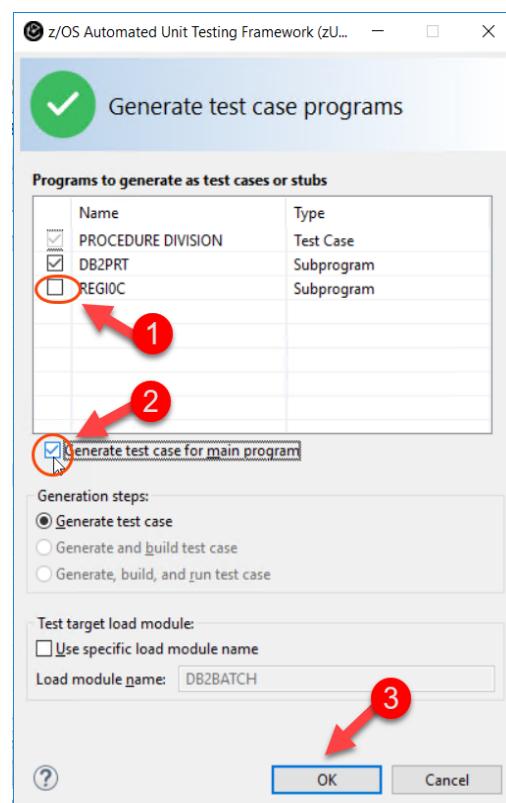
3.1.1 ►► Now that the expected input and output values have been set in the test case editor from the recorded run, click on **Generate** to generate the test case programs.



On the **Generate test case programs** dialog,

3.1.2 ►► Un-select REGI0C (you don't need stubs for this subprogram since it is ready)

►► Select **Generate test case for main program** and then click **OK**.



3.1.3 The COBOL programs that will run the test case are generated under the folder **testcases**.

▶ Expand **testcases** and double click on **TDB2BATC.cbl** to verify the generated programs.

Notice that in fact 7 COBOL programs were generated from this test case . This can be seen on the **Outline** view

You could navigate to each program if time allows, note that NONE of these programs will require CICS or DB2 to be executed. All will be executed in batch using JCL.

The screenshot shows the IBM Rational Developer for z/OS interface. On the left, the 'z/OS Projects' view displays a project structure with several subfolders like application-conf, cobol_db2, and testcases. Inside testcases, there are files DB2BATCH.bzucfg, DB2BATCH.json, DB2BATCH.plbck, and TDB2BATC.cbl. A red arrow points from the 'testcases' folder towards the COBOL editor on the right. The COBOL editor displays the source code for TDB2BATC.cbl, which includes comments about the product being IBM Developer for z/OS and the date generated. Below the editor, the 'Outline' view is open, showing a list of generated programs: TEST_TEST2, BZU_TEST, BZU_INIT, BZU_TERM, PGM_DB2PRT, GTMEMRC, and AZU_GENERIC_DB2. These programs are highlighted with a red box.

```

-----+*A-1-B-----+-----2-----+-----3-----+-----4-----+-----5-----+-----6-----+-----7-----
1      PROCESS NODLL,NODYNAM,TEST (NOSEP),NOCICS,NOSQL,PGMN (LU)
2
3      *+
4      *| TDB2BATC
5      *| PRODUCT: IBM DEVELOPER FOR Z/OS
6      *| COMPONENT: IBM Z/OS AUTOMATED UNIT TESTING FRAMEWORK (ZUNIT)
7      *| FOR ENTERPRISE COBOL AND PL/I
8      *| PROGRAM: ENTERPRISE COBOL ZUNIT TEST CASE FOR DYNAMIC RUNNER
9      *| DATE GENERATED: 04/20/2021 18:30
10     *| ID: 2d075945-d9cf-4384-9272-89c7a546ddb4
11
12     *+
13     *| TEST_TEST2
14     *| THIS PROGRAM IS FOR TEST TEST2
15     *+
16
17     IDENTIFICATION DIVISION.
18     PROGRAM-ID. 'TEST_TEST2'.
19     DATA DIVISION.
20     WORKING-STORAGE SECTION.
21     01 PROGRAM-NAME PIC X(8) VALUE 'DB2BATCH'.
22     01 BZ-ASSERT.
23         03 MESSAGE-LEN PIC S9(4) COMP-4 VALUE 24.
24         03 MESSAGE-TXT PIC X(254) VALUE 'HELLO FROM TEST CALLBACK'.
25     01 BZ-P1 PIC S9(9) COMP-4 VALUE 4.
26     01 BZ-P2 PIC S9(9) COMP-4 VALUE 2001.
27     01 BZ-P3 PIC X(3) VALUE 'AZU'.

```

3.2 Generate, build, and run the unit test generated program.

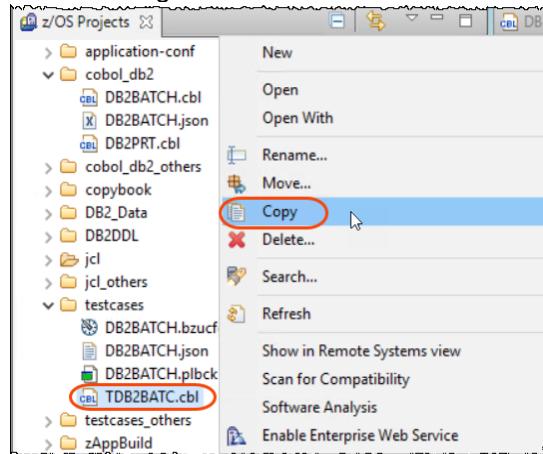
The 7 generated COBOL programs need to be compiled/link edited to be executed. This must be done on z/OS using the COBOL compiler.

To make this easier you could use the IBM DBB (Dependency Based Build) that is part of IDz.

But you also could use the traditional JCL once you moved the generated programs to the z/OS (PDS) to be complied and linked. Here we will show the traditional JCL way.

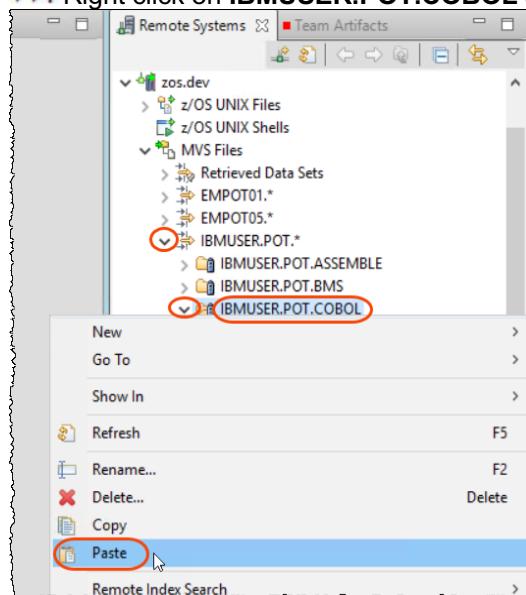
3.2.1 ▶ Use **Ctrl + Shift + F4** to close all opened editors.

3.2.2 ▶ Right click on **TDB2BATC.cbl** and select **Copy**

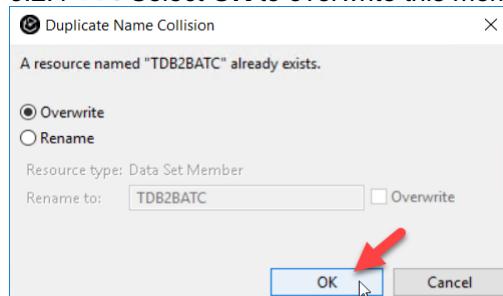


3.2.3 ► Using the Remote systems (on right), expand **IBMUSER.POT.*** and **IBMUSER.POT.COBOL**

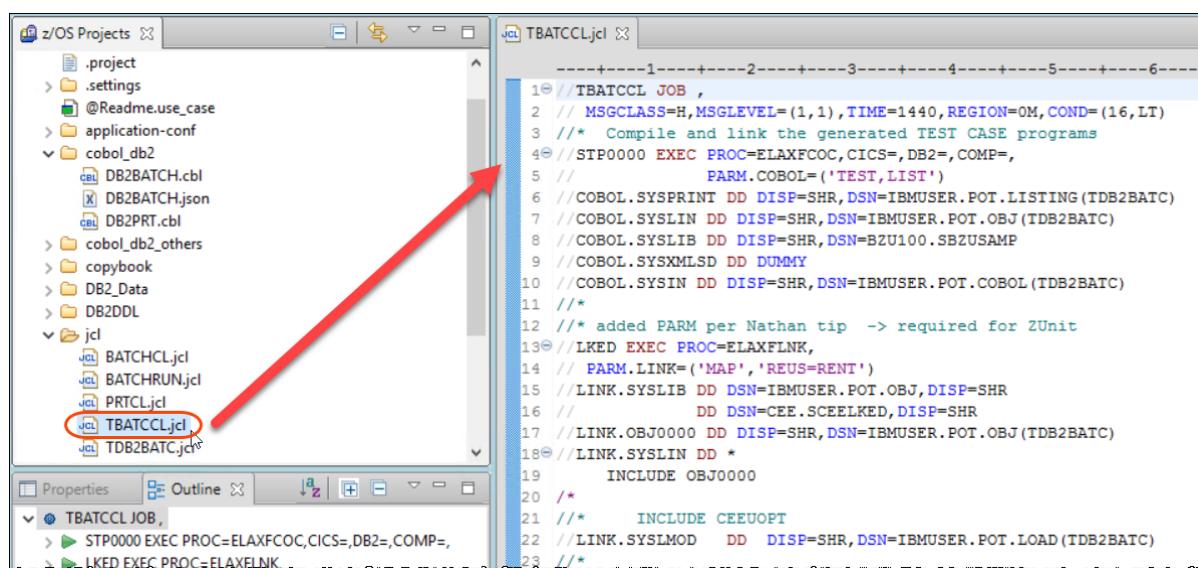
► Right click on **IBMUSER.POT.COBOL** and select **Paste**



