



# Exercise Guide: ABL Essentials

Course Version: 1.1  
Product Version: 12.2

# Copyright

---

Visit the following page online to see Progress Software Corporation's current Product Documentation Copyright Notice/Trademark Legend: <https://www.progress.com/legal/documentation-copyright>.

**February 2023**

**Product version:** Progress OpenEdge 12.2

**Updated:** 2023/02/21

# Contents

<b>Exercises.....</b>	<b>3</b>
Guided Exercise 1.1: Setting up your development environment.....	4
Part 1—Setting up a workspace and project in the Developer Studio.....	5
Part 2—Starting the Database Server.....	8
Part 3—Configuring the project to use the database.....	9
Part 4—Importing the example code files.....	14
Part 5—Configure the PROPATH.....	16
Try It 1.1: Using FIND, DISPLAY, and REPEAT.....	18
Try It 1.2: Using FIND, FOR EACH, and WHERE.....	19
Try It 1.3: Using frames.....	20
Try It 1.4: Using BEGINS, CONTAINS, and MATCHES.....	22
Try It 1.5: Sorting records.....	23
Try It 1.6: Displaying messages in a dialog box.....	24
Try It 2.1: Define variables.....	25
Try It 2.2: Dates and numbers.....	27
Try It 2.3: Data types, comparisons, strings, lists, and functions.....	29
Try It 2.4: Conditional logic.....	31
Try It 3.1: Write a simple query using both functions.....	32
Try It 3.2: Join tables in a query.....	33
Try It 3.3: Scroll a query results list.....	34
Guided Exercise 4.1: Generate a database sequence report.....	36
Guided Exercise 4.2: Create a sequence.....	38
Guided Exercise 4.3: Locate trigger code.....	41
Try It 4.1: Create a new record.....	44
Try It 4.2: Modify a sequence.....	45
Try It 4.3: Modify a database trigger.....	46
Try It 4.4: Modify records and assign values.....	48
Try It 4.5: Delete records.....	49
Guided Exercise 5.1 : Generate a compile listing.....	50
Guided Exercise 5.2: Using the TRANSACTION function to identify scope.....	53
Try It 5.1: Identify the transaction scope.....	54
Guided Exercise 5.3 : Save data with default transaction scope.....	56
Guided Exercise 5.4: Save data with increased transaction scope.....	58
Guided Exercise 5.5: Save data with reduced transaction scope.....	60
Guided Exercise 5.6: Running a sub-procedure within a transaction.....	62
Try It 5.2: Modify the transaction scope.....	65
Try It 6.1: Identify the record scope and transaction duration.....	66
Try It 6.2: Implement optimistic locking.....	68
Try It 7.1: Overriding the default error handling.....	71

Try It 7.2: Checking for errors.....	73
Try It 7.3: Using ON-ERROR.....	76
Try It 8.1: Parameters and functions.....	78
Try It 8.2: Persistent procedures.....	85
Try It 9.1: Identify the record scope and transaction duration.....	87

## **Solutions.....91**

Try It 1.1: Using FIND, DISPLAY, and REPEAT, Solutions.....	92
Try It 1.2: Using FIND, FOR EACH, and WHERE, Solutions.....	93
Try It 1.3: Using frames, Solutions.....	94
Try It 1.4: Using BEGINS, CONTAINS, and MATCHES, Solutions.....	96
Try It 1.5: Sorting records, Solutions.....	97
Try It 1.6: Displaying messages in a dialog box, Solutions.....	97
Try It 2.1: Define variables, Solutions.....	98
Try It 2.2: Dates and numbers, Solutions.....	99
Try It 2.3: Data types, comparisons, strings, lists, and functions, Solutions.....	101
Try It 2.4: Conditional logic, Solutions.....	102
Try It 3.1: Write a simple query using both functions, Solutions.....	103
Try It 3.2: Join tables in a query, Solutions.....	104
Try It 3.3: Scroll a query results list, Solutions.....	105
Try It 4.1: Create a new record, Solutions.....	105
Try It 4.2: Modify a sequence, Solutions.....	106
Try It 4.3: Modify a database trigger, Solution.....	106
Try It 4.4: Modify records and assign values, Solutions.....	107
Try It 4.5: Delete records, Solutions.....	107
Try It 5.1: Identify the transaction scope, Solutions.....	108
Try It 5.2: Modify the transaction scope, Solutions.....	110
Try It 6.1 : Identify record scope and locking duration, Solutions.....	112
Try It 6.2: Implement optimistic locking, Solutions.....	113
Try It 7.1: Overriding the default error handling, Solutions.....	115
Try It 7.2: Checking for errors, Solutions.....	116
Try It 7.3: Using ON-ERROR, Solutions.....	117
Try It 8.1: Parameters and functions, Solutions.....	119
Try It 8.2: Persistent procedures, Solutions.....	120
Try It 9.1 : Integrate logic into an application, Solutions.....	121

# Exercises

---

This section contains the following topics:

- [Guided Exercise 1.1: Setting up your development environment](#)
- [Try It 1.1: Using FIND, DISPLAY, and REPEAT](#)
- [Try It 1.2: Using FIND, FOR EACH, and WHERE](#)
- [Try It 1.3: Using frames](#)
- [Try It 1.4: Using BEGINS, CONTAINS, and MATCHES](#)
- [Try It 1.5: Sorting records](#)
- [Try It 1.6: Displaying messages in a dialog box](#)
- [Try It 2.1: Define variables](#)
- [Try It 2.2: Dates and numbers](#)
- [Try It 2.3: Data types, comparisons, strings, lists, and functions](#)
- [Try It 2.4: Conditional logic](#)
- [Try It 3.1: Write a simple query using both functions](#)
- [Try It 3.2: Join tables in a query](#)
- [Try It 3.3: Scroll a query results list](#)
- [Guided Exercise 4.1: Generate a database sequence report](#)
- [Guided Exercise 4.2: Create a sequence](#)
- [Guided Exercise 4.3: Locate trigger code](#)
- [Try It 4.1: Create a new record](#)
- [Try It 4.2: Modify a sequence](#)
- [Try It 4.3: Modify a database trigger](#)
- [Try It 4.4: Modify records and assign values](#)
- [Try It 4.5: Delete records](#)
- [Guided Exercise 5.1 : Generate a compile listing](#)
- [Guided Exercise 5.2: Using the TRANSACTION function to identify scope](#)

- [Try It 5.1: Identify the transaction scope](#)
- [Guided Exercise 5.3 : Save data with default transaction scope](#)
- [Guided Exercise 5.4: Save data with increased transaction scope](#)
- [Guided Exercise 5.5: Save data with reduced transaction scope](#)
- [Guided Exercise 5.6: Running a sub-procedure within a transaction](#)
- [Try It 5.2: Modify the transaction scope](#)
- [Try It 6.1: Identify the record scope and transaction duration](#)
- [Try It 6.2: Implement optimistic locking](#)
- [Try It 7.1: Overriding the default error handling](#)
- [Try It 7.2: Checking for errors](#)
- [Try It 7.3: Using ON-ERROR](#)
- [Try It 8.1: Parameters and functions](#)
- [Try It 8.2: Persistent procedures](#)
- [Try It 9.1: Identify the record scope and transaction duration](#)

## Guided Exercise 1.1: Setting up your development environment



### Overview

In this Guided Exercise, you set up your development environment for this course using Progress Developer Studio for OpenEdge (Developer Studio).

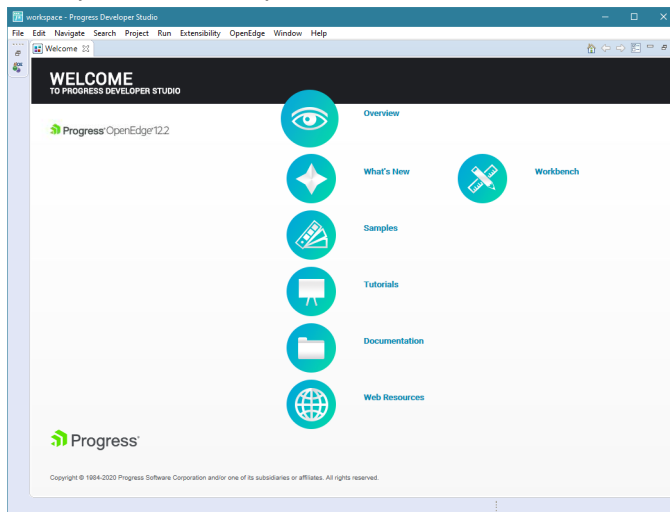
This exercise has several parts. The exercise steps take approximately 30 minutes to complete. You practice this task in your live version of Progress OpenEdge.

Before you begin this Guided Exercise, you must set up your exercise environment, if you have not done so already. See the "Set up your exercise environment" section at the beginning of the course for details.

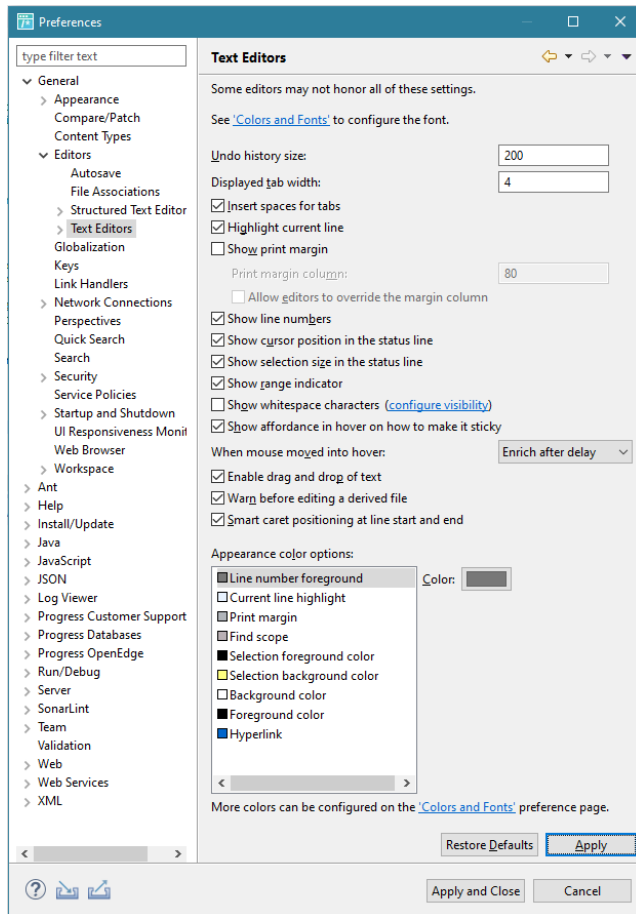
## Part 1—Setting up a workspace and project in the Developer Studio

You will create a Developer Studio workspace area and the Project space that will be used for all your ABL development for this course.

1. Start Developer Studio by selecting **Start > Progress > Developer Studio 12.2 (64 bit)**.
2. If this is your first time using Developer Studio, the first dialog is for you to specify where your workspace is located.
  - a. Specify `C:\progress_education\openedge` as the workspace location.
  - b. Click **Launch**.
  - c. You may first see the welcome screen as shown here. Click the **Workbench** icon to continue to the workspace for Developer Studio.

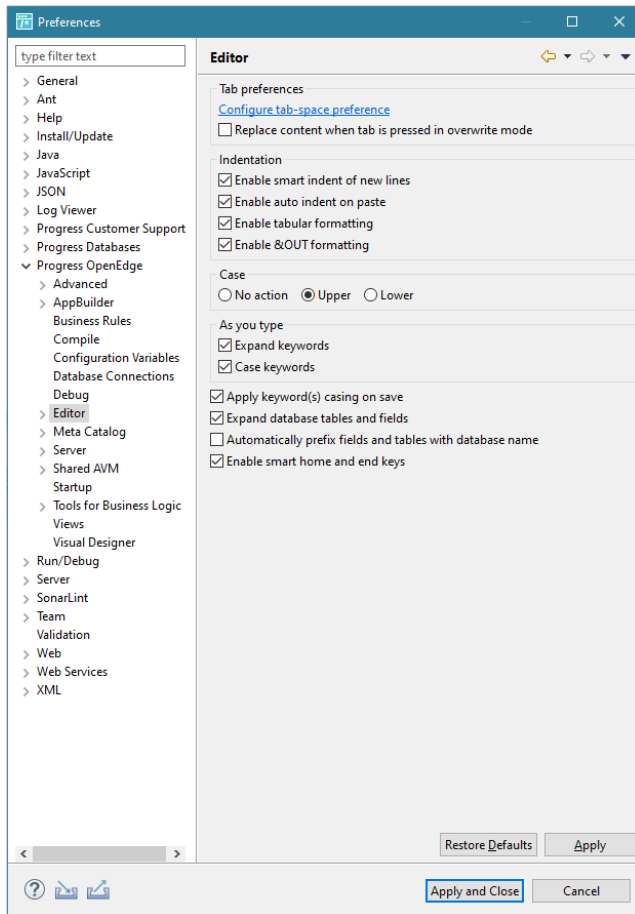


3. If you have previously used Developer Studio for OpenEdge, you must switch to a new workspace.
  - a. Select **File > Switch Workspace > Other....**
  - b. Specify `C:\progress_education\openedge` as the workspace location.
  - c. Click **Launch**. Developer Studio will restart.
4. Modify these workspace preferences as follows:
  - a. Select **Window > Preferences**.
  - b. In the Preferences window, navigate to **General > Editors > Text Editors**.
  - c. Select **Show line numbers**.

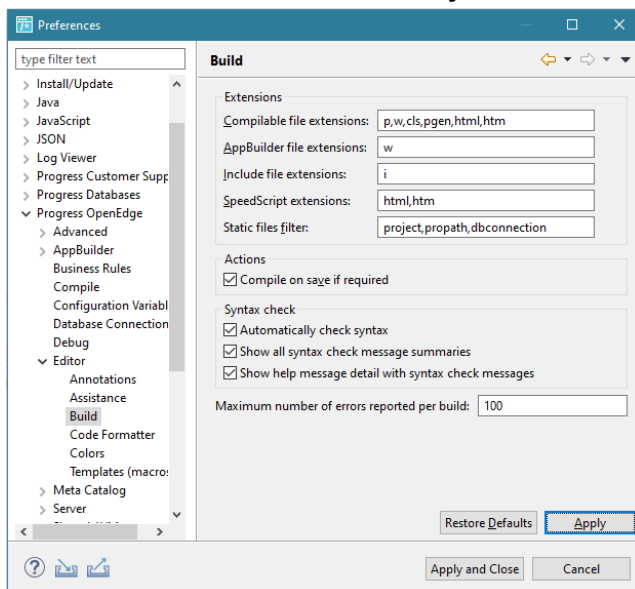


- d. Click **Apply**.
- e. Navigate to **Progress OpenEdge > Editor**.
- f. Select **Upper** in the **Case** area if not already selected.
- g. Select **Expand keywords** and **Case keywords** in the **As you type** area.
- h. Select **Apply keyword(s) casing on save**.
- i. Select **Expand database tables and fields**.





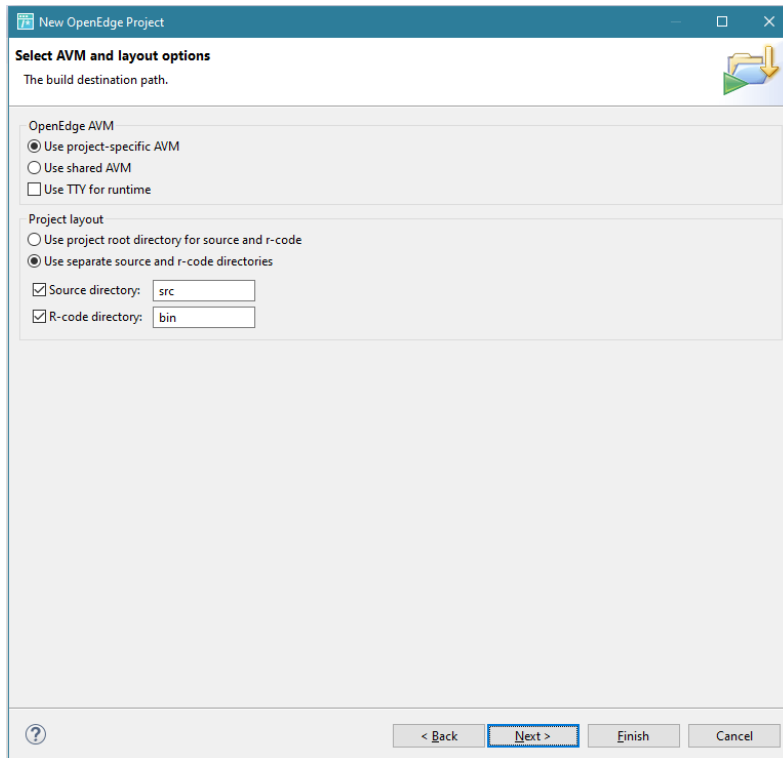
- j. Click **Apply**.
- k. Navigate to **Progress OpenEdge > Editor > Build**.
- l. Select all boxes in the **Actions** and **Syntax check** areas.



- m. Click **Apply and Close**.


Follow these steps to create your ABL Essentials project in the workspace you created in the preceding section.

1. Select **File > New > OpenEdge Project**.
2. In the **New OpenEdge Project** wizard:
  - a. Enter **ABLEssentials** for the project name.
  - b. Select the **General** icon.
  - c. Select **OpenEdge Basic** for the project type.
  - d. Click **Next**.
  - e. In the **Select AVM and layout options** window, select **Use separate source and r-code directories**.
  - f. Change the R-code directory from **rcode** to **bin**.



- g. Click **Finish**.
- h. If the **Open Associated Perspective** dialog is displayed, asking you if you want to open the project in the OpenEdge Editor perspective, click **Open Perspective**.

---

**Note:** Your project should always be open in the OpenEdge Editor perspective. The  icon should be visible on the top right corner of the developer studio. If it is not visible then select **Windows > Perspective > Open Perspective > Other > OpenEdge Editor (default)**.

---

## Part 2—Starting the Database Server

This course uses a database named `sports2020`. In this part of the Guided Exercise, you create the database files and start the database server.

1. Open a Proenv window by selecting **Start > Progress > Proenv 12.2 (64 bit)**.
2. Prepare a location for the database files:

- a. Navigate to C:\progress\_education\openedge\ABLEssentials:

```
cd C:\progress_education\openedge\ABLEssentials
```

- b. Create a directory called db for the database:

```
mkdir db
```

- c. Navigate to the db directory:

```
cd db
```

3. Create a copy of the sports2020 database in the course db directory as follows:

```
proddb sports2020 sports2020
```

You should see a message similar to the following in the terminal:

```
proenv>proddb sports2020 sports2020
Procopy session begin for admin on CON:. (451)
Database copied from C:\Progress\OpenEdge\sports2020. (1365)
Procopy session end. (334)
```

4. Start the database server for the sports2020 database as follows:

```
proserve sports2020 -H localhost -S 50000
```

You should see a message similar to the following in the terminal.

```
proenv>proserve sports2020 -H localhost -S 50000
OpenEdge Release 12.2 as of Tue Mar 17 19:02:26 EDT 2020
22:21:35 BROKER      This broker will terminate when session ends. (5405)
22:21:35 BROKER      The startup of this database requires 40Mb of shared memory.
Maximum segment size is 1024Mb.
22:21:35 BROKER      0: Multi-user session begin. (333)
22:21:35 BROKER      0: Before Image Log Initialization at block 0  offset 639. (15321)
22:21:35 BROKER      0: Login by admin on CON:. (452)
22:21:35 BROKER      0: Started for 50000 using TCP IPV4 address 127.0.0.1, pid 25536.
(5644)
```

---

**Note:** If port 50000 is unavailable on your system, use a different port number. Make a note of the port number you use, because you will need it later.

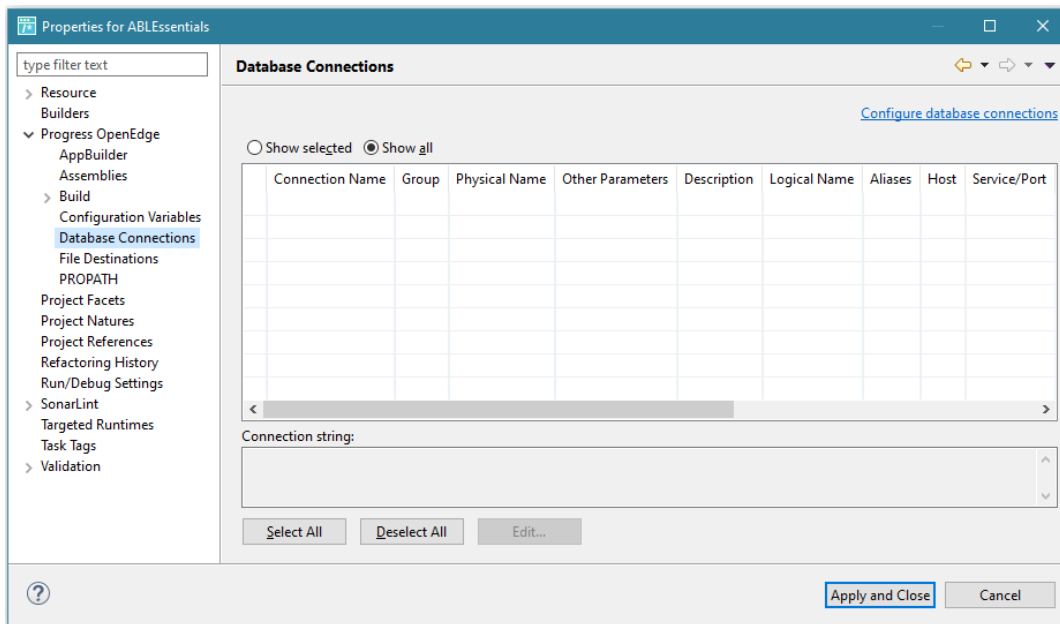
---

5. Close the Proenv window.

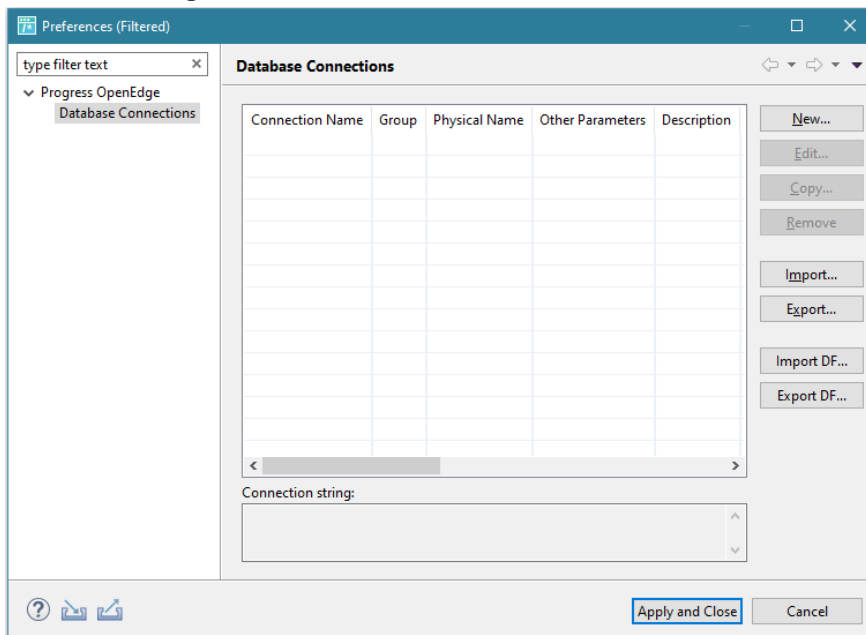
## Part 3—Configuring the project to use the database

Code from the ABLEssentials project accesses the database. Follow these steps to configure the ABLEssentials project to use the sports2020 database.

1. In this step, you will add the database connection to the workspace and associate it with the ABLEssentials project. Right-click the **ABLEssentials** project in the **Project Explorer** pane and then select **Properties**.
2. Navigate to **Progress OpenEdge > Database Connections**.

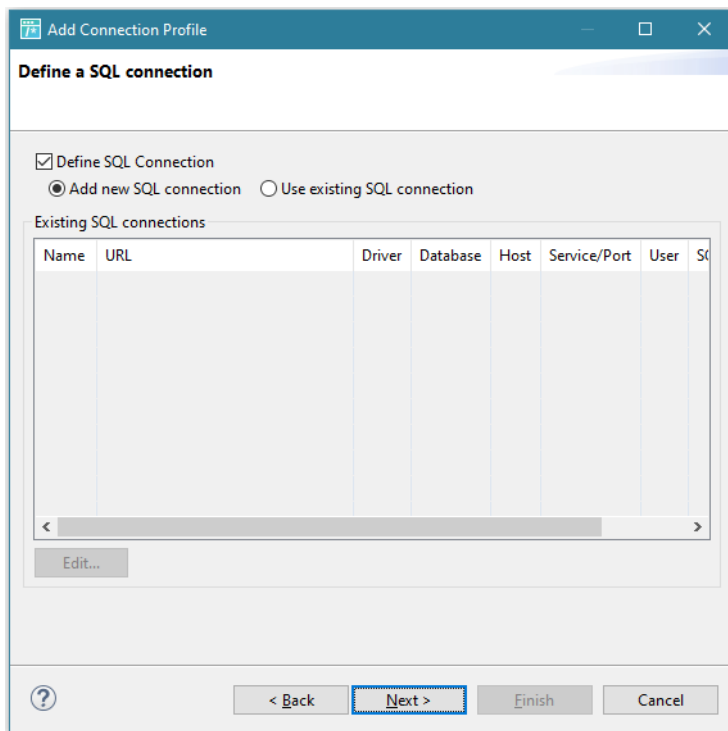


3. Click the **Configure database connections** link.

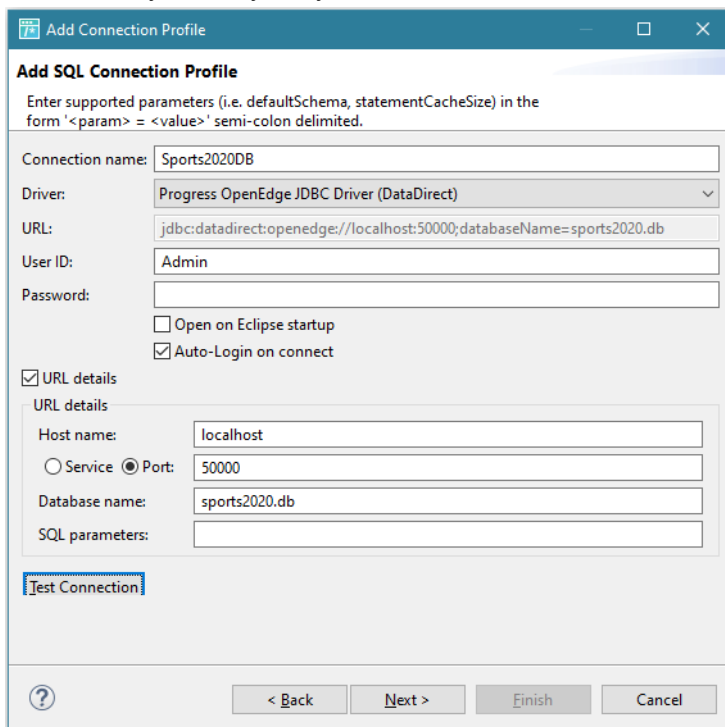


4. Click **New** to open the **Add Connection Profile** wizard.

5. Enter the connection name, **Sports2020DB**.
6. Click the **Browse** button and then navigate to and select `C:\progress_education\openedge\ABLEssentials\db\sports2020.db`. Click **Save**.
7. Specify the host name of **localhost**.
8. Specify the service/port number as **50000**.  
If port 50000 is unavailable on your system, use a different port number. (It must be the same port number that you specified when you started the Database Server.)
9. Click the **Test Connection** button.
10. After the test succeeds, click **OK**. If the connection did not succeed, you will need to investigate why the connection failed. Is the Database Server for the database running?
11. Click **Next**. The **Define a SQL connection** window opens. You will leave the defaults for this window.



12. Click **Next**. The **Add SQL Connection Profile** window opens. Here you retain the values that have been automatically set for you by the wizard.