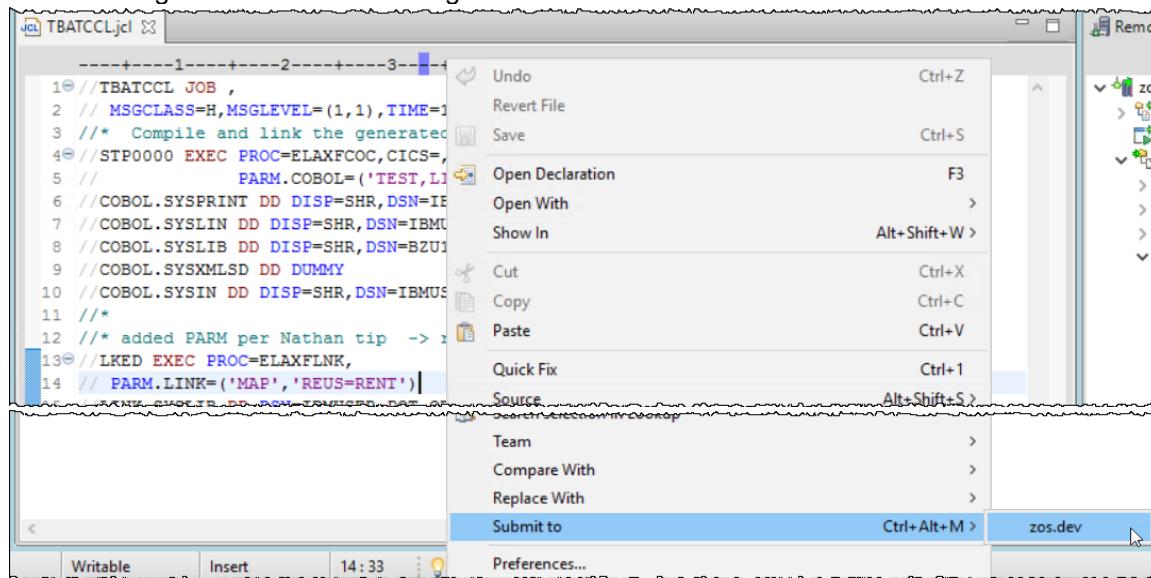
3.2.4 ► Select **OK** to overwrite this member since it already existed on z/OS



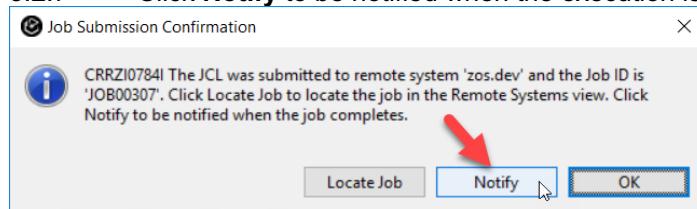
3.2.5 ► Under z/OS Projects expand **ZunitCOBOL/jcl**, and double click on **TBATCCL.jcl**. This JCL will compile and link the 7 programs that will create the test case load module.



3.2.6 ► Right click on the JCL being edited and select **Submit to > zos.dev** for execution on z/OS

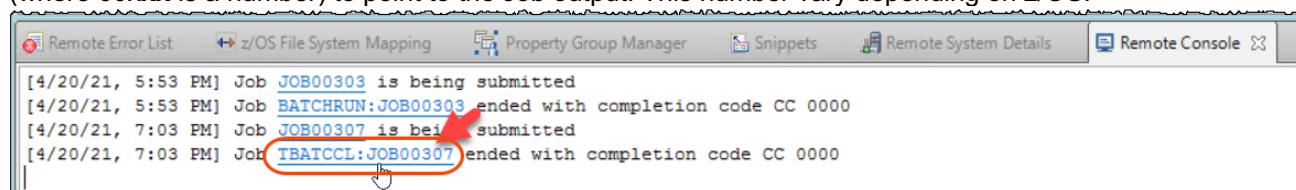


3.2.7 ► Click **Notify** to be notified when the execution is complete.



3.2.8 Under *Remote Console*, you will be notified when execution is completed.

► Once the execution ends, click on the link **TBATCCL:JOB00xxx** (where 00xxx is a number) to point to the Job output. This number vary depending on z/OS.



3.2.9 ► Under *Remote Systems* view expand **TBATCCL:JOB00xxx** and double click **LKED:LINK:SYSPRINT** step.

► Scroll down the report displayed and you will see the test case load module **TDB2BATC** created at **EMPOT.POT.LOAD**.



3.2.10 ► Use **Ctrl + Shift + F4** to close all opened editors.

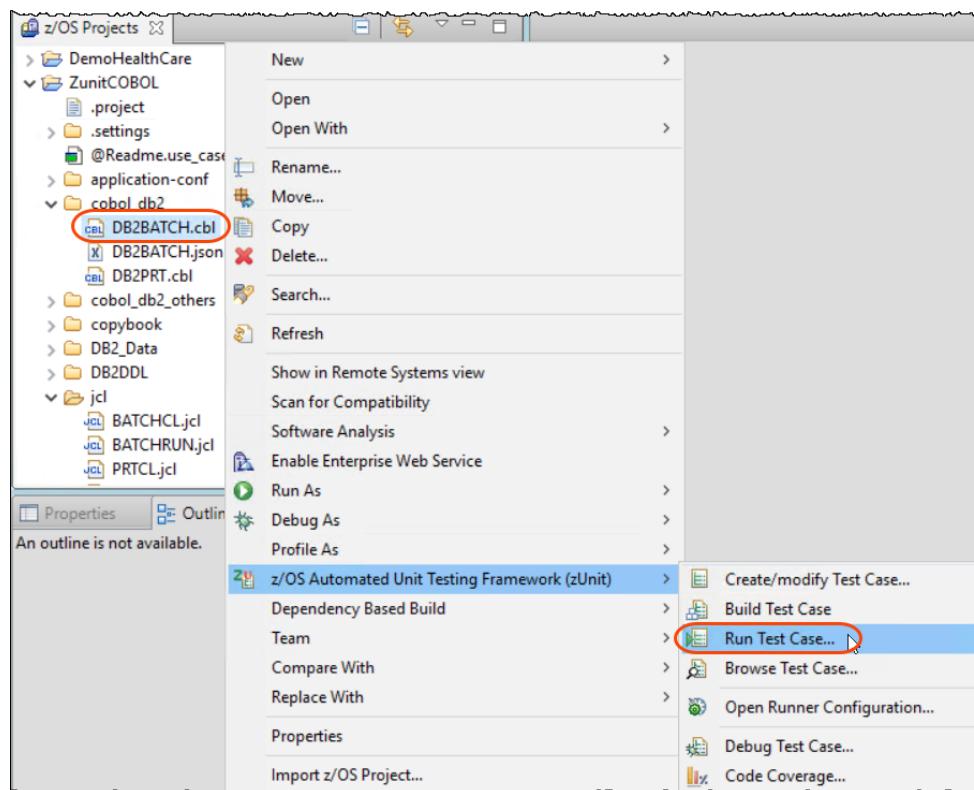
What have you done so far?

i You generated a test case load module (**TDB2BATHC**). This load module when executed can verify the correct input and output values handled by the program being verified. The test case can run each time that the program is modified, and the test case must pass it. You will see that Notice that even though the tested program accessed DB2, this data is stubbed, and the test case will run in Batch via JCL without need to have DB2 active.

3.3 Running the test case,

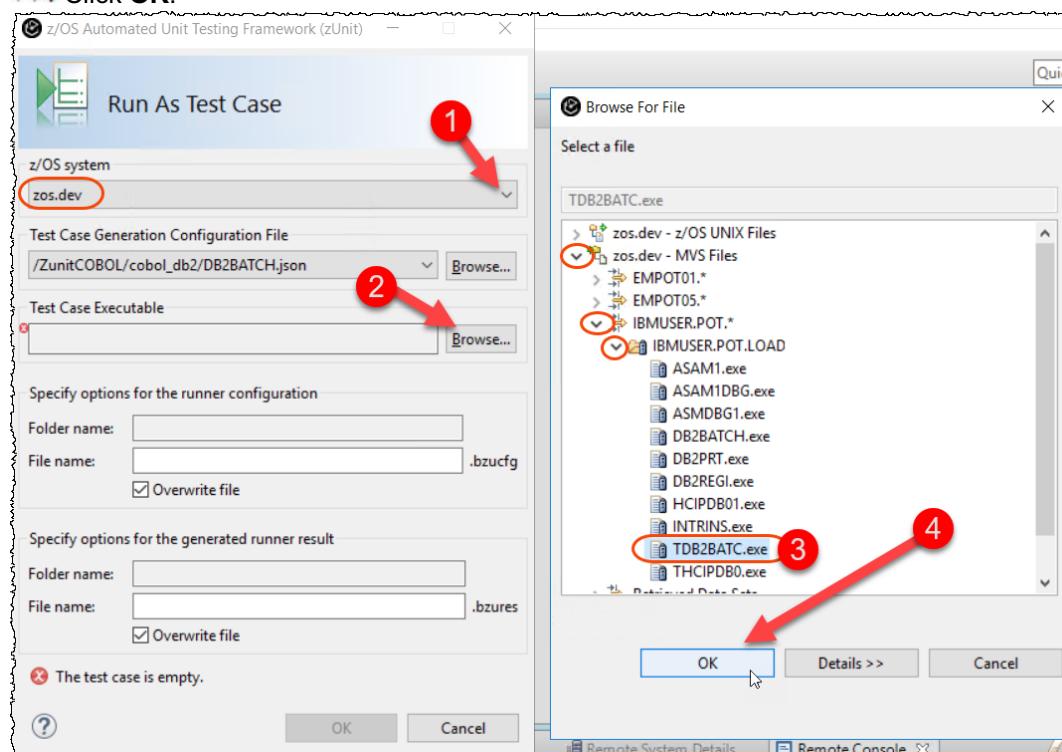
Once you have the test cases load module created you can run the test case against the *DB2BATCH* COBOL program. Since you did not make changes to the program the return code must be zero and all tests should pass.