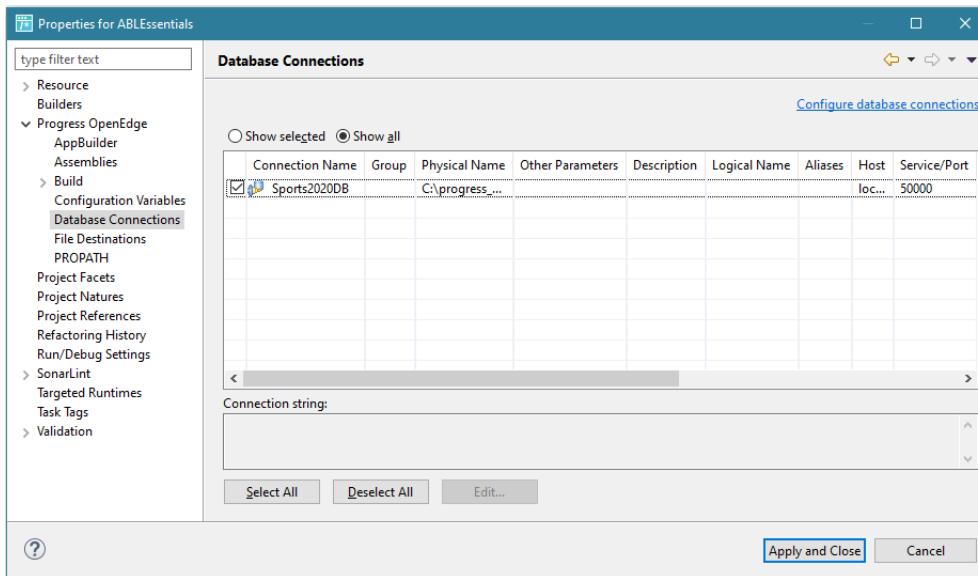


**Note:** The database name specified in the Database name field must be in all lower case. If the name has upper-case letters in it, you must modify them to be lower case.

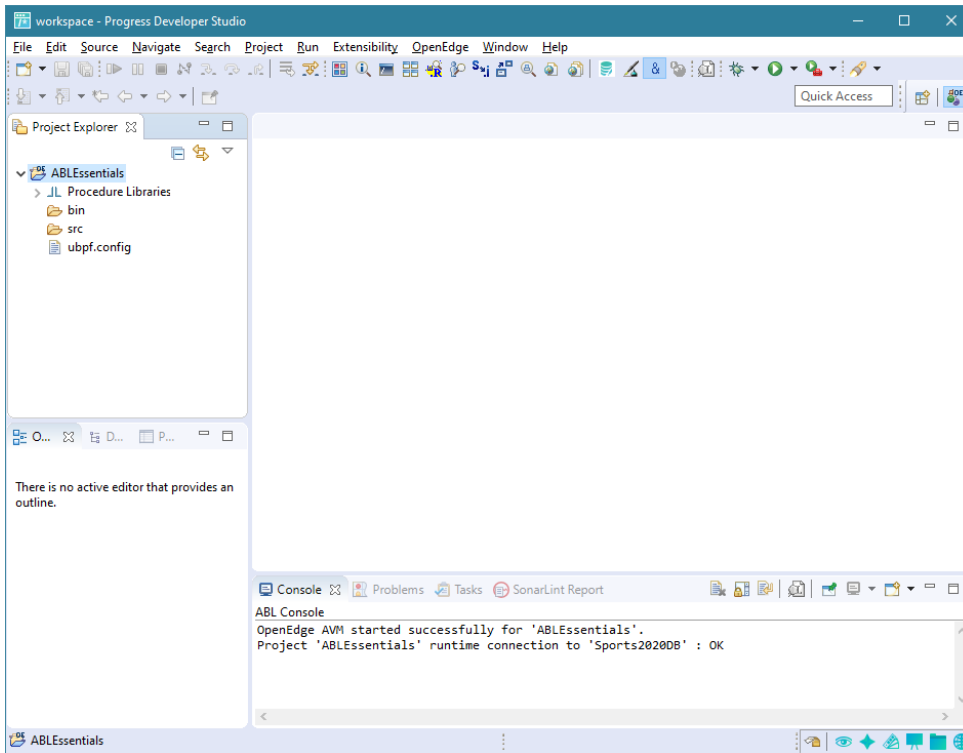
13. Click the **Test Connection** button. Then click **OK** when the test succeeds.

14. Click **Next**. The **Define Database Server Configuration** window opens. Notice that **Auto-start database server** is selected. Leave that box selected. Developer Studio will automatically start the Database Server for you. Do not select **Auto-shutdown database server**.
15. Click **Finish**.
16. After the configuration is completed, the connection string is shown as:
 

```
-db ABLEssentials\db\sports2020.db -H localhost -S 50000
```
17. Click **Apply and Close**. You are back in the **Database Connections** window.
18. Check the box next to the connection profile you just configured.



19. Click **Apply and Close** to complete the configuration. You will notice in the console that the AVM for the ABLEssentials project restarts. This is because it must start an AVM that has an active connection to the Database Server. Notice also that in the **Project Explorer** pane (on the left), the ABLEssentials project appears with the `bin` and `src` folders.

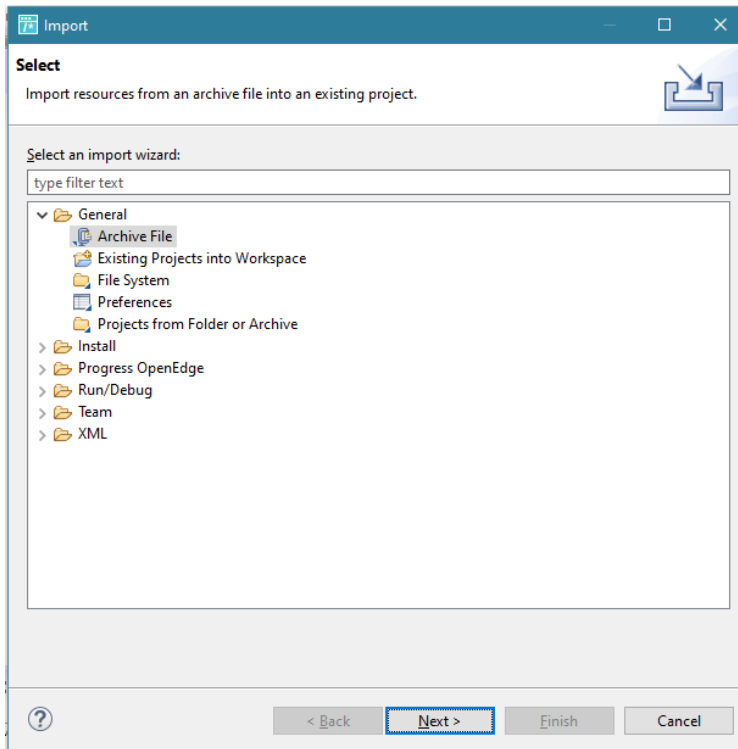


## Part 4—Importing the example code files

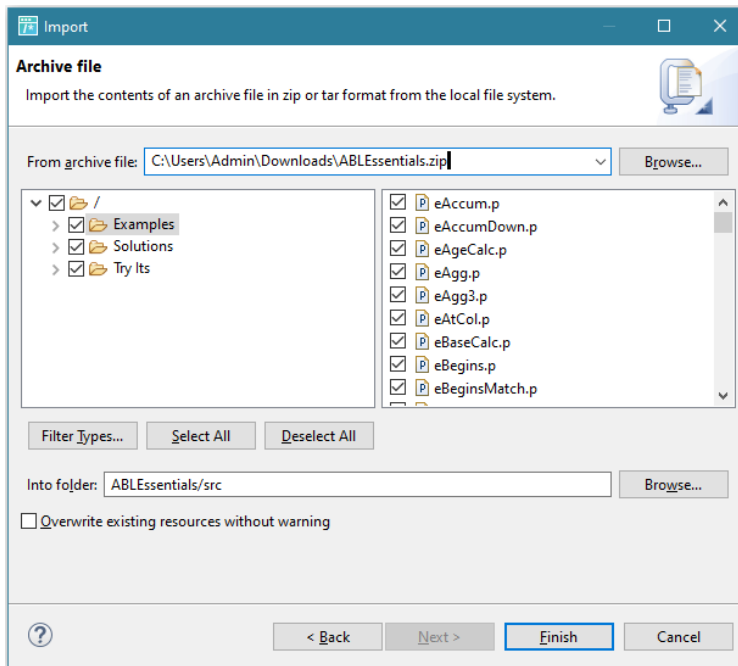
In this course, you will see example code, some of which is available for you to examine and run in the **Examples** project. You will import this project into your workspace and specify that it will use the database connection. You will need the zipped source code file which you can download here, [ABLEssentials.zip](#).

1. In the Project Explorer pane, right click on **src** and select **Import > Import**. The **Import wizard** is displayed.



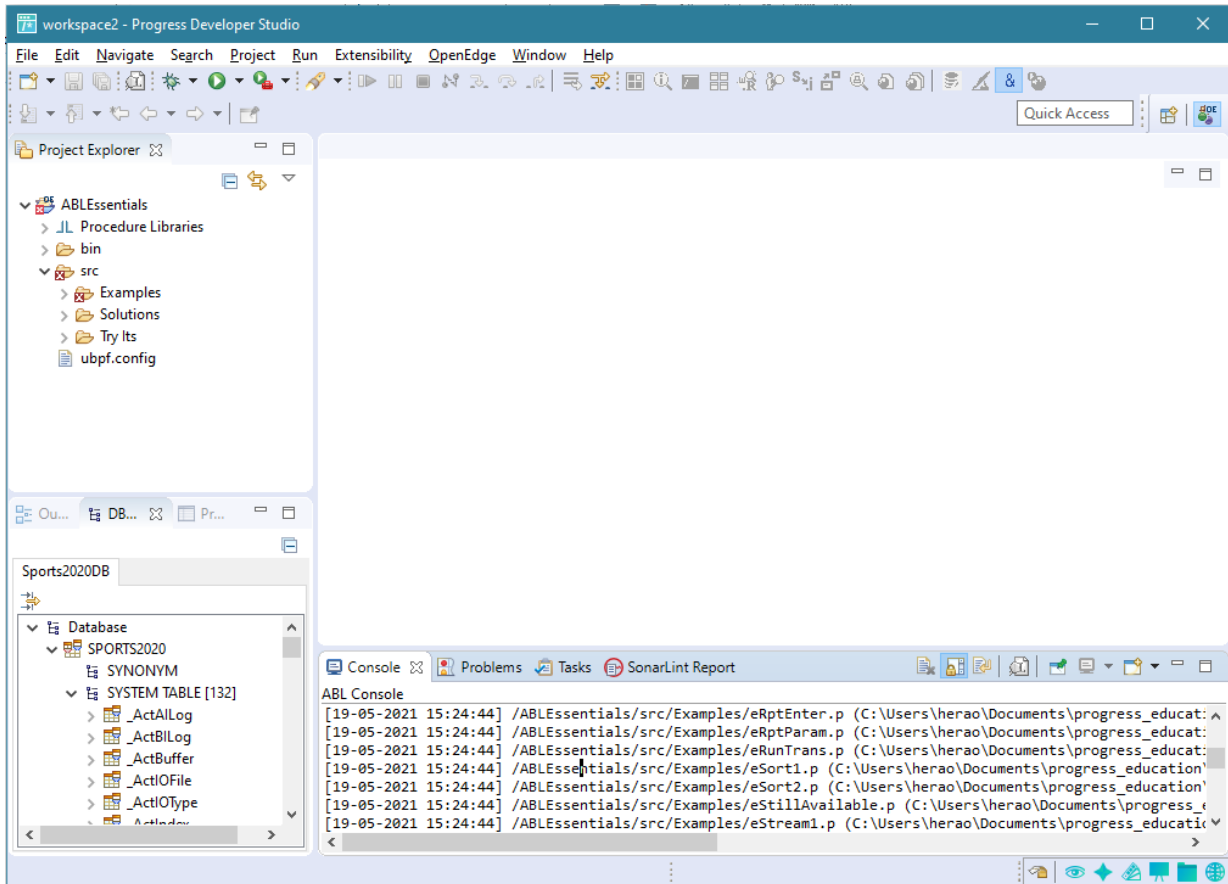


- Expand the **General** node and select **Archive File**. Click **Next**.
- Click **Browse** and navigate to the folder where you downloaded the zipped file, `ABLEssentials.zip`. Select `ABLEssentials.zip` and click **Open**.
- Expand the root folder of the zipped file (`/`) displayed in the left box. All the sub-folders and the files in the folders are selected by default.



Notice that the **Into folder** points to the **ABLEssentials/src** folder in the project workspace.

5. Click **Finish**. The example code files should be imported into the **ABLEssentials** project and visible in the **Project Explorer** tab.



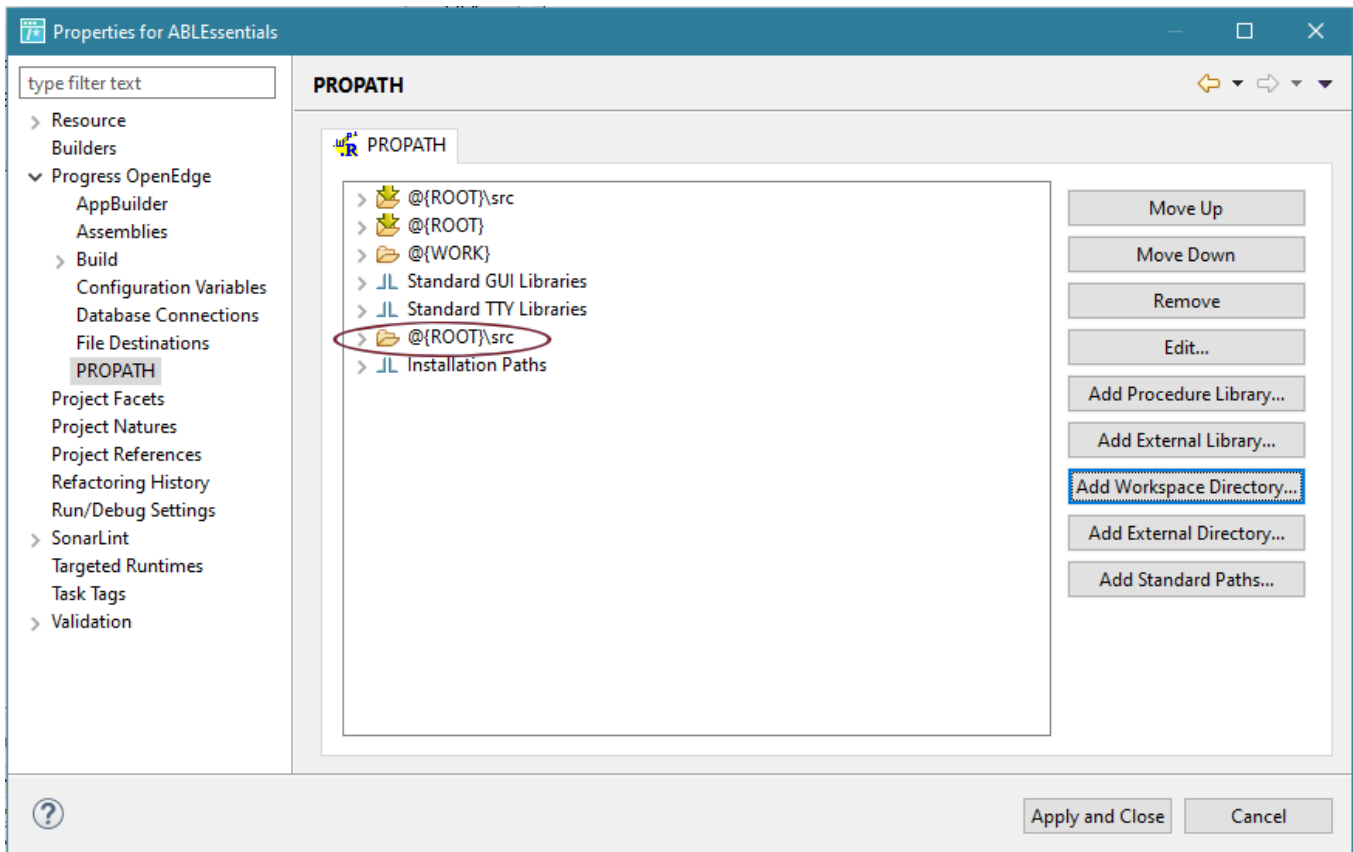
**Note:** If errors (red Xs) appear in the project files, it is because the the AVM could not build those files. You will be fixing the code as part of your learning.

## Part 5—Configure the PROPATH

You should configure the **PROPATH** environment variable with the location of the sample files and libraries you imported into your project space. The **PROPATH** environment variable is used by the AVM at run time and by the Progress OpenEdge Developer studio to find the required sample files containing the required resources.

To set the PROPATH environment variable, perform the following:

1. Right click on your project's root node (**ABLEssentials**) in the Project Explorer and select **Properties**. The **Properties for ABLEssentials** window is displayed.
2. In the left panel expand and select **Progress OpenEdge > PROPATH**.
3. Click **Add Workspace Directory**. The **Select PROPATH Directory** displays.
4. Expand and select **ABLEssentials > src > Examples**.
5. Click **OK**. Click **Apply and Close**. You should see a new entry in the PROPATH list, `@{ROOT}\src`.



## Wrap-up

In this Guided Exercise, you created a workspace to develop and work with ABL code. You set some workspace preferences useful for ABL development. Then you started the database server for the database that is used for this course. Next, you created a simple OpenEdge project that contains your sample code files and configured this project to use the database. Then, you imported the sample code files into your project that you will use during this course. Finally, you set the project PROPATH to point to the sample code files you imported into your project space.

## Try It 1.1: Using FIND, DISPLAY, and REPEAT



In this Try It, you construct ABL statements using the REPEAT block and test them in the Progress Developer Studio.

This Try It should take approximately 10 minutes to complete.

### Before you begin

Before you begin, you must:

1. Install the OpenEdge Developer Studio, see "Set up your exercise requirements" in the *Before you begin* section.
2. Complete "Guided Exercise 1.1: Setting up your development environment" in the *Before you begin* section.

### Location of solution code:

`/progress_education/openedge/ABLEssentials/src/Solutions/Lesson01`

### Part 1—Write statements to find, display and repeat records

Write REPEAT statements to do the following:

1. Find and display all information in the State table. Save the file as `lState.p`.
2. Find and display the first name, last name, and position for all employees. Save the file as `lEmployee.p`.
3. Find and display all information in the SalesRep table except MonthQuota. Save the file as `lSalesrep.p`.

Test your statements by running them in the Progress Developer Studio.

### Wrap-up

In this Try It, you used the REPEAT block to find and display information stored in the different tables in the database.

# Try It 1.2: Using FIND, FOR EACH, and WHERE



In this Try It, you will write ABL code using FIND, FOR EACH, and WHERE statements to retrieve data from the database.

This Try It has three parts and should take approximately 20 minutes to complete.

## Before you begin

Before you begin, you must:

1. Complete *Try It 1.1*.

### Location of solution code:

/progress\_education/openedge/ABLEssentials/src/Solutions/Lesson01

## Part 1—Determine the correct statement

For each of the following scenarios determine if you would use a FIND or FOR EACH statement to get the data outlined in the scenario.

	Scenario	FIND	FOR EACH
a.	All data for the first item in the Item table.		
b.	The name and city for all customers.		
c.	The name of the last SalesRep in the database SalesRep table. <b>Hint:</b> The SalesRep name field is RepName.		
d.	A list of names of all suppliers. <b>Hint:</b> The table name is Supplier.		

Part 2—Retrieve and display data

Write separate procedures to retrieve and display data for each scenario in *Part-1*. Test your procedures by running them in the Progress Developer Studio. Save them with the following names:

- 1. tFirstItem.p
- 2. tCustomers.p
- 3. tLastSalesrep.p
- 4. tSuppliers.p

Part 3—Construct ABL procedures

Write a procedure to find and display the items mentioned in the second column.

	To find and display...	Save procedure as...
a.	The name and country of customers who are <b>not</b> in the USA.	tNotUSA.p
b.	All orders shipped after 6/1/98, with the output in a single column.	tAfter.p
c.	The customer's name and balance for customers with a balance of at least \$30,000.	tCustBalance.p

Wrap-up

In this Try It, you used FIND, FOR EACH, and WHERE statements to find and display information stored in the different tables in the database.

Try It 1.3: Using frames



In this Try It, you create frames, hide frames, add attributes to frames, and pause frames between displays. This Try It has two parts and should take approximately 15 minutes to complete.

## Before you begin

Before you begin, you must:

1. Complete *Try It 1.1* and *Try It 1.2*.

### Location of solution code:

/progress\_education/openedge/ABLEssentials/src/Solutions/Lesson01

## Part 1—Create and display colored frames

Create a procedure with 10 frames, named f1 – f10, with the following criteria:

- Each frame has a title which includes its frame number, such as “Frame 1”.
- The background color of each frame matches its number. For example, frame #1 has a background color of 1.
- The displayed text includes the frame number. For example, frame #1 might display the text “Color #1”.
- Frames number 6 – 10 are centered.
- Display all 10 frames, with a pause between each display.

Save and run the procedure as `tColoredFrames.p`.

### Questions:

1. Where do frames 1 – 5 appear on the default window?
2. Where do frames 6 – 10 appear on the default window?
3. What color is frame 9?

## Part 2—Display and hide frames

Modify the code you created in `tColoredFrames.p` to meet the following criteria:

- After frame 2 displays, hide frame 1.
- After frame 3 displays, hide frame 2.
- After frame 4 displays, hide frame 3.
- After frame 5 displays, hide frame 4.
- After frame 7 displays, view frame 1.
- After frame 8 displays, view frame 3.

Save and run the procedure as `tHideFrames.p`.

### Questions:

1. Which frames are visible on the left side of the default frame, and what is their sequence?
2. Which frames are visible in the center of the default frame?

## Wrap-up

In this Try It, you created frames and controlled them by displaying, hiding and pausing them between displays. You added frame attributes to position and color the frames.

## Try It 1.4: Using BEGINS, CONTAINS, and MATCHES



In this Try It, you determine whether to use BEGINS, CONTAINS, or MATCHES operator to find the requested data.

This Try It has two parts and should take approximately 20 minutes to complete.

### Before you begin

Before you begin, you must:

1. Complete *Try It 1.3*.

#### Location of solution code:

`/progress_education/openedge/ABLEssentials/src/Solutions/Lesson01`

### Part 1—Determine the correct use of BEGINS, CONTAINS and MATCHES statements

Determine which of these tasks can be accomplished using BEGINS, MATCHES, or CONTAINS. Write ABL procedures to retrieve data and test them in the Progress Developer Studio.

1. Find all items with the string “Go” in their name. Display the item name.  
Save the procedure as `tGo.p`.
2. Find items that might be used in water sports and display their name and description.  
Hint: Use `CatDescription`.  
Save the procedure as `tWater.p`.
3. Display the names of customers whose name begins with “Hoo”.  
Save the procedure as `tHoo.p`.
4. Find and display the name of all customers with “tennis” in their name.  
Save the procedure as `tTennis.p`.



## Part 2—Bonus Tasks

1. Write a procedure to retrieve all customers in the USA whose postal code begins with a zero (0) or a nine (9). Display the customer name, country, and postal code. Save the procedure as `tUSA09.p`.
2. Read the output to make sure you have only the customers you think you should have. (If you see “Finland” and “France”, you need to modify your code.)

## Wrap-up

In this Try It, you used the `BEGINS`, `CONTAINS`, or `MATCHES` operator to find the required data.

# Try It 1.5: Sorting records



In this Try It, you will sort the records in a table based on specified fields.

This Try It has should take approximately 15 minutes to complete.

## Before you begin

Before you begin, you must:

1. Complete *Try It 1.4*.

Location of solution code:

`/progress_education/openedge/ABLEssentials/src/Solutions/Lesson01`

## Part 1—Sorting records on select fields

1. Generate a list of employees by department, then, within each department, by last name, then by first name. Do not include any other fields in the output.

**Hint:** The field for Department in the Employee table is DeptCode.

Save the procedure as `tEmployeeNames.p`.

2. Generate a list of all customers who live in the USA. Sort by, and display, their states, cities and names. Format the output to allow 15 characters for the city, two characters for the state, and 30 characters for the name

Save the procedure as `tUSCustomers.p`.

3. List the family members, from youngest to oldest, of all employees, sorted by birthdates. Include the employee number in the output.

Save the procedure as `tBirthdates.p`.

## Wrap-up

In this Try It, you retrieved records from a database table and sorted the records based on specified fields.

## Try It 1.6: Displaying messages in a dialog box



In this Try It, you display messages in different dialog boxes based on the alert type.

This Try It has should take approximately 15 minutes to complete.

## Before you begin

Before you begin, you must:

1. Complete *Try It 1.5*.

### Location of solution code:

`/progress_education/openedge/ABLEssentials/src/Solutions/Lesson01`

## Display messages

Display messages in dialog boxes to do the following:

1. Display the message "Here is my message".

Save the file as `tMyMessage.p`.

2. Find the department whose department code is 700.

Display a message that shows the name of the department for department 700. The department code field is of `CHARACTER` data type.

Save the file as `tDepartment.p`

3. Find all customers whose customer number is less than 3040.

Display a message that shows the name of the customer and the contents of the **Comments** field.

Add a colon (:) between the customer name and the comment.

Save the file as `tComments.p`

4. Find all the suppliers who have the word "Sports" in their name.

Display a message that states "Supplier <number> is <supplier name>", where the number and supplier name are replaced with the correct data.

Save the file as `tSportsSupplier.p`

## Wrap-up

In this Try It, you displayed message dialogs.

# Try It 2.1: Define variables



In this Try It, you will practice how to choose a data type for a variable and how to define variables using the ABL data types.

This Try It has should take approximately 15 minutes to complete.

## Before you begin

Before you begin, you must complete the Try Its in lesson 1.

### Location for the solution code:

/progress\_education/openedge/ABLEssentials/src/Solutions/Lesson02

## Part 1—Specify the data type

Specify ABL data types you would use to hold the following values:

Value	Data type
151	
151.19	
One hundred fifty two	
True	
10/10/04	
NO	
January, February, March	
December 31, 2001	
365	

## Part 2—Define variables

Perform the following instructions, for each of the scenarios listed after the instructions:

- Choose a data type for the variable.
  - Write a statement to define the variable.
  - Apply an appropriate prefix to the variable name based on a naming convention of your choice.
  - Type your variable definitions in Progress Developer Studio to check your syntax.
1. Define a variable to be used as a counter that starts from zero.
  2. Define a variable to store monthly sales total with two decimal places.
  3. Define a variable to store a value of a country with the initial value USA.
  4. Define a variable to store a value of either Yes or No.
  5. Define a variable to store an employee's date of birth.

6. Define a variable to store a name with the initial value as your name. Specify a display format that is long enough to hold your first and last names.
7. Define a variable to store decimal values. Specify a display format for three positions before and two positions after the decimal point.

### Wrap-up

In this Try It, you practiced how to choose a data type for a variable and how to define a variable using the ABL data types.

## Try It 2.2: Dates and numbers



In this Try It, you will practice how to use date and numeric functions.

This Try It has three parts should take approximately 30 minutes to complete.

### Before you begin

Before you begin, you must complete *Try It 2.1*.

#### Location for the solution code:

`/progress_education/openedge/ABLEssentials/src/Solutions/Lesson02`

### Part 1—Date functions

Suppose you have a date field called `StartDate` in the employee table:

1. What date function would return the month that this employee began work?
2. How would you find out the year the employee started work?

### Part 2—Numeric functions

1. Write `DISPLAY` or `MESSAGE` statements to view the results for each of the following.

Description	ABL Expression
Display 256.1234 with two decimal places.	
Convert 15.23 to an integer value.	
Calculate the remainder when you divide 29 by 3.	

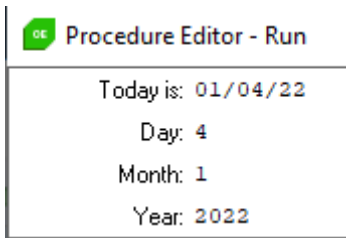
2. What does `dAmount` represent in the following code sample?

```
DEFINE VARIABLE dAmount AS DECIMAL NO-UNDO.
ASSIGN dAmount = (orderLine.Price * Orderline.Qty)
               * ((100 - Orderline.Discount) / 100).
```

3. Write a procedure named `tToday.p` to display the following in a single column:

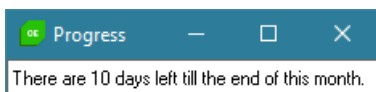
- Today's date
- The day of the month
- The month
- The year

Your output should look something like this:



4. Write a procedure that calculates the number of days until the end of the month.

Display the output with a message like the following:



Save the file as `tEndMonth.p`.

### Part 3—Dates and numbers (Bonus questions)

1. Write a procedure to calculate how many seconds until midnight tonight. Display the results.

Save the file as `tMidnight.p`.

2. Modify `tMidnight.p` to display the number of hours, minutes, and seconds until midnight.

### Wrap-up

In this Try It, you practiced using some of the date and numeric functions.

## Try It 2.3: Data types, comparisons, strings, lists, and functions



In this Try It, you will practice how to compare values, work with string variables, and access items in a list. This Try It has four parts should take approximately 30 minutes to complete.