3.3.1 ► Using z/OS Projects view, right click on **DB2BATCH.cbl** and select **z/OS Automated Unit Testing Framework (zUnit)**->**Run Test Case...**

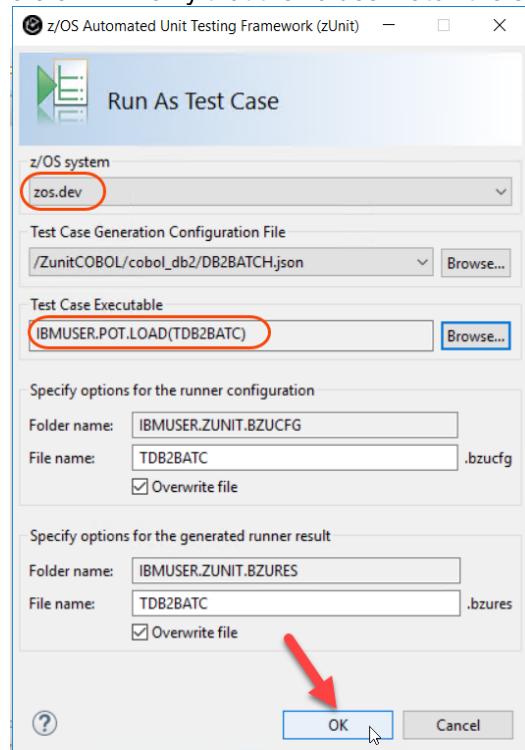


3.3.2 ► Select **zos.dev** for z/OS system,
Use the second **Browse** button to select **IBMUSER.POT.LOAD (TDB2BATC)**
under **MVS Files** and **IBMUSER.POT.***.

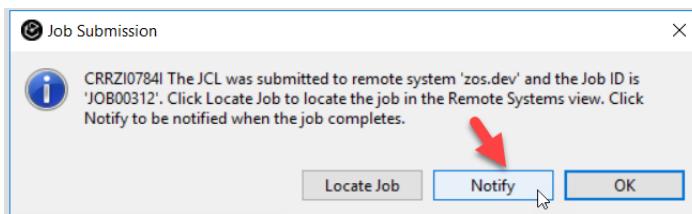
► Click **OK**.



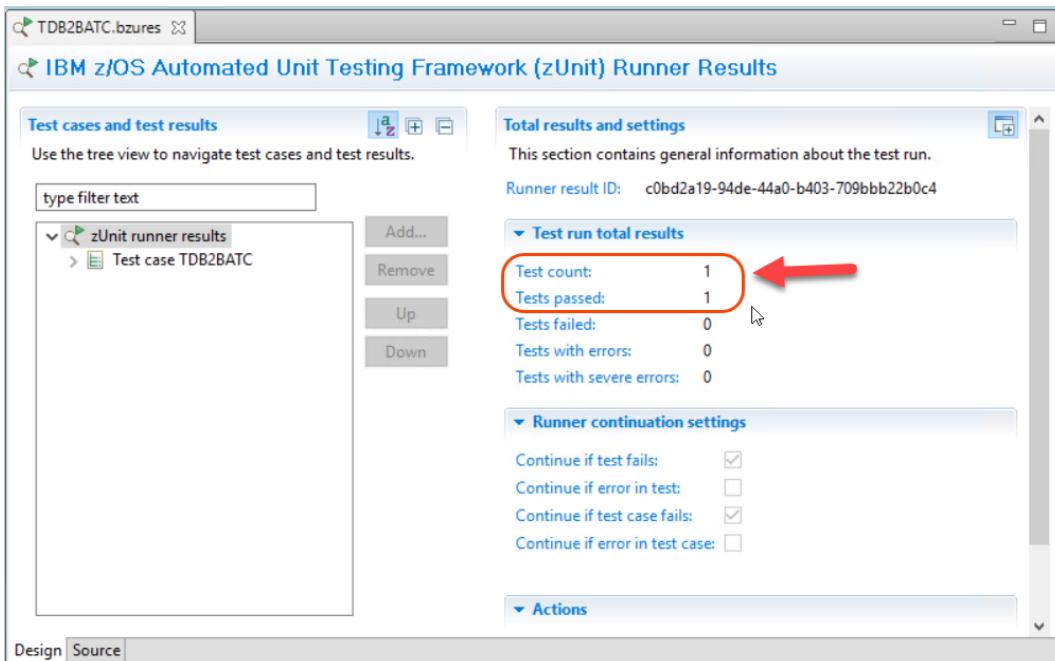
3.3.3 ► Verify that the values match the screen below and click **OK** to generate and submit a JCL.



3.3.4 ► A Job Submission dialog opens, click **Notify** to be notified of job completion.



3.3.5 Once the unit test run has completed, the results screen is displayed showing test cases ran and passed.



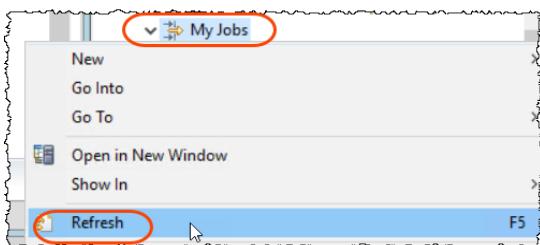
You have now created a test case with data imported from a recorded run and have been successful in testing a COBOL/DB2 batch program without the need of the DB2 environment.

3.3.6 ► Use **Ctrl + Shift + F4** to close all opened editors.

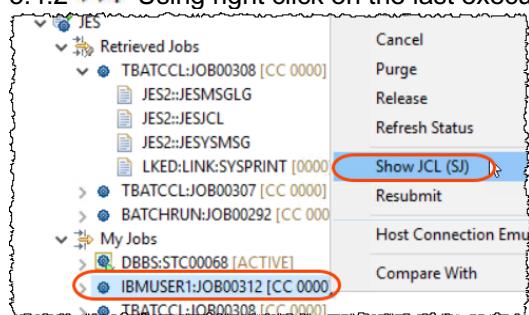
3.4 Verify the JCL submitted that runs the test case

To see the JCL submitted on the previous execution

3.4.1 ► Using **Remote Systems** view on right, under **JES** right click on **My Jobs**, and select **Refresh**.



3.4.2 ► Using right click on the last executed and select Show JCL (SJ)



3.4.3 ► The job submitted will be displayed. You could save it in a PDS member and use it when want to run the test cases for this program.

As you see there is no DB2 environment in that batch execution.



Section 4. Modify the COBOL/DB2 program (introduce a bug) and rerun the unit test

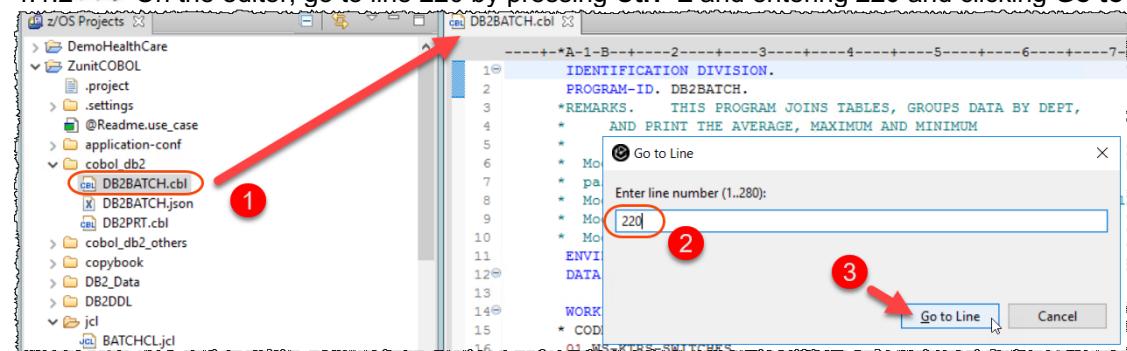
Using IDz you will modify the program and introduce a bug. Then you will rerun the unit test created in the previous section.

4.1 Modifying the program and introduce a bug

4.1.1 ► On IDz, close all active windows by pressing **Ctrl+Shift+F4**

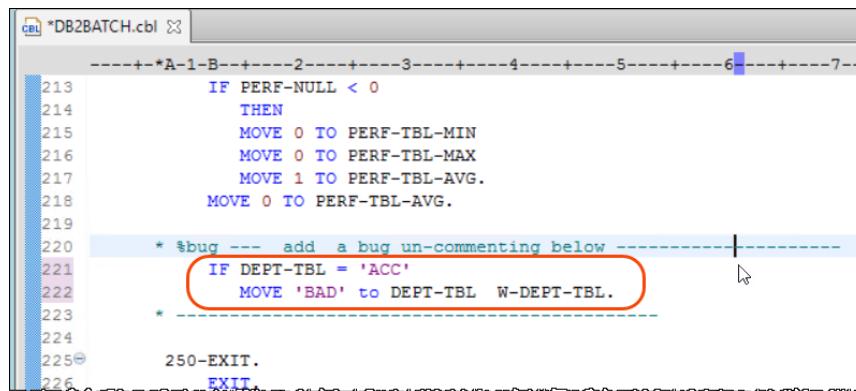
► Open **DB2BATCH.cbl** under **cobol_db2** by double clicking on it in the **z/OS Projects** view.