### Before you begin

Before you begin, you must:

1. Complete *Try It 2.2*.

#### Location for the solution code:

`/progress_education/openedge/ABLEssentials/src/Solutions/Lesson02`

### Part 1—Compare values

Write ABL expressions using the comparison symbols for the listed conditions:

1. The customer's credit limit is less than or equal to the customer's balance.
2. The customer's sales representative is JAL and the customer's country is not the USA.
3. The customer's number is less than or equal to 10.
4. The customer's city is either Boston or Bedford.

Use the field names from the customer table: `CustNum`, `CreditLimit`, `Balance`, `Country`, `City`, and `Salesrep`.

### Part 2—Strings and lists

1. Define a character variable to hold the value of a telephone number in your country, for example, 781-555-1111.
2. Specify the part of the telephone number that represents the area of the country where this number is located (such as Paris, northern Australia, the state of New Jersey, western Massachusetts). In the US, this is called the area code.

3. Write a display statement to view your results. For example, the displayed message might be:  
**My area code is 781.**
4. Save the procedure as `tPhoneNum.p`.

### Part 3—Access items in a list

1. Define a variable to hold a list of several of your favorite kinds of beer, wine, or soft drink.
2. Write statements to display:
  - The number of items in the list
  - The third item in the list

For example, the list might include the following items: Zinfandel, Sake, Port, Chardonnay, Chianti, Merlot, Shiraz, and the displayed message would be:

**There are 7 entries in the list. The third entry is Port .**

3. Save the file as `tList.p`.

### Part 4—Bonus question

Review this procedure and guess what the output will be if you enter your own birth date as the initial value of `dtBdate`. (In the US, the date format is mm/dd/yyyy.)

```
/* lbdate.p */

DEFINE VARIABLE bdate AS DATE INITIAL 08/16/65.
DEFINE VARIABLE wday AS CHARACTER
    INITIAL "Sunday,Monday,Tuesday,Wednesday,Thursday,Friday,Saturday".
DEFINE VARIABLE yold AS INTEGER.

ASSIGN yold = TRUNCATE(((TODAY - bdate) / 365.25), 0).
DISPLAY SUBSTITUTE("You were born on a &1.", ENTRY(WEEKDAY(bdate), wday)) FORMAT "X(30)"
SKIP(1)
    SUBSTITUTE("You are &1 years old.", yold) FORMAT "X(50)".
```

Enter your birth date in `tBdate.p`. Run the procedure and see if the output matches what you anticipated.

---

**Note:** Solutions are not provided for bonus question. They are only for practice.

---

### Wrap-up

In this Try It, you practiced how to compare values, work with string variables, and access items in a list.



## Try It 2.4: Conditional logic



In this Try It, you will practice how to write conditional and branching logic statements.

This Try It has four parts should take approximately 30 minutes to complete. Perform the following steps for the scenario described in each part.

- Determine if the logic is best expressed using an `IF . . THEN` statement or a `CASE` statement.
- Write the statement and test it.

### Before you begin

Before you begin, you must:

1. Complete *Try It 2.3*.

#### Location for the solution code:

`/progress_education/openedge/ABLEssentials/src/Solutions/Lesson02`

### Part 1—Scenario 1

If today is Monday or Tuesday, display a message that it is the beginning of the week. If today is Wednesday, display a message that it is Wednesday. If today is Thursday or Friday, then display a message that the weekend is coming soon. If today is Saturday or Sunday, display a message that it is the weekend.

Save the file as `tWeekday.p`.

---

**Note:** Use the `WEEKDAY` function to help you write your code.

---

### Part 2—Scenario 2

Search the **Family** table for all records where the employee number is less than or equal to 20, and the family member (field **Relation**) is either son or daughter.

Display the employee number, relationship of the employee's family member, and the relation's name and gender.

Save your file as `tFamily.p`.

### Part 3—Scenario 3

Display an alert box with a message, Even numbers from 2 to 20:. The alert box shows the number 2 during the first iteration, then shows the next even number at the next iteration, and so on, until 20 is displayed on the last iteration.

- Use a `DO WHILE` loop to set up the counter. Save the file as `tCountingWhile.p`.
- Use a `DO BY` loop to set up the counter. Save the file as `tCountingDo.p`.

### Part 4—Scenario 4

Use an alert box to determine whether your user speaks French (or any language of your choosing). If the response is positive, display a message in that language. If the response is negative, display a message in English. Save the file as `tFrench.p`.

### Wrap-up

In this Try It, you identified and used the appropriate conditional and branching logic statements for the given scenarios.

## Try It 3.1: Write a simple query using both functions



In this Try It, you will practice how to write simple queries to retrieve data from a database.

This Try It should take approximately 15 minutes to complete.

### Before you begin

Before you begin, you must:

1. Complete *Guided Exercise 1.1: Setting up your development environment*.

#### Location of solution code:

`/progress_education/openedge/ABLEssentials/src/Solutions/Lesson03`

## Write simple queries

1. Write a procedure that defines a query for the Item table and displays the item number, name, price, and number of items available.
  - Use the `IF NOT AVAILABLE` statement for the query.
  - Save the procedure as `tQueryItem.p`.
2. Write a procedure that defines a query that retrieves every employee in department code 500 (Training). Display the employee number, first name, last name, and department code.
  - Use the `QUERY-OFF-END` function.
  - Save the procedure as `tQueryEmployee.p`.

---

**Note:** Your queries should display all items and work without any error even if someone has deleted a record from the item table after `OPEN QUERY` is executed (i.e. the ROWID of this record is in the result list of the query but the record itself is no longer in the database).

---

## Wrap-up

In this Try It, you wrote simple queries to retrieve data from a database. You used the `IF NOT AVAILABLE` function to check that the required record is available in the record buffer before you retrieve it. You also used the `QUERY-OFF-END` function to check for the last record in the record buffer.

## Try It 3.2: Join tables in a query



In this Try It, you will write a query to join two tables.  
This Try It should take approximately 10 minutes to complete.

## Before you begin

Before you begin, you must:

1. Complete *Try It: 3.1*.

### Location of solution code:

/progress\_education/openedge/ABLEssentials/src/Solutions/Lesson03

## Join tables

1. Create a query to:
  - a. Find the orders for each customer.
  - b. Sort the results list by customer number and order date.
  - c. Display the customer number, customer name, order number, and order date.
2. Save the procedure as `tMult.p`.
3. Modify the code in `tMult.p` to show only the first order for each customer.
4. Save the procedure as `tMult2.p`.

## Wrap-up

In this Try It, you wrote a query to create a join of two tables.

# Try It 3.3: Scroll a query results list



In this Try It, you will create a scrolling query and use the `REPOSITION` statement to reposition the cursor in a browse widget.

---

**Note:** The file `lEmpFam.w` has a browse control that displays family members of employees. It includes a **First**, **Last**, and **Done** button.

---

This Try It should take approximately 10 minutes to complete.

## Before you begin

Before you begin, you must:

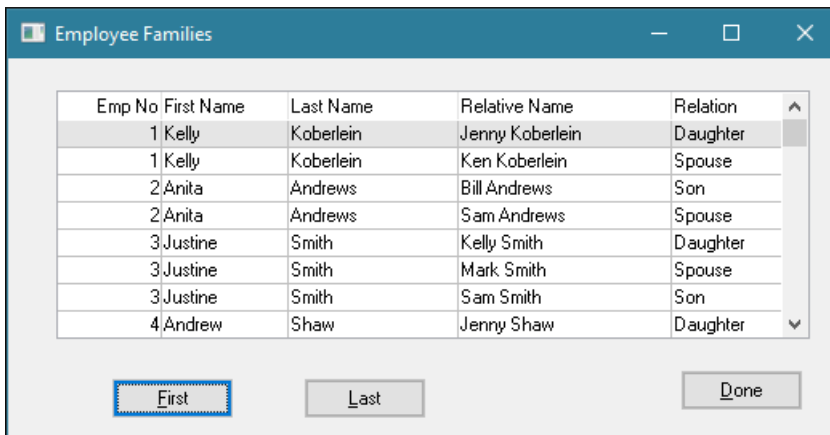
1. Complete *Try It: 3.2*.

### Location of solution code:

/progress\_education/openedge/ABLEssentials/src/Solutions/Lesson03

## Update a scrolling query

1. In the **Project Explorer** pane, expand **ABLEssentials > Src > Try Its**.
2. Right-click the file `lEmpFam.w` and select **Run As > Progress OpenEdge Application**.
3. The following dialog box is displayed:



Notice that you can scroll up and down the results list in the browse.

4. Click the **First** and **Last** buttons. Notice that they do nothing.
5. Click the **Done** button to close the dialog box.
6. Right-click `lEmpFam.w` and select **Open with > Other ...**. The **Editor Selection** dialog displays.
7. Select **OpenEdge ABL Editor**. The source editor opens and displays the code for `lEmpFam.w`.
8. Add code to make the **First** button (**BtnFirst**) reposition the browse to the first record.
9. Add code to make the **Last** button (**BtnLast**) reposition the browse to the last record.
10. Test your code by running the procedure.

## Wrap-up

In this Try It, you used a scrolling query and the Browse widget. You used the `REPOSITION` statement to position the cursor to the first and last record in the result list.

## Guided Exercise 4.1: Generate a database sequence report



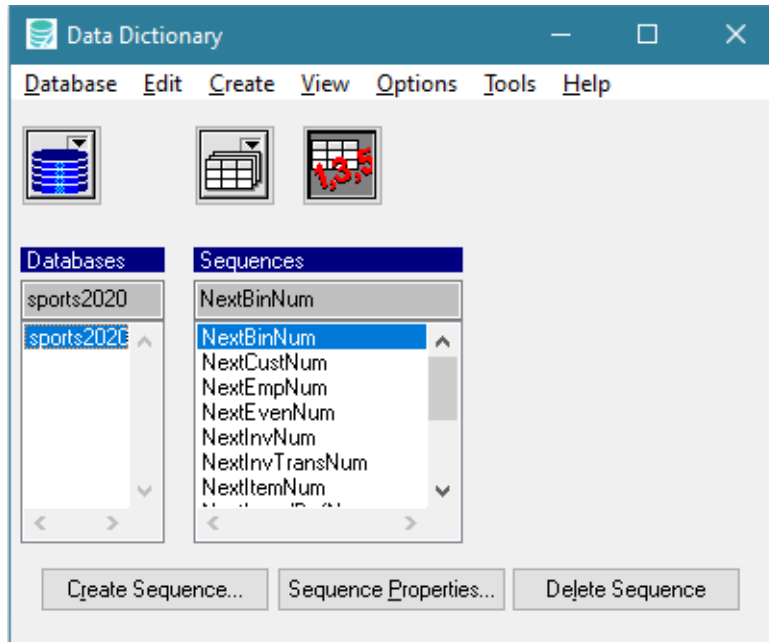
To generate a report of the existing sequences in your database, perform the following steps in Progress Developer Studio:

1. Select **OpenEdge > Admin > Data Dictionary**.

The **Data Dictionary** window is displayed.

2. Click the sequence icon .

The list of sequences defined in the database are listed.



3. Select **Database > Reports > Sequence**.

The sequence report is displayed.

The screenshot shows the 'Sequence Report' window. It contains a table with the following data:

Sequence Name	Initial Value	Increment	Max/Min Value	Cycle?
NextBinNum	1000	1	? no	
NextCustNum	1000	5	? no	
NextEmpNum	10	1	? no	
NextEvenNum	2	2	? no	
NextInvNum	100	1	? no	
NextInvTransNum	100	1	? no	
NextItemNum	10	10	? no	
NextLocalDefNum	10	1	? no	
NextOrdNum	10000	5	? no	
NextPONum	10000	1	? no	
NextRefNum	10	1	? no	
NextSupplNum	10	1	? no	
NextWareNum	10	1	? no	

At the bottom of the window are buttons for 'OK', 'Print', and 'Help'.

The report lists the sequence name, initial value, increment, maximum or minimum value, and whether the numbers should cycle after they reach that value.


## Guided Exercise 4.2: Create a sequence

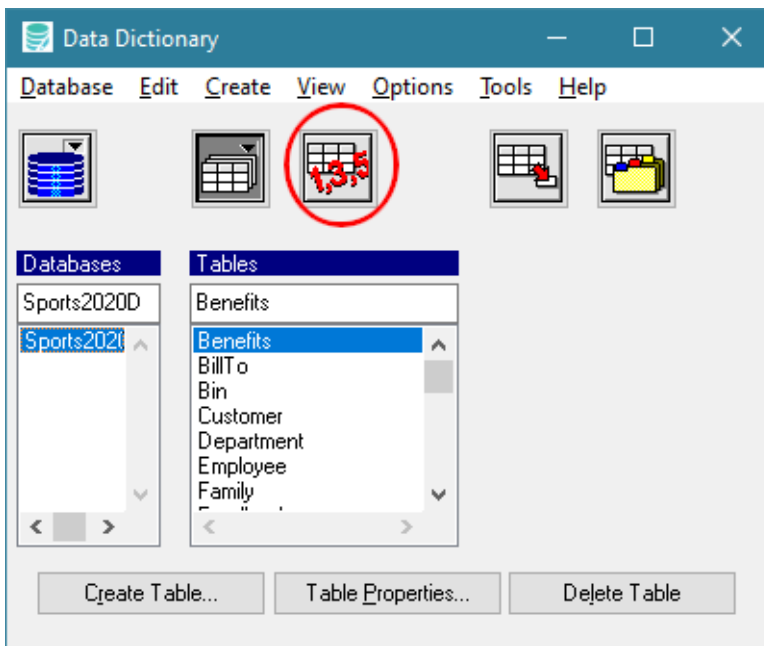


To create a sequence:

1. In Progress Developer Studio, select **OpenEdge > Admin > Data Dictionary**.

The **Data Dictionary** window is displayed.

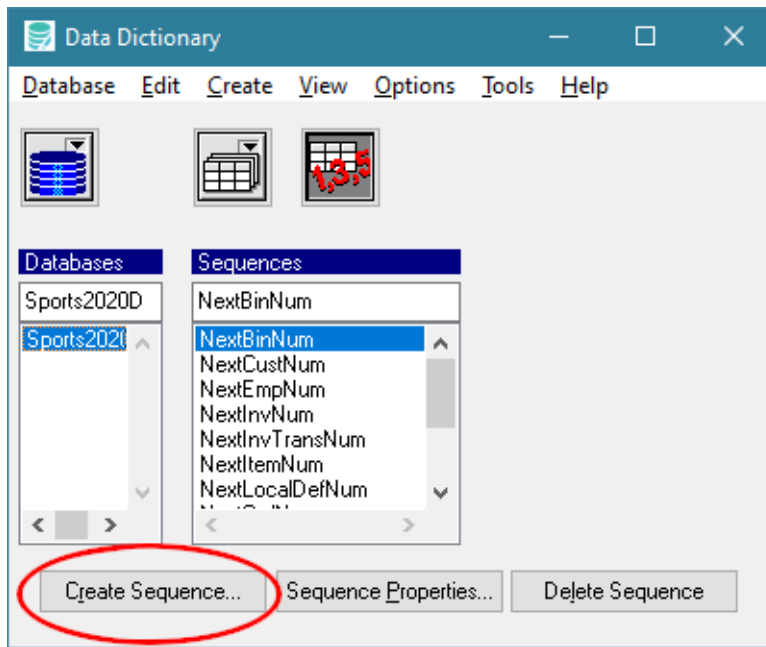
2. In the data dictionary window, click the sequence icon .



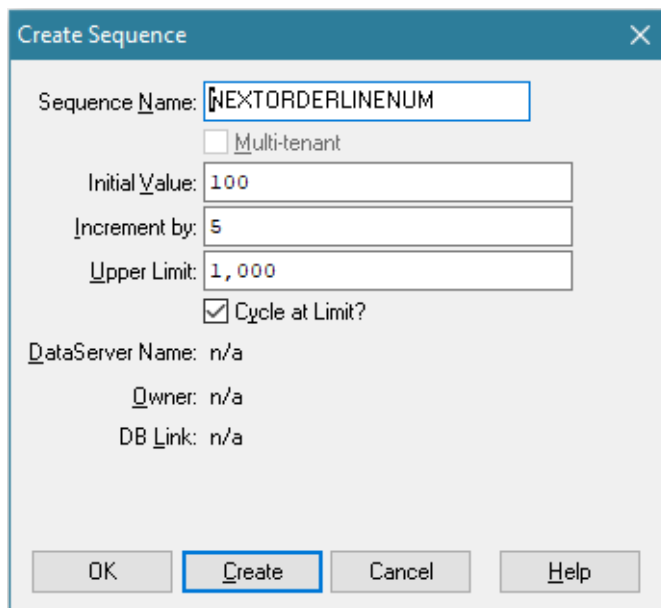
3. In the sequence window, click **Create Sequence**.

The **Create Sequence** window is displayed.



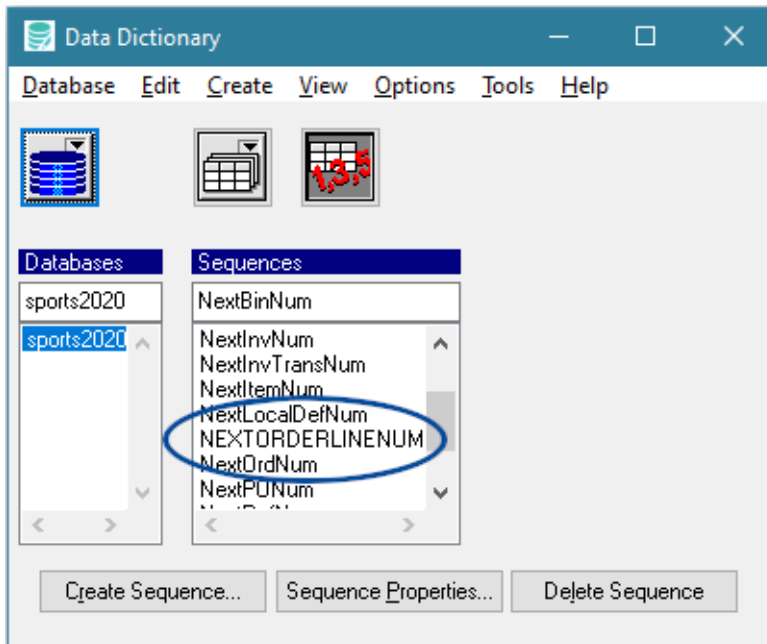


4. Create a new sequence called **NEXTORDERLINENUM**, and edit the field values in the sequence window, as shown in the following figure:

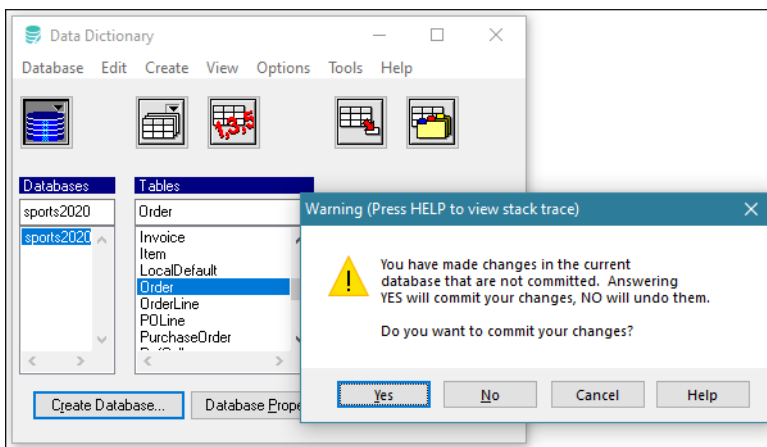


5. Click **Create**.

The new sequence is displayed in the list of sequences.



6. Close the **Data Dictionary** and select **Yes** to commit your changes in the **Warning** dialog.

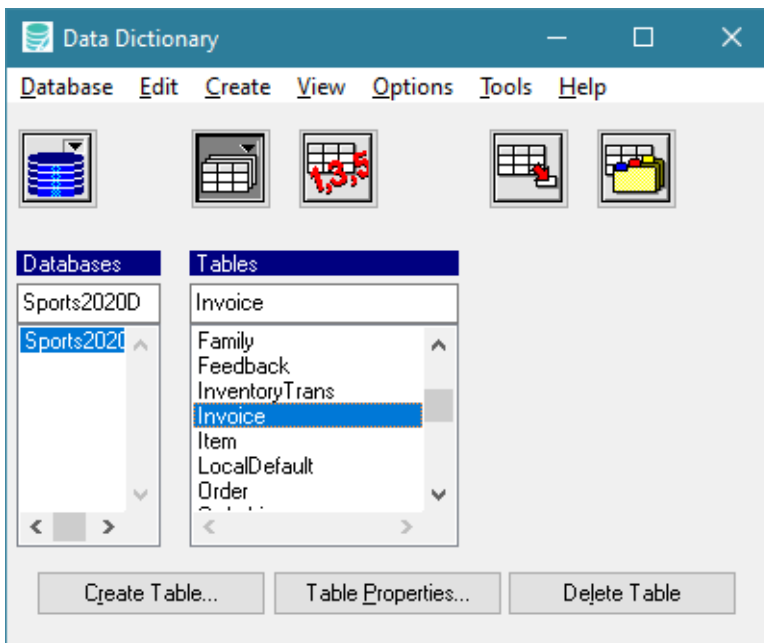


## Guided Exercise 4.3: Locate trigger code

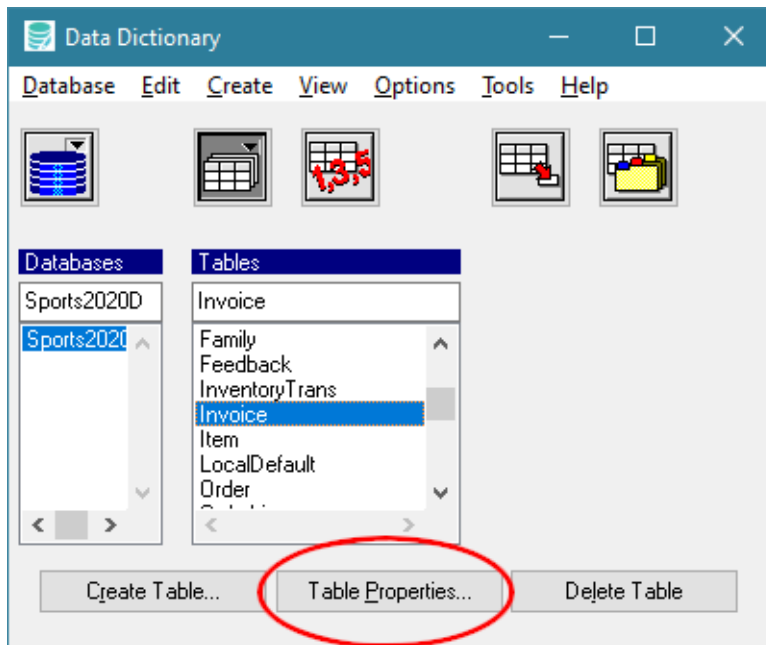


To access the trigger code for a table, follow these steps:

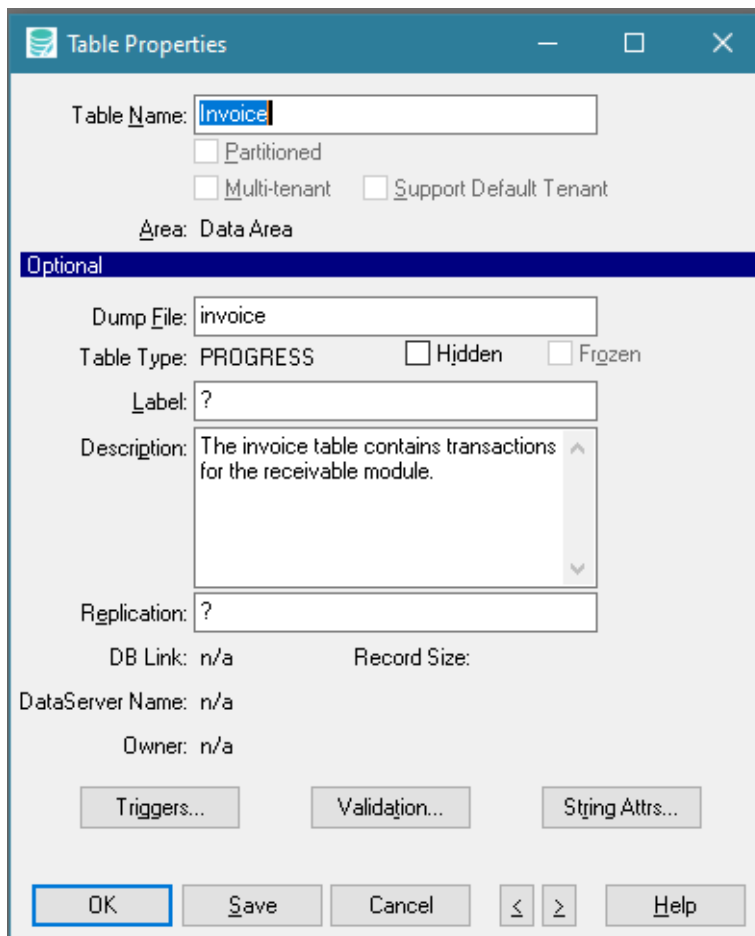
1. In Progress Developer Studio, select **OpenEdge > Admin > Data Dictionary**.  
The **Data Dictionary** window is displayed.
2. Select the required table. The **Invoice** table is selected in the following example.



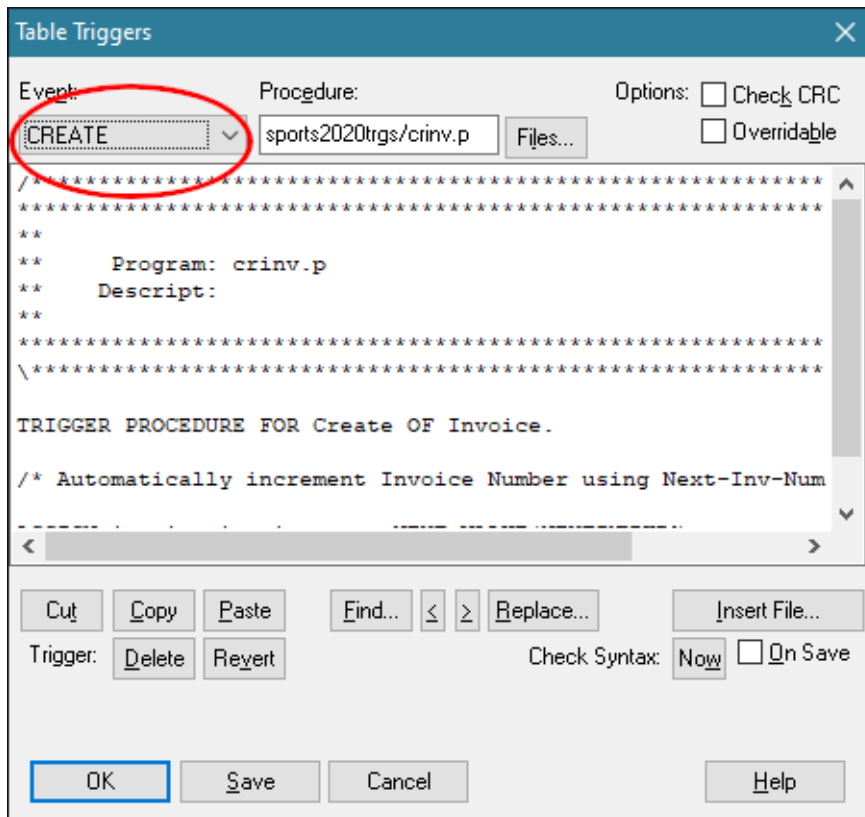
3. Click Table Properties.



4. In the **Table Properties** dialog, click **Triggers**.



- By default, the first trigger for the table is displayed. For the **Invoice** table, this will be the `CREATE` trigger, as shown in the Event field:



- To access a different trigger, select it from the **Event** drop-down list. From here you can view or edit the trigger code in the editor.
- Press **Cancel** to exit the **Table Triggers** window.
- Press **Cancel** to exit the **Table Properties** window.
- Close the **Data Dictionary** window.

## Try It 4.1: Create a new record



In this Try It, you will write code to assign `Orderline` numbers automatically when a new order is created..

In a previous guided exercise, you saw how a sequence could be used to generate the next invoice number automatically. The `Orderline` table presents a different situation. `Orderlines` are incremental and `Orderline` numbers are unique to an order but are not unique within the `Orderline` table. Every `Order` record can have an `OrderLine` with a line number of 1. Automating the process of generating `Orderline` numbers is convenient and less error prone.

This Try It should take approximately 30 minutes to complete.

### Before you begin

Before you begin, you must:

1. Complete *Guided Exercise 1.1: Setting up your development environment*.

#### Location of solution code:

`/progress_education/openedge/ABLEssentials/src/Solutions/Lesson04`

### Create a record

Write code to assign `Orderline` numbers automatically when a new order is created.

1. Create a new procedure called `tNewLineNum.p` in Progress Developer Studio.
2. Define a variable to store the value of the `Orderline` number.
3. Write code to find the latest `Order` in the database.
4. Write code to find the last `Orderline` for that `Order`.