4.1.2 ► On the editor, go to line 220 by pressing **Ctrl+L** and entering **220** and clicking **Go to Line**.



4.1.3 ► Change the lines 221 and 222 removing the * from the statement that moves “BAD”,
 Tip -> Could use **Source > Toggle Comment**

► Press **Ctrl+S** to save the changes. and **Ctrl + Shift + F4** to close all editors.



```

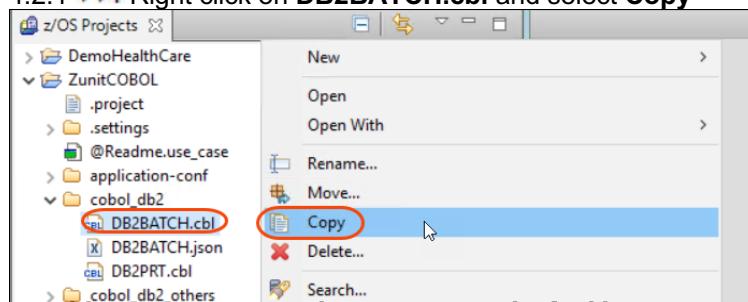
*DB2BATCH.cbl
-----+-----+-----+-----+-----+-----+-----+
213      IF PERF-NUL < 0
214      THEN
215      MOVE 0 TO PERF-TBL-MIN
216      MOVE 0 TO PERF-TBL-MAX
217      MOVE 1 TO PERF-TBL-AVG.
218      MOVE 0 TO PERF-TBL-AVG.
219
220      * %bug --- add a bug un-commenting below
221      IF DEPT-TBL = 'ACC'
222          MOVE 'BAD' to DEPT-TBL W-DEPT-TBL.
223      *
224
225      250-EXIT.
226      EXIT.

```

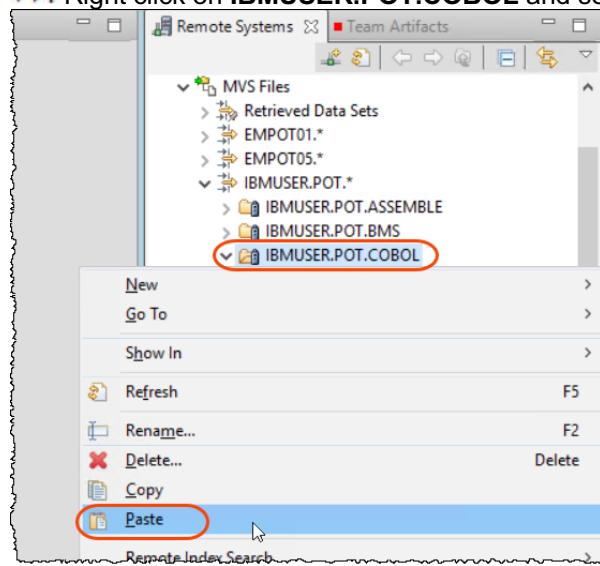
4.2 Re-compile and link the changed program

You could use the IBM DBB (part of IDz) to do the compile and link as we already mentioned before, but on this exercise we decided to do it using the old way, via JCL. Notice that the change was made at the local IDz project and the code must be moved to z/OS to compile it there. We had done similar task when we compiled/link the generated test case.

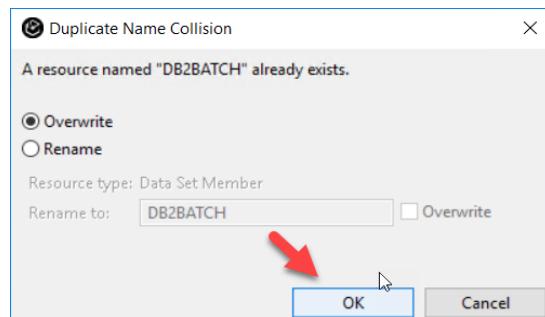
4.2.1 ► Right click on **DB2BATCH.cbl** and select **Copy**



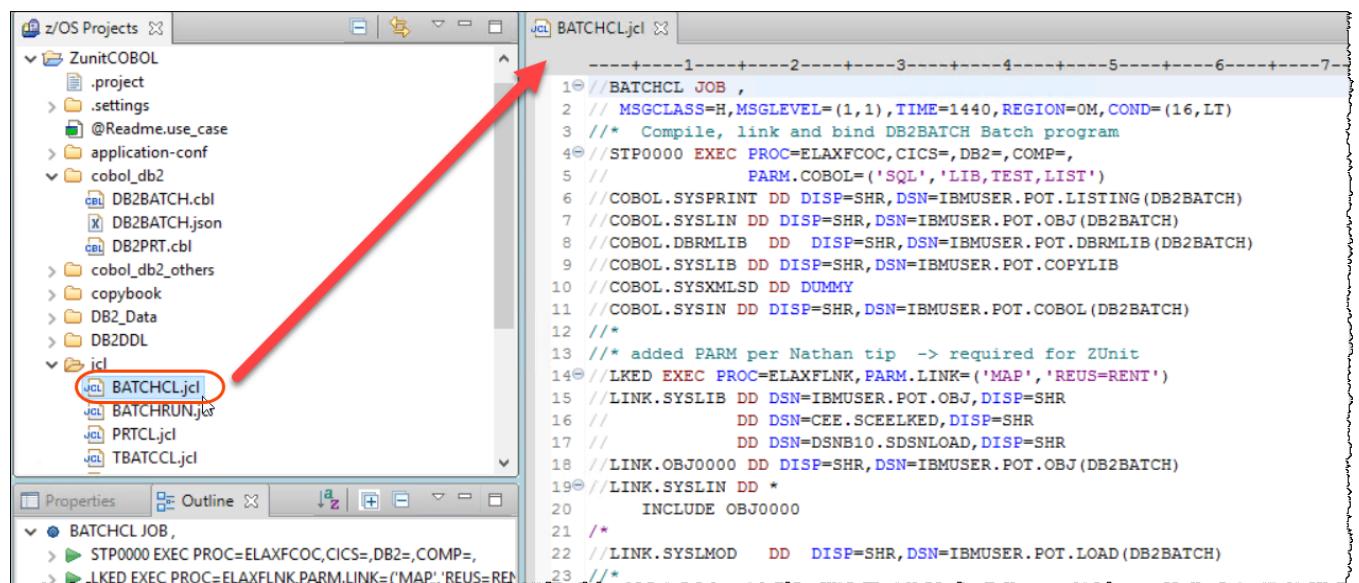
- 4.2.2 ► Using the Remote systems (on right),
► Right click on **IBMUSER.POT.COBOL** and select **Paste**



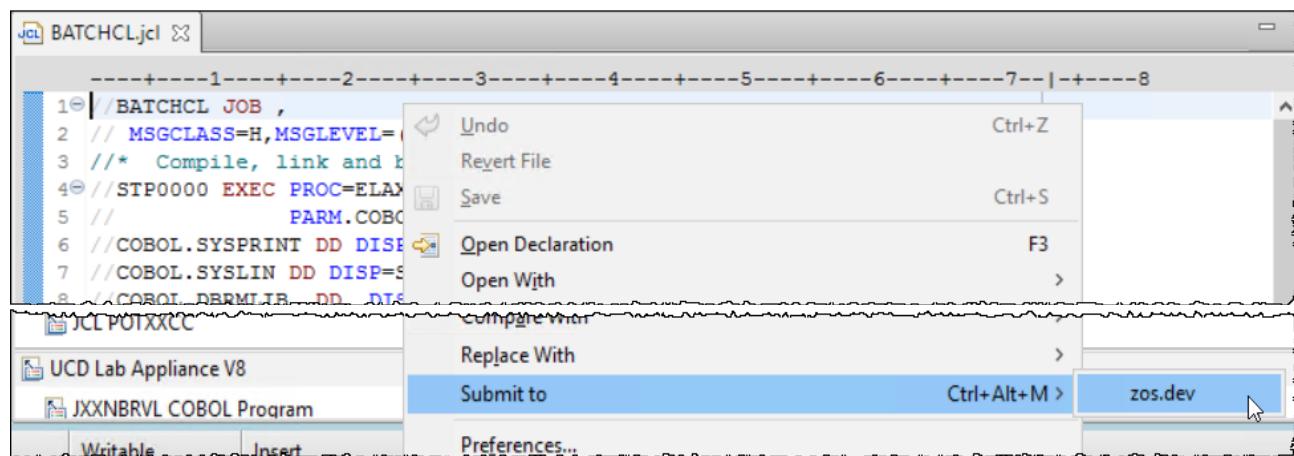
4.2.3 ► Select **OK** to overwrite this member since it already existed on z/OS



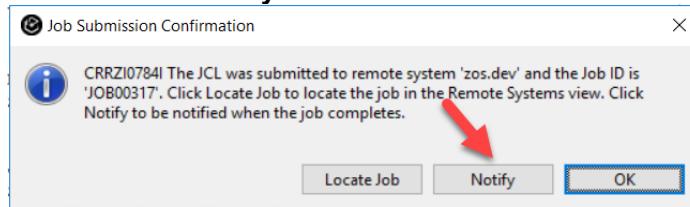
4.2.4 ► Under z/OS Projects expand **ZunitCOBOL/jcl**, and double click on **BATCHCL.jcl**. This JCL will compile, link and bind the program being tested.