---

**Note:** Find the `Orderline` record using the `NO-ERROR` option and test for the availability of a line item.

---

5. Determine the appropriate value for the next line number and assign it to your variable. If this is the first line number, assign it a value of 1.

6. Create a new `Orderline` record and populate it with the appropriate **OrderNum** and **LineNum**. Also assign some values for **ItemNum**, **Price**, and **Qty**.
7. Display the `Orderline` record.
8. Save the procedure.
9. Run the procedure and verify the output is as expected.

### Wrap-up

In this Try It, you automated the process of generating `Orderline` numbers when a new order record is created.

## Try It 4.2: Modify a sequence



The Sports2020 database has many sequences defined, but you might want to define different sequences for different requirements.

In this Try It, you will modify a sequence.

This Try It should take approximately 10 minutes to complete.

### Before you begin

Before you begin, you must:

1. Complete *Try It 4.1*.
2. Complete *Guided Exercise 4.2: Create a sequence*.

#### Location of solution code:

`/progress_education/openedge/ABLEssentials/src/Solutions/Lesson04`

### Modify a sequence

The next sequence number for a new customer is currently incremented by 5. You will change the sequence to increment by 10.

1. Write a procedure named `tNextCustNum.p` to perform the following:

- a. Display the customer number and name of the last customer.
- b. Create a new customer, assign a name to the customer and display the customer number and name..

---

**Note:** You must build in a pause to see both records displayed.

---

2. Run the procedure and note the customer number assigned to the new customer.
3. Repeat Step 2, assign a different name, and note the customer number assigned.
4. Verify that the customer number increments by 5.
5. Modify the sequence such that the customer number increments by 10.
6. Run the procedure again.

**Question:** Did the new customer number increment by 5 or 10?

### Wrap-up

In this Try It, you changed the value of the sequence number generated by using an increment value.

## Try It 4.3: Modify a database trigger



The Sports2020 database has many database triggers defined, but you might want to define triggers with different functionality.

In this Try It, you will modify a database trigger.

This Try It should take approximately 10 minutes to complete.

### Before you begin

Before you begin, you must:

1. Complete *Try It 4.2*.
2. Complete *Guided Exercise 4.3: Locate trigger code*.



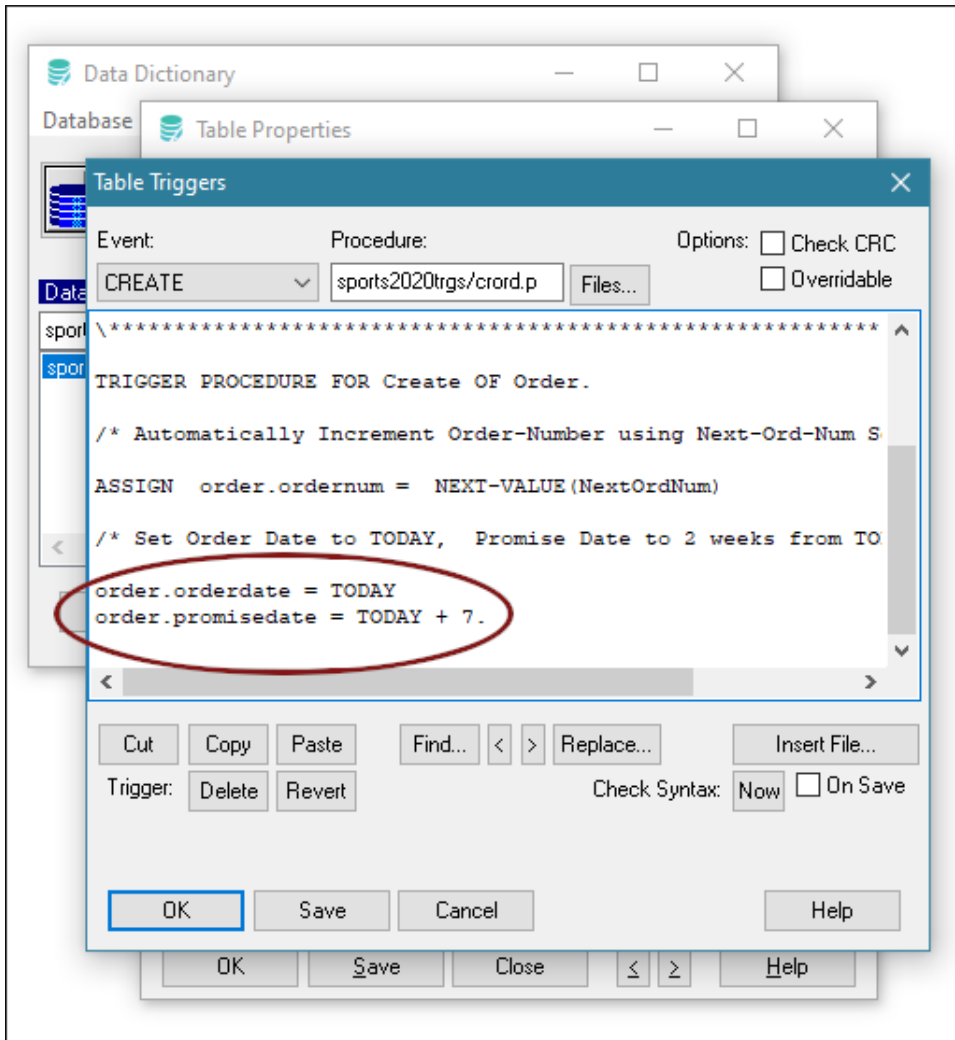
**Location of solution code:**

/progress\_education/openedge/ABLEssentials/src/Solutions/Lesson04

**Modify a database trigger**

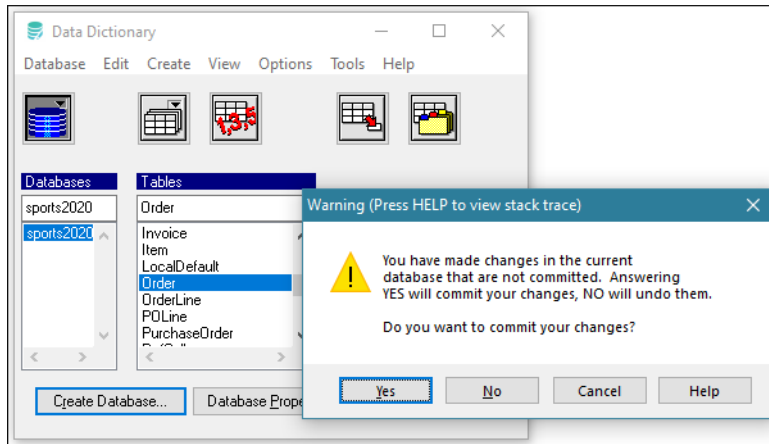
The trigger code to create an order automatically sets the promise date to 14 days from the current date. Your company improved its distribution methods and is now promising to deliver within 7 days. You need to change the database trigger for this purpose.

1. Write a procedure called `tCreateOrder.p` to:
  - a. Create an order.
  - b. Display the order number, order date, and promise date.
  - c. Run the procedure. Verify that the order date is today and that the promise date is 14 days later.
2. Open the database trigger (`crord.p`) associated with creating an order.
3. Change the promise date from 14 days to 7 days from the current date.



4. Click **Save** and click **OK** to close the **Table Triggers** dialog.
5. Click **OK** to close the **Table Properties** dialog.

6. Close the **Data Dictionary** and select **Yes** to commit your changes in the **Warning** dialog.



7. Run `tCreateOrder.p`.

**Question:** Is the promise date displayed 7 days or 14 days from today?

### Wrap-up

In this Try It, you modified a database trigger code to automatically set the promised date of delivery to 7 days from the date on which the order is placed.

## Try It 4.4: Modify records and assign values



In this Try It, you will modify the existing `Orderline` records to reflect a price change. This Try It should take approximately 10 minutes to complete.

## Before you begin

Before you begin, you must:

1. Complete *Try It 4.3*.

### Location of solution code:

/progress\_education/openedge/ABLEssentials/src/Solutions/Lesson04

## Modify a record

Your company decided to increase the price of its products by 25%. `Order# 10000` was created using the old price list. Modify `Order# 10000` to reflect the new prices.

Write a procedure called `tIncreasePrice.p` to:

1. Store the original price as a variable.
2. Increase the price in each `Orderline` for `Order# 10000` by 25%.
3. Display the line number, original price, and revised price.

---

**Note:** Provide meaningful labels for the two price columns.

---

4. Save the procedure.
5. Run the procedure. Verify the output is as expected.

## Wrap-up

In this Try It, you modified the existing `Orderline` records for an order to reflect the new price of products.

# Try It 4.5: Delete records



In the `Sports2020` database, the `DELETE` trigger on the `customer` table prevents deletion of a `Customer` record if there are outstanding orders or invoices.

In this Try It, you will write a procedure that successfully deletes a customer record.

This Try It should take approximately 20 minutes to complete.

### Before you begin

Before you begin, you must:

1. Complete *Try It 4.4*.

#### Location of solution code:

/progress\_education/openedge/ABLEssentials/src/Solutions/Lesson04

### Delete a record

Follow these steps to write a delete trigger. Use Customer No: 3255 as the parent record.

1. Update `lDelFail.p` with `Customer.CustNum = 3255.` in the first line of code.
2. Run `lDelFail.p` to see the messages when a customer cannot be deleted due to open orders.
3. Identify the parent-child relationships that prevent the deletion of the parent record.
4. Write a procedure called `tDelParentChild.p` to:
  - a. Find the child records (*Invoice* and *Order*) of Customer No: 3255 and delete them first. Include messages to identify what is happening.
  - b. Delete the parent record Customer No: 3255.
  - c. Save and test your procedure.

### Wrap-up

In this Try It, you wrote a procedure to successfully delete a customer record after checking that it does not have any child records (outstanding order records or invoice records) associated with it.

## Guided Exercise 5.1 : Generate a compile listing



This guided exercise describes how to generate a compile listing.

1. Examine the procedure, eTrans1.p, and see if you can identify the transaction scope:

```

/*****
 * eTrans1.p
 * Identify the transaction scope.
 * Use the following statement to verify answer.
 * compile eTrans1.p listing eTrans1.lst.
 *****/

```

FOR EACH Customer:

```

  DISPLAY Customer.Custnum.
  UPDATE Name CreditLimit.

```

FOR EACH Order OF Customer:

```

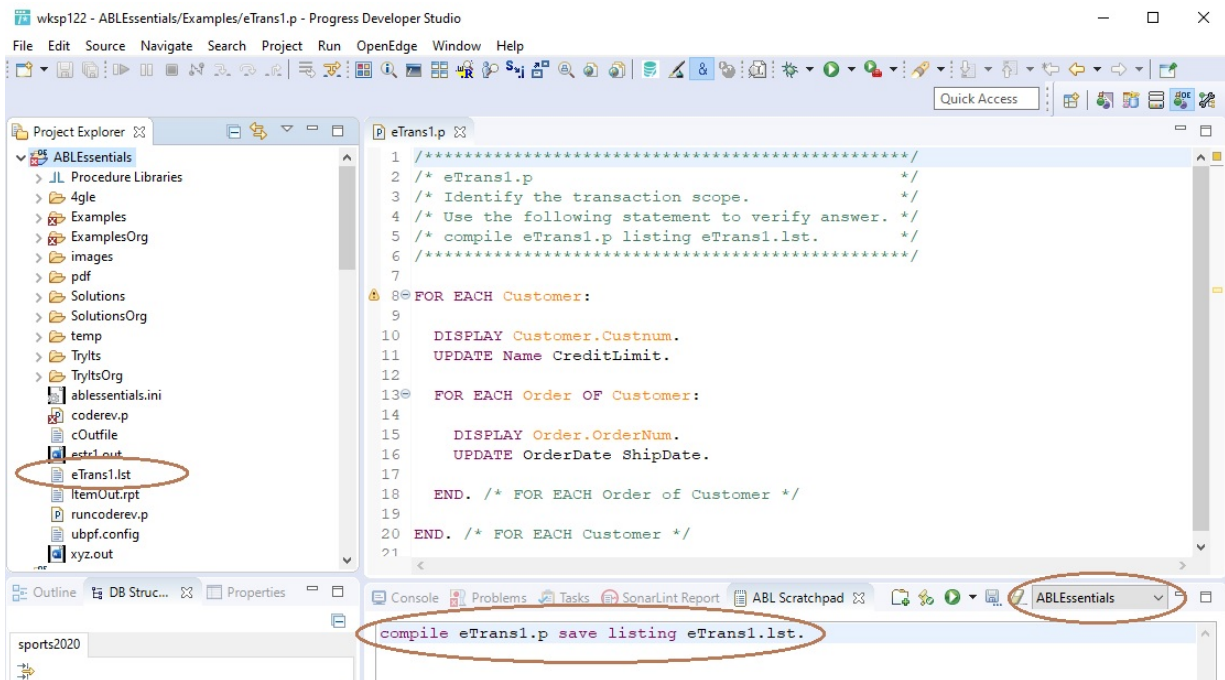
  DISPLAY Order.OrderNum.
  UPDATE OrderDate ShipDate.

```

END. /\* FOR EACH Order of Customer \*/

END. /\* FOR EACH Customer \*/

2. Select **Windows > Show View > ABL Scratchpad** to open the **ABL Scratchpad** tab in the lower panel.



3. Ensure that your current project is selected from the drop-down list that is at the top right of the **ABL Scratchpad**. See the figure in step2.
4. Type the following command in the ABL Scratchpad and click the **Run** icon (▶), to generate a compile listing for eTrans1.p.  
  

```

COMPILE eTrans1.p SAVE LISTING eTrans1.lst

```
5. In the **Project Explorer** pane, right-click and select **Refresh**. The compile listing file eTrans1.lst appears under the **ABLEssentials** project node. See the figure in step2.
6. Double-click eTrans1.lst to open it in the editor view.