4.2.5 ► Right click on the JCL being edited and select **Submit to > zos.dev** for execution on z/OS

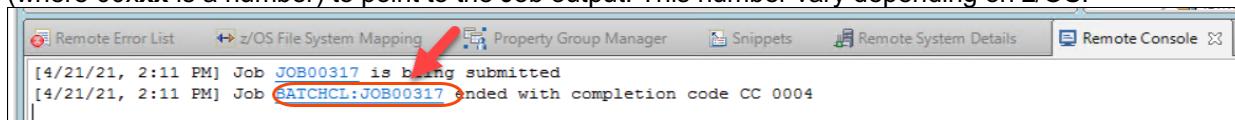


4.2.6 ► Click **Notify** to be notified when the execution is complete.



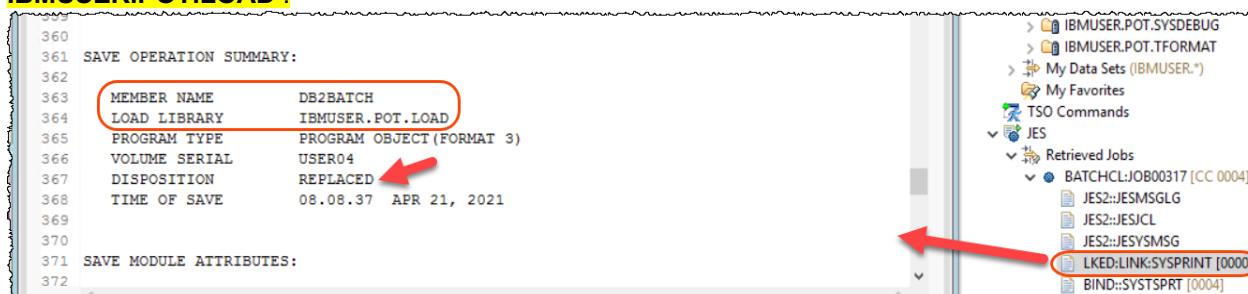
4.2.7 Under **Remote Console**, you will be notified when execution is completed. The completion code must be 4.

► Once the execution ends, click on the link **BATCHJCL:JOB00xxx** (where 00xxx is a number) to point to the Job output. This number vary depending on z/OS.



4.2.8 ► Under **Remote Systems view** expand **BATCHJCL:JOB00xxx** and double click **LKED:LINK:SYSPRINT** step.

► Scroll down the report displayed and you will see the load module **DB2BATCH** is replaced at **IBMUSER.POT.LOAD**.

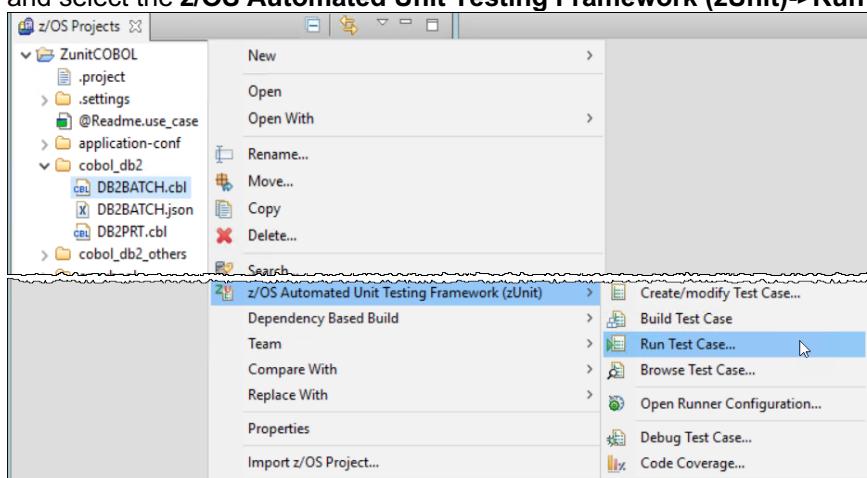


4.2.9 ► Use **Ctrl + Shift + F4** to close all opened editors.

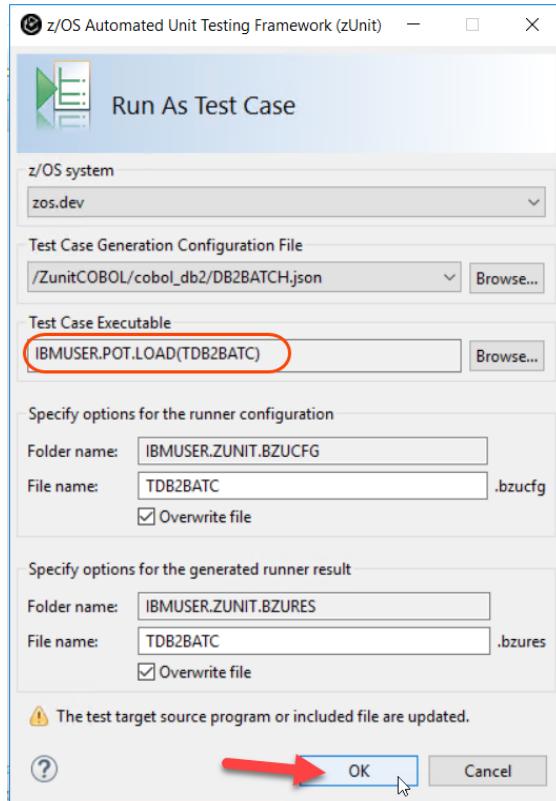
4.3 Running the test case again

Since we introduced a bug, now the test case must fail.

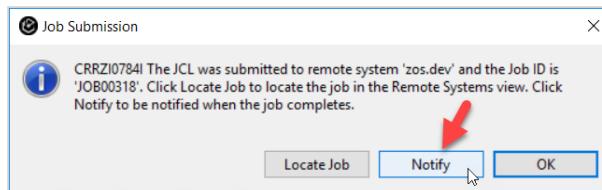
4.3.1 ► On the z/OS Projects view, select **DB2BATCH.cbl**, right mouse click, and select the **z/OS Automated Unit Testing Framework (zUnit)**->**Run Test Case..**



4.3.2 ► Verify that the test case load module **IBMUSER.POT.LOAD(TDB2BATC)** is already selected and click **OK**

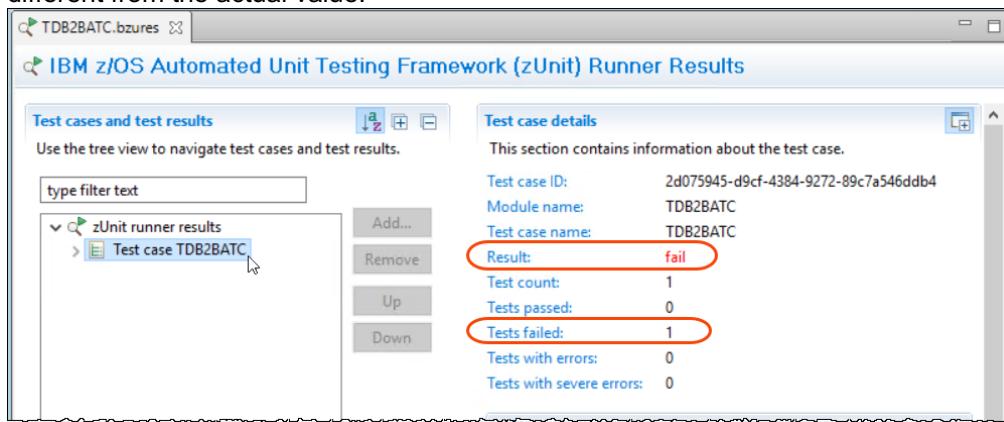


4.3.3 ► Click **Notify** on the Job Submission Confirmation dialog.



4.3.4 When the test run completes, the test results will be displayed, and it showed that one test ran and it failed. **Also notice that the completion code for this JCL execution is 0004 instead of 0000**

► Click on **Test case TDB2BATC**. The failure is expected because the expected value of the DEPT is different from the actual value.



- 4.3.5 ► Expand **Test case TDB2BATIC, Test TEST2 (fail)** and click **Exception message number** to verify the value received versus the expected value and verify the failure

The screenshot shows the 'zUnit Runner Results' interface. On the left, under 'Test cases and test results', 'Test TEST2 (fail)' is expanded, and 'Exception message number' is selected. On the right, the 'Exception details' pane displays a detailed log of exit points and their parameters. A red box highlights the beginning of the log, and a red arrow points from the 'Exception message number' selection to this highlighted area.

```

BZUP232W EXIT POINT INVOKED, REQUEST=D
BZUP220I TEST RUN TEST2 REGISTERED FOR SI
BZUP200I STARTING PROGRAM=TEST2
BZUP225I STUB CALL OPTION PREVENTED LIN
BZUP232W EXIT POINT INVOKED, REQUEST=D
BZUP225I STUB CALL OPTION PREVENTED LIN
EXIT POINT CHANGE FOR=TEST2 MODE=
EXIT PNT COLS=0-----1-----2---
EXIT PNT DATA= BAD .00 8.00
EXIT PNT DATA=44CCC4444444444FFz
EXIT PNT DATA=0021400000000000B000I
RECORDED DATA= ACC .00 8.0
RECORDED DATA=44CCC44444444444F
RECORDED DATA=0013300000000000B0C
BZUP225I STUB CALL OPTION PREVENTED LIN

```

As you see the Developer introduced a BUG and the test using ZUnit fails.

PART #2 – Fix program DB2BATCH and re-run the Unit test

Another developer will see the bug and fix it.

On previous steps you saw on the field *DEPT* the value **BAD** instead of **ACC**.

If you submit the JCL and run this program you will see that bug in the reported printed.

Section 5. Run the batch program and verify the bug .

You will run the Batch JCL and observe the bug on the printed report

5.1 Verify the BUG on the printed report

- 5.1.1 ► Under **ZunitCOBOL** expand **jcl** and double click on **BATCHRUN.jcl** to edit the JCL that will be submitted for execution.

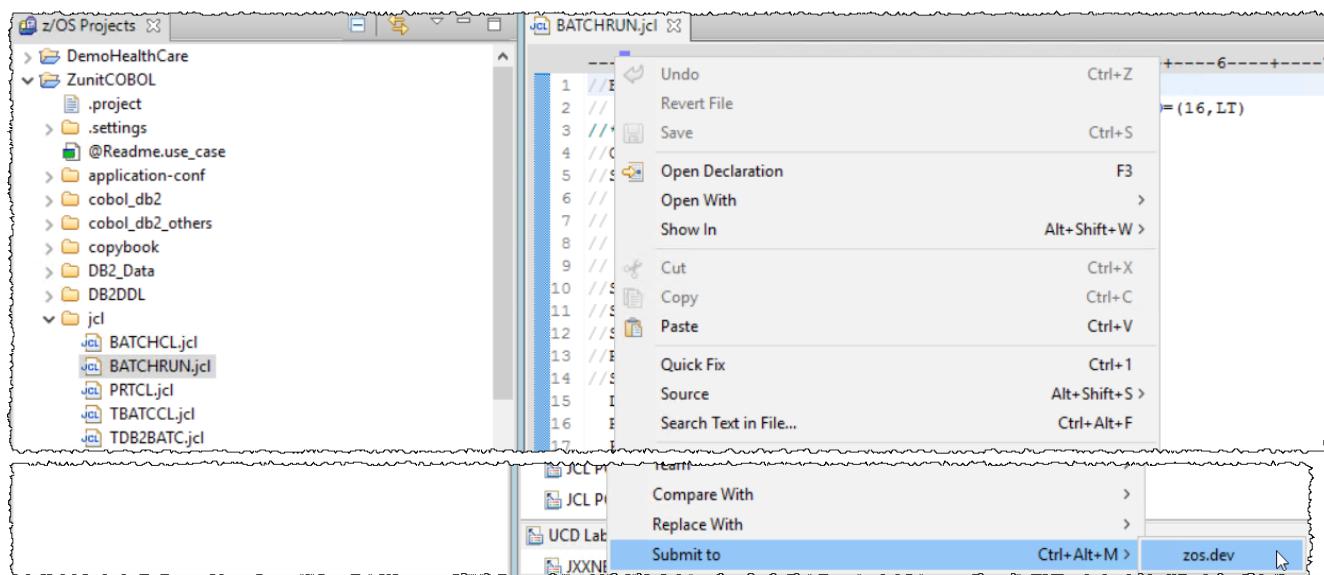
The screenshot shows the 'z/OS Projects' interface. The left pane displays a file tree with 'ZunitCOBOL' expanded, showing subfolders like '.project', '.settings', and 'application-conf'. Within 'ZunitCOBOL', there is a 'jcl' folder containing several JCL files: 'BATCHCL.jcl', 'BATCHRUN.jcl', 'PRTCL.jcl', 'TBATCL.jcl', and 'TDB2BATIC.jcl'. A red arrow points from the 'jcl' folder to the 'BATCHRUN.jcl' file, which is open in the right pane for editing. The right pane shows the JCL code for the BATCHRUN job, which includes various DD statements and EXEC PGM= commands.

```

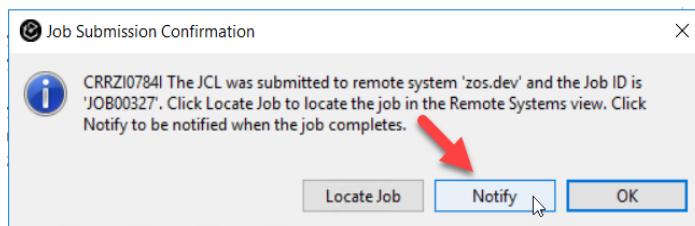
//BATCHRUN JOB ,
// MSGCLASS=H,MSGLEVEL=(1,1),TIME=1440,REGION=70M,COND=(16,LT)
///* Execute DB2BATCH Batch program
// CURSRVA4 EXEC PGM=IKJEFT01,DYNAMNBR=20
// STEPLIB DD DSN=IBMUSER.POT.LOAD,DISP=SHR
// DD DSN=DSNB10.SDSNLOAD,DISP=SHR
// DD DSN=DSNB10.DBDBG.RUNLIB.LOAD,DISP=SHR
// DD DISP=SHR,DSN=BZU100.SBZULOAD
// DD DISP=SHR,DSN=EQAE10.SEQAMOD
// SYSPRINT DD SYSOUT=*
// SYSOUT DD SYSOUT=*
// SYSPRINT DD SYSOUT=**
// RPTPRINT DD SYSOUT=*,DCB=(RECFM=FBA,BLKSIZE=1330)
// SYSSIN DD *
// DSN SYSTEM(DBDBG)
// RUN PROGRAM(DB2BATCH) -
// PLAN(DB2BATCH)
// END
//*/

```

5.1.2 ► Right click on the JCL edited and select **Submit to > zos.dev**

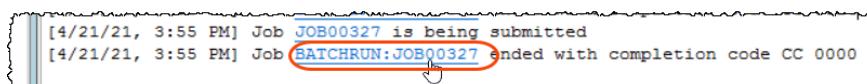


5.1.3 ► Click **Notify** to be notified when the execution is complete.



5.1.4 Under *Remote Console*, you will be notified when execution is completed.

► Once the execution ends, click on the link **BATCHRUN:JOB00xxx**
(where 00xxx is a number) to point to the Job output. This number vary depending on z/OS.



5.1.5 ► Under *Remote Systems* view scroll down, expand **BATCHRUN:JOB00xxx** and double click **CURSRAV4::IRPTPRINT** step
and you will see the report produced by the **DB2PRT** called COBOL subprogram.



5.1.6 ► Close all the opened editors using **Ctrl + Shift + F4**,

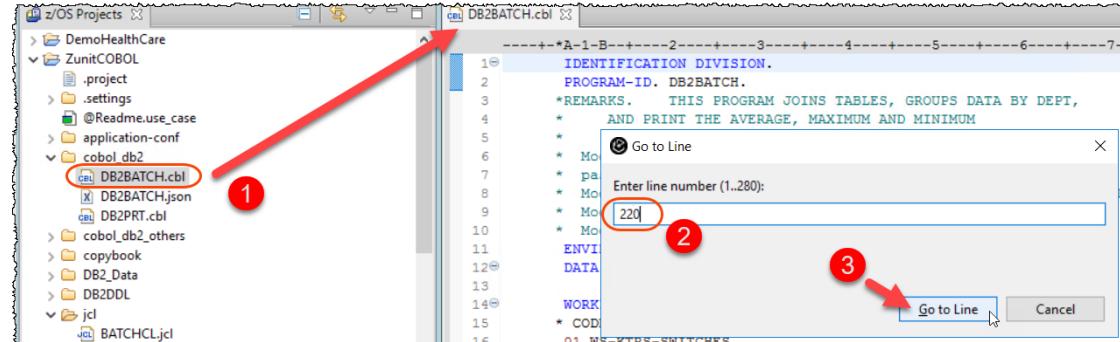
Section 6. Use IDz to fix the bug, recompile the COBOL/DB2 program.

You will fix the using IDz, and rebuilt the executable

6.1 Modifying the program to eliminate the bug

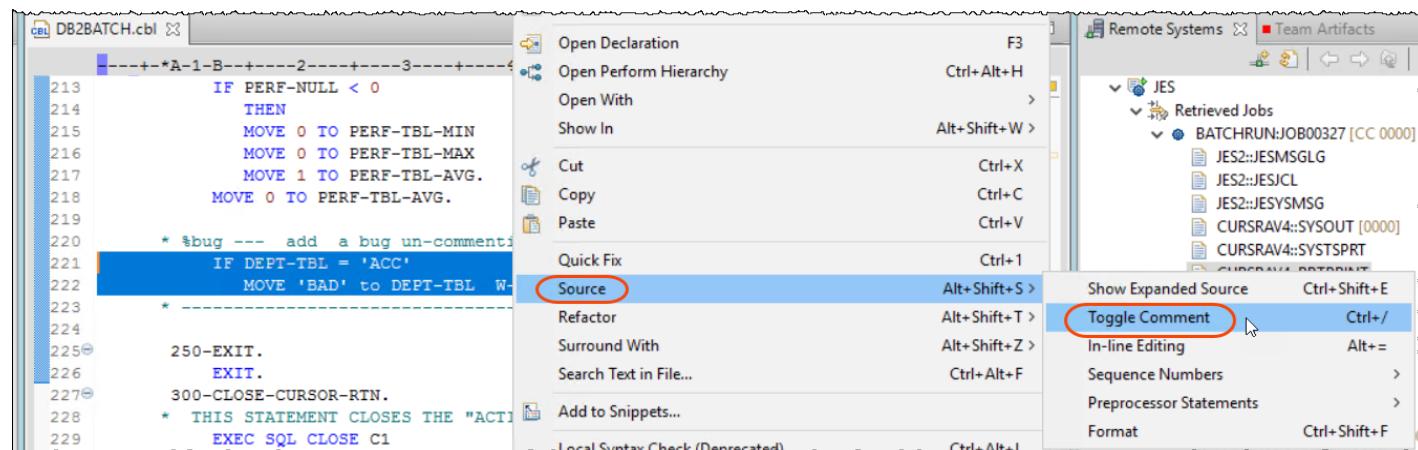
6.1.1 ► Using z/OS Projects view open DB2BATCH.cbl under **cobol_db2** by double clicking on it

6.1.2 ► On the editor, go to line 220 by pressing **Ctrl+L** and entering **220** and clicking **Go to Line**.

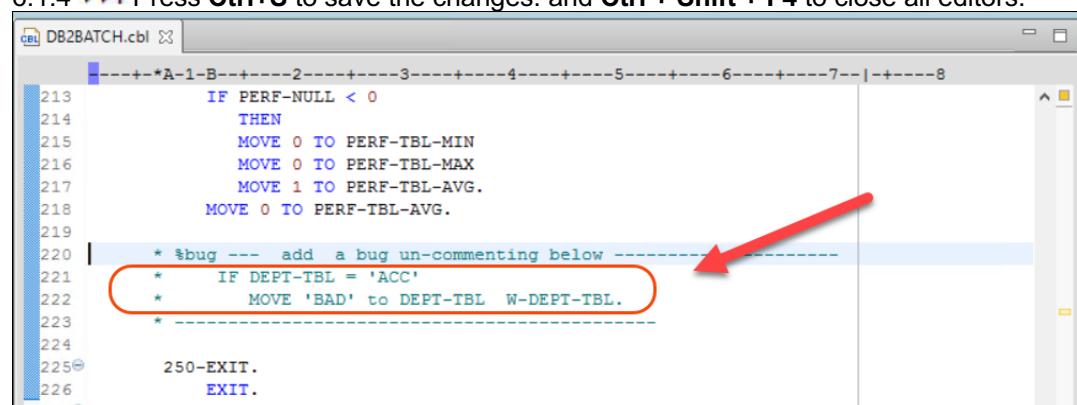


6.1.3 ► Change the lines 221 and 222 adding an * on the column 7.

The easiest way to do that is **selecting those 2 lines**, right click and select **Source > Toggle Comment**



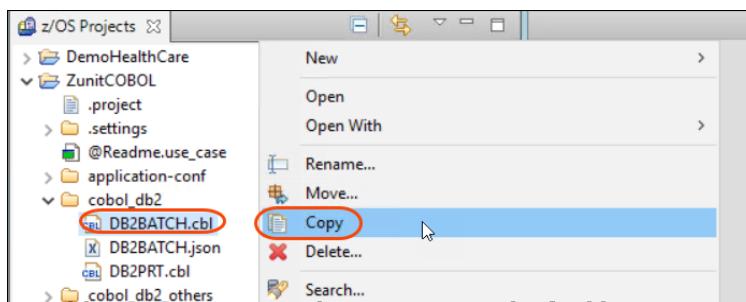
6.1.4 ► Press **Ctrl+S** to save the changes. and **Ctrl + Shift + F4** to close all editors.



6.2 Re-compile and link the changed program

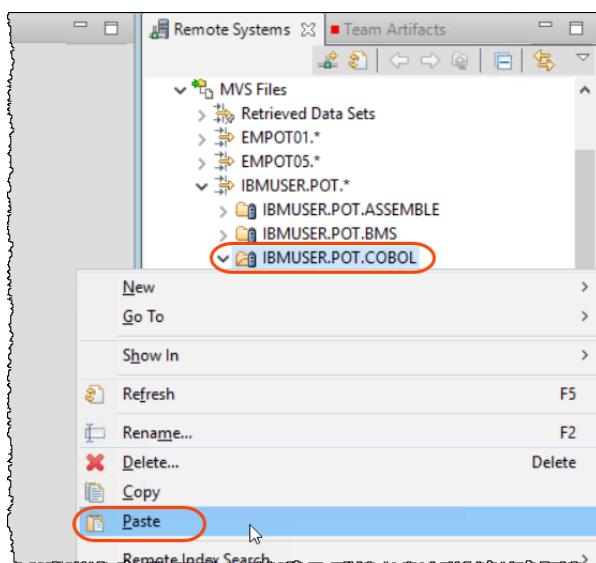
You could use the IBM DBB (part of IDz) to do the compile and link as we already mentioned before, but on this exercise we decided to do it using the “old way, via JCL. Notice that the change was made at the local IDz project and the code must be moved to z/OS to compile it there. We had done similar task when we compiled/link the generated test case.

6.2.1 ► Right click on **DB2BATCH.cbl** and select **Copy**

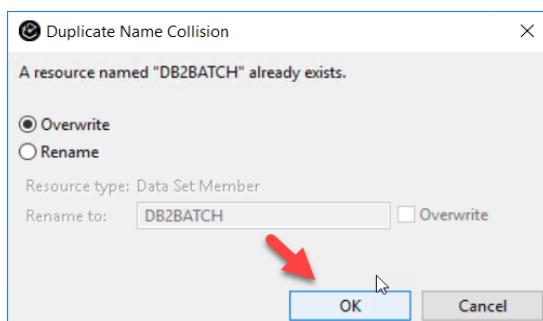


6.2.2 ► Using the Remote systems (on right),

► Right click on **IBMUSER.POT.COBOL** and select **Paste**



6.2.3 ► Select **OK** to overwrite this member since it already existed on z/OS



6.2.4 ► Under z/OS Projects expand **ZunitCOBOL/jcl**, and double click on **BATCHCL.jcl**. This JCL will compile, link and bind the program being tested.

Notice that the DB2 bind would not be necessary unless you will really execute it.

Using zUnit you run the test case using DB2 stubs data and do not need DB2 neither binding..

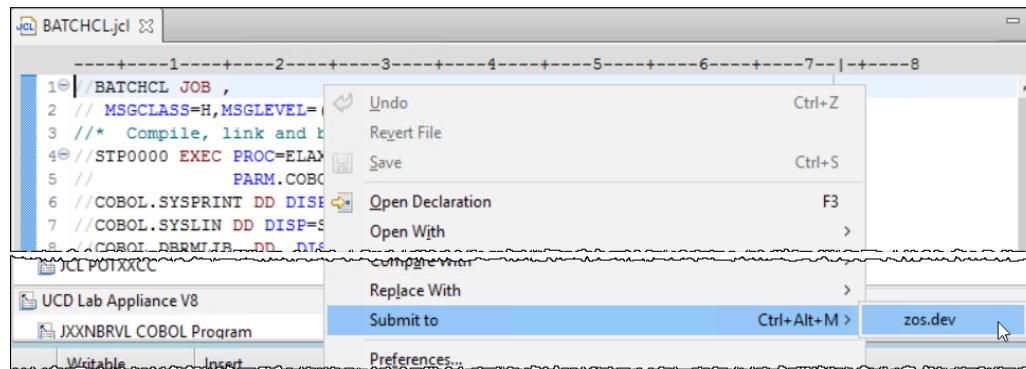
```

z/OS Projects ✎
  ZunitCOBOL
    .project
    .settings
    @Readme.use_case
    application-conf
    cobol_db2
      DB2BATCH.cbl
      DB2BATCH.json
      DB2PRT.cbl
    cobol_db2_others
    copybook
    DB2_Data
    DB2DDL
    jcl
      JCL BATCHCL.jcl (highlighted)
      JCL BATCHRUN.jcl
      JCL PRTCL.jcl
      JCL TBATCCL.jcl

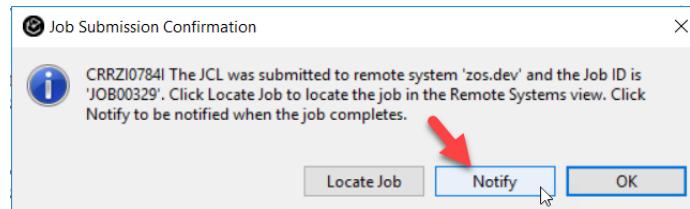
JCL BATCHCL.jcl ✎
-----+---1---+---2---+---3---+---4---+---5---+---6---+---7
1 //> BATCHCL JOB ,
2 // MSGCLASS=H,MSGLEVEL=(1,1),TIME=1440,REGION=0M,COND=(16,LT)
3 /* Compile, link and bind DB2BATCH Batch program
4 //STP0000 EXEC PROC=ELAXFCOC,CICS=DB2,COMP=
5 // PARM.COBOL='SQL','LIB,TEST,LIST'
6 //COBOL.SYSPRINT DD DISP=SHR,DSN=IBMUSER.POT.LISTING(DB2BATCH)
7 //COBOL.SYSLIN DD DISP=SHR,DSN=IBMUSER.POT.OBJ(DB2BATCH)
8 //COBOL.DBRMLIB DD DISP=SHR,DSN=IBMUSER.POT.DBRMLIB(DB2BATCH)
9 //COBOL.SYSLIB DD DISP=SHR,DSN=IBMUSER.POT.COPYLIB
10 //COBOL.SYXMLSD DD DUMMY
11 //COBOL.SYSIN DD DISP=SHR,DSN=IBMUSER.POT.COBOL(DB2BATCH)
12 /*
13 /* added PARM per Nathan tip -> required for ZUnit
14 //LKED EXEC PROC=ELAXFLNK,PARMLINK='MAP','REUS=RENT'
15 //LINK.SYSLIB DD DSN=IBMUSER.POT.OBJ,DISP=SHR
16 // DD DSN=CEE.SCEELKED,DISP=SHR
17 // DD DSN=DSNB10.SDSNLOAD,DISP=SHR
18 //LINK.OBJ0000 DD DISP=SHR,DSN=IBMUSER.POT.OBJ(DB2BATCH)
19 //LINK.SYSLIN DD *
20 INCLUDE OBJ0000
21 /*
22 //LINK.SYSLMOD DD DISP=SHR,DSN=IBMUSER.POT.LOAD(DB2BATCH)
23 /*

```

6.2.5 ► Right click on the JCL being edited and select **Submit to > zos.dev** for execution on z/OS

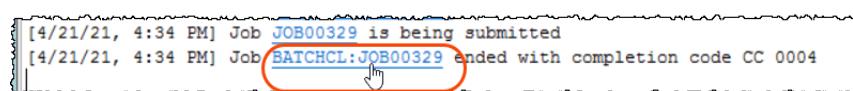


6.2.6 ► Click **Notify** to be notified when the execution is complete.



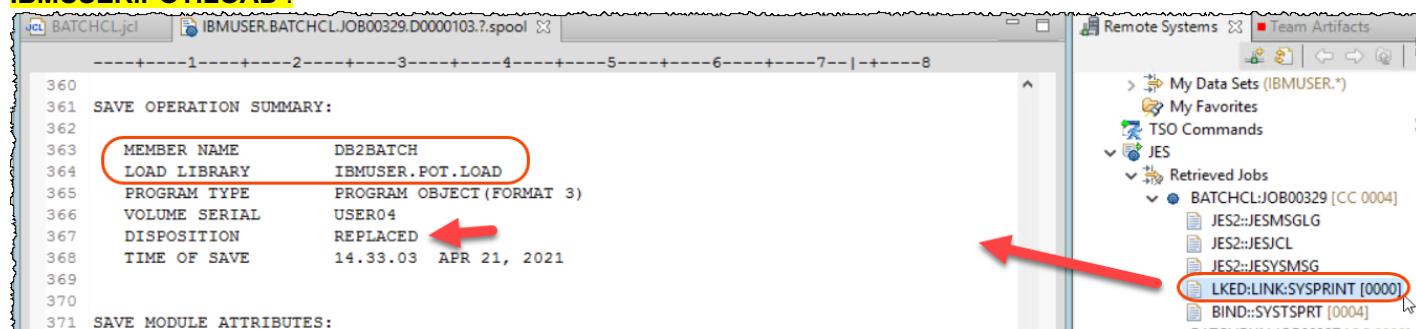
6.2.7 Under **Remote Console**, you will be notified when execution is completed. The completion code must be 4.

► Once the execution ends, click on the link **BATCHJCL:JOB00xxx** (where 00xxx is a number) to point to the Job output. This number vary depending on z/OS.



6.2.8 ► Under **Remote Systems** view expand **BATCHJCL:JOB00xxx** and double click **LKED:LINK:SYSPRINT** step.

► Scroll down the report displayed and you will see the load module **DB2BATCH** is replaced at **IBMUSER.POT.LOAD**.



6.2.9 ► Use **Ctrl + Shift + F4** to close all opened editors.

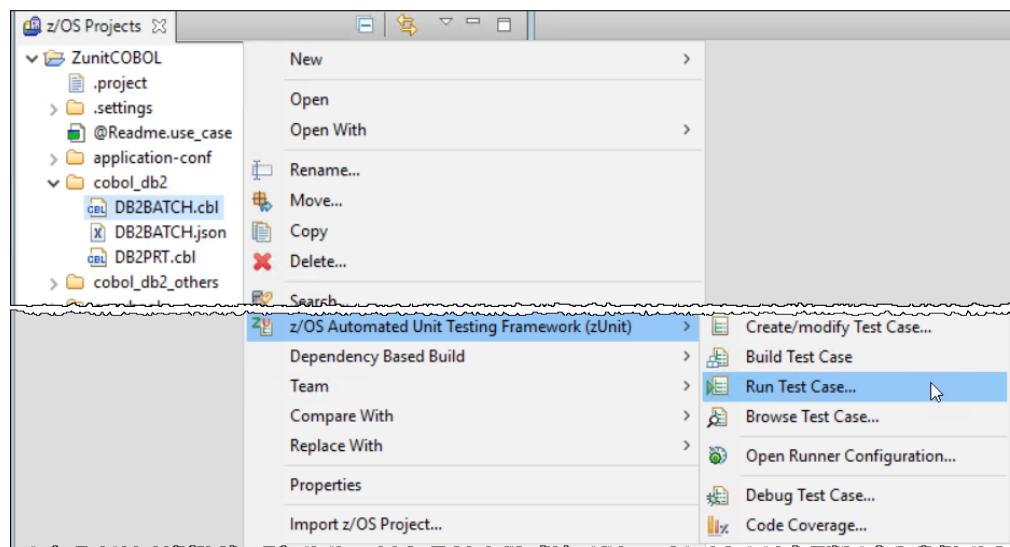
Section 7. Rerun the zUnit and verify that the bug is eliminated.

You will run the zUnit test case and verify that the program is fixed.

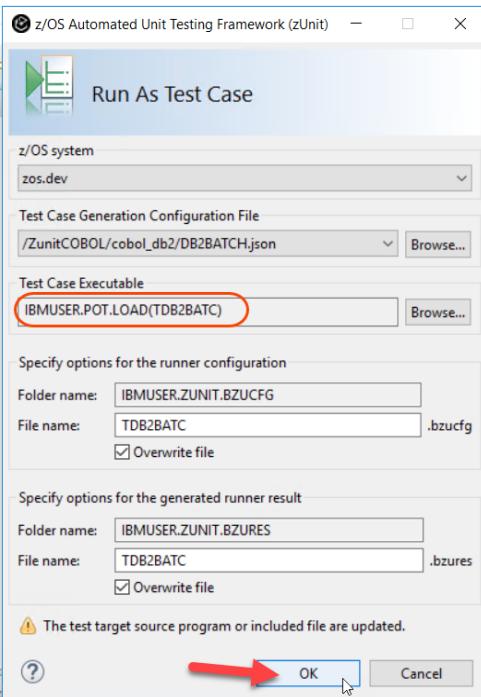
7.1 Running the test case again

Since you fixed the bug the test case now should pass it.

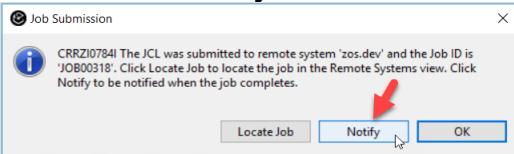
7.1.1 ► On the **z/OS Projects** view, select **DB2BATCH.cbl**, right mouse click, and select the **z/OS Automated Unit Testing Framework (zUnit)**->**Run Test Case..**



7.1.2 ► Verify that the test case load module **IBMUSER.POT.LOAD(TDB2BATC)** is selected and click **OK**

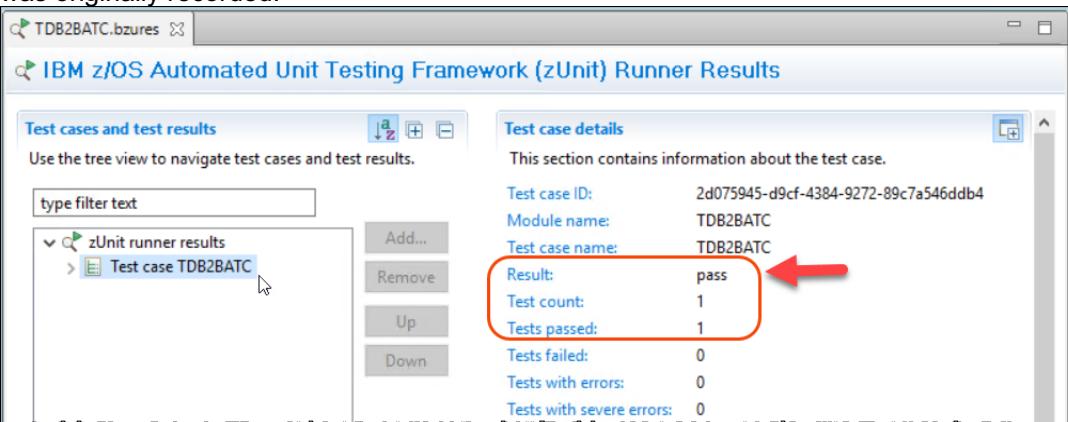


7.1.3 ► Click **Notify** on the Job Submission Confirmation dialog.



7.1.4 When the test run completes, the test results will be displayed, and it showed that one test ran and it failed. **Also notice that the completion code for this JCL execution is now 0000 since the test case passed**

► Click on **Test case TDB2BATC**. The test case now passed since the bug is fixed and results match what was originally recorded.



Congratulations! You have completed the Lab 2B.



© Copyright IBM Corporation 2019.

The information contained in these materials is provided for informational purposes only, and is provided AS IS without warranty of any kind, express or implied. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, these materials. Nothing contained in these materials is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software. References in these materials to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates. This information is based on current IBM product plans and strategy, which are subject to change by IBM without notice. Product release dates and/or capabilities referenced in these materials may change at any time at IBM's sole discretion based on market opportunities or other factors, and are not intended to be a commitment to future product or feature availability in any way.

IBM, the IBM logo and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at ibm.com/legal/copytrade.shtml

Other company, product and service names may be trademarks or service marks of others.



Please Recycle