Your file should look like this:

```
...xamples\eTransl.p                               12/28/2021 15:46:28    PROGRESS(R) Page 1

{ } Line Blk
-- ----
1  /******
2  /* eTransl.p                                     */
3  /* Identify the transaction scope.                */
4  /* Use the following statement to verify answer. */
5  /* compile eTransl.p listing eTransl.lst.        */
6  /******
7
8  1 FOR EACH Customer:
9  1
10 1   DISPLAY Customer.Custnum.
11 1   UPDATE Name CreditLimit.
12 1
13 2   FOR EACH Order OF Customer:
14 2
15 2       DISPLAY Order.OrderNum.
16 2       UPDATE OrderDate ShipDate.
17 2
18 1   END. /* FOR EACH Order of Customer */
19 1
20  END. /* FOR EACH Customer */
```

...xamples\eTransl.p 12/28/2021 15:46:28 PROGRESS(R) Page 2

File Name	Line Blk. Type	Tran	Blk. Label
...xamples\eTransl.p	0 Procedure	No	
...xamples\eTransl.p	8 For	Yes	
Buffers: sports2020.Customer			
Frames: Unnamed			
...xamples\eTransl.p	13 For	Yes	
Buffers: sports2020.Order			
Frames: Unnamed			

7. The Tran column in the report informs you that there is:

- No transaction active at the procedure block level.
- No transaction active in the FOR block on line 8.
- An active transaction in the FOR block on line 9.

## Guided Exercise 5.2: Using the TRANSACTION function to identify scope



The following example, `eIsActive.p`, displays message statements telling you whether a transaction is active or inactive at three points in the procedure:

- Before the start of `Orderblock`, the `REPEAT` block for `Orders`
  - After the start of `Orderblock`
  - After `Orderlineblock`, the `REPEAT` block for `Orderlines`
1. Read the following procedure `eIsActive.p` and try to identify the points where a transaction is active:

```
/* eIsActive.p */

DEFINE VARIABLE iLineNum AS INTEGER NO-UNDO.

MESSAGE
  "A transaction" (IF TRANSACTION THEN "IS" ELSE "IS NOT") "active" SKIP
  "before the first REPEAT block" SKIP
  "before any activity takes place" VIEW-AS ALERT-BOX.

Orderblock:
REPEAT:
  MESSAGE "A transaction" (IF TRANSACTION THEN "IS" ELSE "IS NOT")
    "active in the outer REPEAT block," SKIP "before creating the Order"
    VIEW-AS ALERT-BOX.
  CREATE Order.
  DISPLAY OrderNum OrderDate.
  UPDATE CustNum.
  iLineNum = 0.

Orderlineblock:
REPEAT:
  MESSAGE "A transaction" (IF TRANSACTION THEN "IS" ELSE "IS NOT")
    "active in the inner REPEAT block" SKIP
    "after creating the Order, before creating the Orderline"
    VIEW-AS ALERT-BOX.

  CREATE OrderLine.
```

```

ASSIGN OrderLine.OrderNum = Order.OrderNum
      iLineNum           = iLineNum + 1
      OrderLine.LineNum   = iLineNum.

DISPLAY OrderLine.OrderNum FORMAT ">>>>9"
      LineNum             FORMAT ">>>9".

UPDATE ItemNum.

FIND ITEM OF OrderLine.
ASSIGN Orderline.Price = ITEM.Price.
DISPLAY OrderLine.Price FORMAT ">, >>9.99".
UPDATE Qty.
DISPLAY Qty * OrderLine.Price FORMAT ">>, >>9.99" LABEL "Total price".
END. /* End Orderline block */
END. /* End Order block */

```

2. Open and run `eIsActive.p` in Progress Developer Studio. Compare the messages displayed with the points you identified as having active transactions.
3. When prompted, enter the following values, pressing **Enter** after each:
 

Customer number: 5150

Item number: 5

Quantity: 20
4. Press **Ctrl + Break** to exit the procedure.

## Try It 5.1: Identify the transaction scope



As a developer, you need to be aware of where transactions begin and end in a procedure so that you can determine whether the scope of the transaction is suitable to your business requirements.

In this Try It, you are given some procedures and asked to identify the transaction scope in each procedure.

This Try It has two parts and should take approximately 20 minutes to complete.



## Before you begin

Before you begin, you must:

1. Complete *Guided Exercise 5.2: Using the TRANSACTION function to identify scope*.

### Location of solution code:

/progress\_education/openedge/ABLEssentials/src/Solutions/Lesson05

## Part 1 - Identify the transaction scope 1

1. Review the procedure, lTrans1.p:

```
/* lTrans1.p */

DEFINE VARIABLE iLineNum AS INTEGER NO-UNDO.

RepOrdblk:
REPEAT:
    CREATE Order.
    DISPLAY OrderNum.
    UPDATE Order.OrderDate Order.CustNum Order.Carrier.
    FIND Customer OF Order.
    iLineNum = 0.
    RepOrdlineblk:
    REPEAT:
        CREATE OrderLine.
        ASSIGN OrderLine.OrderNum = Order.OrderNum
            iLineNum = iLineNum + 1
            OrderLine.LineNum = iLineNum.
        ASSIGN Qty = 0.
        DISPLAY LineNum.
        UPDATE OrderLine.ItemNum Qty.
    END. /* RepOrdlineblk */
END. /* RepOrdblk */
ForCustblk:
FOR EACH Order OF Customer:
    DISPLAY Order.CustNum Order.OrderNum.
    UPDATE Customer.SalesRep.
END. /* ForCustblk */
```

2. Identify the block or blocks that scope a transaction.
3. Verify your answer with a compile listing.

## Part 2—Identify the transaction scope 2

1. Review the procedure, lTrans2.p:

```
/* lTrans2.p */

Repblk:
REPEAT WITH 1 COLUMN:
    PROMPT-FOR ItemName.
    FIND item USING ItemName NO-ERROR.
    IF NOT AVAILABLE item THEN
        AskQuestion:
        DO:
            MESSAGE "The item" INPUT ItemName "does not exist."
            "Do you wish to add this item?"
            VIEW-AS ALERT-BOX QUESTION BUTTON YES-NO
            UPDATE lAnswer AS LOGICAL.
            IF lAnswer THEN CREATE ITEM.
            ASSIGN ItemName.
            SET Item EXCEPT ItemNum.
```

```
END. /* AskQuestion */  
DISPLAY Item.It
```

2. Identify the block or blocks that scope a transaction.
3. Verify your answer with a compile listing.

### Wrap-up

In this Try It, you tried to identify the scope of transactions in a procedure by studying the code and then verified it by generating the compile listing for the procedure.

## Guided Exercise 5.3 : Save data with default transaction scope



### Introduction

In this exercise, you run the procedure, `eTrans2.p`, enter order data, simulate a crash, and then check to see what data was saved. The procedure, `eTrans2.p`, uses the default transaction scope. In this case, the outermost `REPEAT` block defines the scope of the transaction. The units of work are:

- The modification of an `Order` record
- The modification of all the `OrderLine` records for that `Order` record

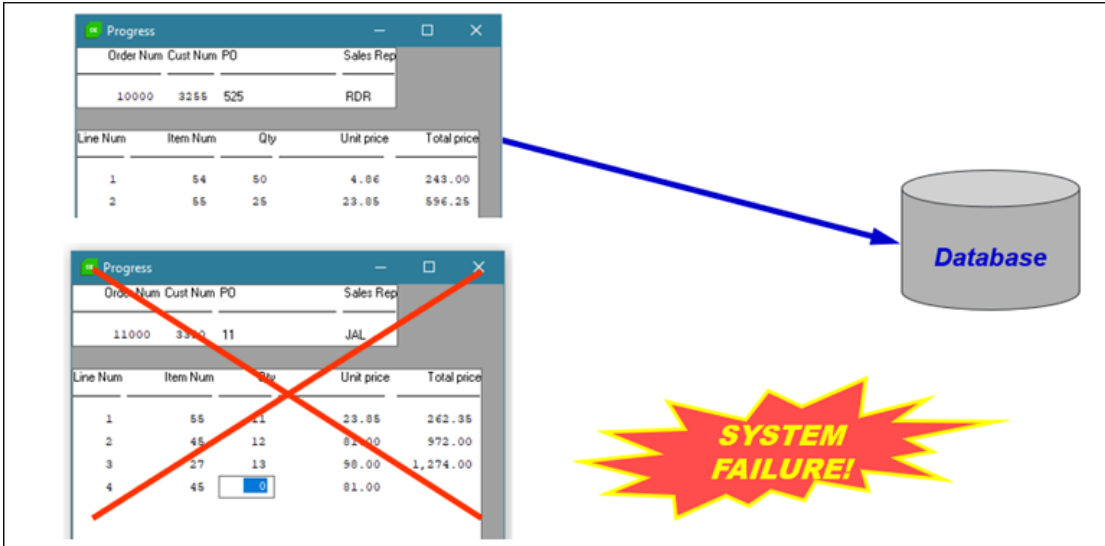
```
/* eTrans2.p */  
  
/* Start of the transaction */  
REPEAT:  
    PROMPT-FOR Order.OrderNum.  
    FIND Order WHERE OrderNum = INPUT Order.OrderNum.  
    DISPLAY OrderNum CustNum PO SalesRep.  
    SET Order.PO.  
    REPEAT:  
        FIND NEXT OrderLine OF order.  
        DISPLAY LineNum ItemNum Qty Price LABEL "Unit price".
```

```
SET Qty.  
DISPLAY Qty * Price LABEL "Total price".  
END. /* FIND NEXT OrderLine block */  
END. /* FIND Order block - End of the transaction */
```

In the preceding example if the system crashes, then:

- At run time, the AVM rolls back any database modifications performed in the current iteration of the outer REPEAT block, the Order record, and all the Orderline records.
- Any previously modified Order is safely stored in the database.

The following image shows that order number 10000 is saved, but order number 11000 is not saved.



**Part 1—Add some Order data to the database and simulate a crash**

Run the procedure, eTrans2.p, and record the data you enter in the following table:

OrderNum	PONum	Qty

1. Enter an order number for each order.
2. Enter data for three orders and their order lines.
3. Save the first two orders.
4. Press **Ctrl+Enter** to stop the procedure before saving the data for the third order.

## Part 2—Test for saved data with default transaction scope

The procedure, `eCheckOrders.p`, lets you retrieve an order record by the order number.

1. Run the procedure `eCheckOrders.p`.
2. Enter the order number from the preceding table.
3. Verify that the data matches what you entered.
4. Repeat steps 2 and 3 for the second and third orders that you entered.

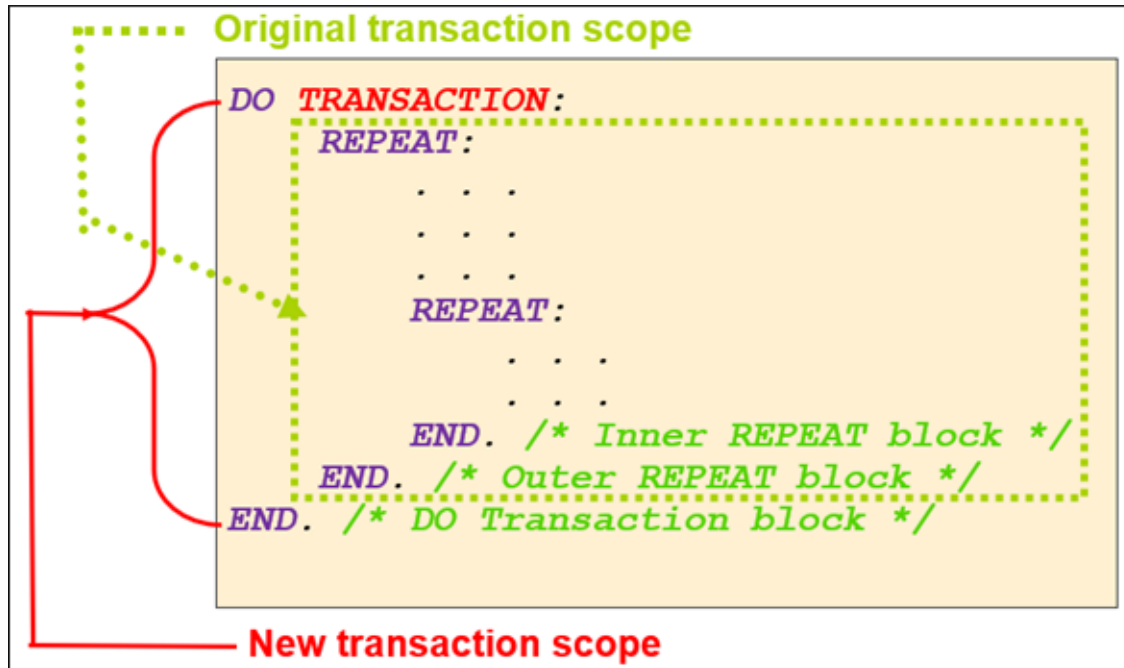
You should see that the first two records were saved, but the third record was not saved

## Guided Exercise 5.4: Save data with increased transaction scope



### Introduction

In this exercise, you run the procedure, `eTrans4.p`, enter order data, simulate a crash, and then check to see what data was saved. In the following procedure, `eTrans4.p`, you increase the scope of the transaction to the whole procedure by placing all the code in the procedure in a `DO TRANSACTION` block.



The unit of work in the DO TRANSACTION block in this code example includes:

- The modification of all orders with the first REPEAT block.
- The modification of all order lines associated with an order in the inner REPEAT block.

```

/* eTrans2.p */

/* Start of the transaction */
REPEAT:
  PROMPT-FOR Order.OrderNum.
  FIND Order WHERE OrderNum = INPUT Order.OrderNum.
  DISPLAY OrderNum CustNum PO SalesRep.
  SET Order.PO.
  REPEAT:
    FIND NEXT OrderLine OF order.
    DISPLAY LineNum ItemNum Qty Price LABEL "Unit price".
    SET Qty.
    DISPLAY Qty * Price LABEL "Total price".
  END. /* FIND NEXT OrderLine block */
END. /* FIND Order block - End of the transaction */

```

### Part 1—Add some Order data to the database and simulate a crash

Run the procedure, eTrans4.p, and record the input data you enter in the following table:

OrderNum	PONum	Qty

OrderNum	PONum	Qty

- Enter an order number for each order.
- Enter data for three orders and their order lines in the table after these instructions.
- Save the first two orders.
- Press `Ctrl+Enter` to stop the procedure before saving the data for the third order.

### Part 2—Test for saved data with increased transaction scope

Run `eCheckOrders.p` to retrieve order records by order number.

1. Run the procedure `eCheckOrders.p`.
2. Enter the order number from the preceding table.
3. Verify that the data matches what you entered.
4. Repeat steps 2 and 3 for the second and third orders that you entered.

You should see that none of the data was saved.

## Guided Exercise 5.5: Save data with reduced transaction scope



### Introduction

In this exercise, you run the procedure, `eTrans5.p`, enter order data, simulate a crash, and then check to see what data was saved. In the procedure, `eTrans5.p`, the larger transaction in the `REPEAT` block is broken into two smaller transactions: creating orders as one transaction and creating order lines as a separate transaction.

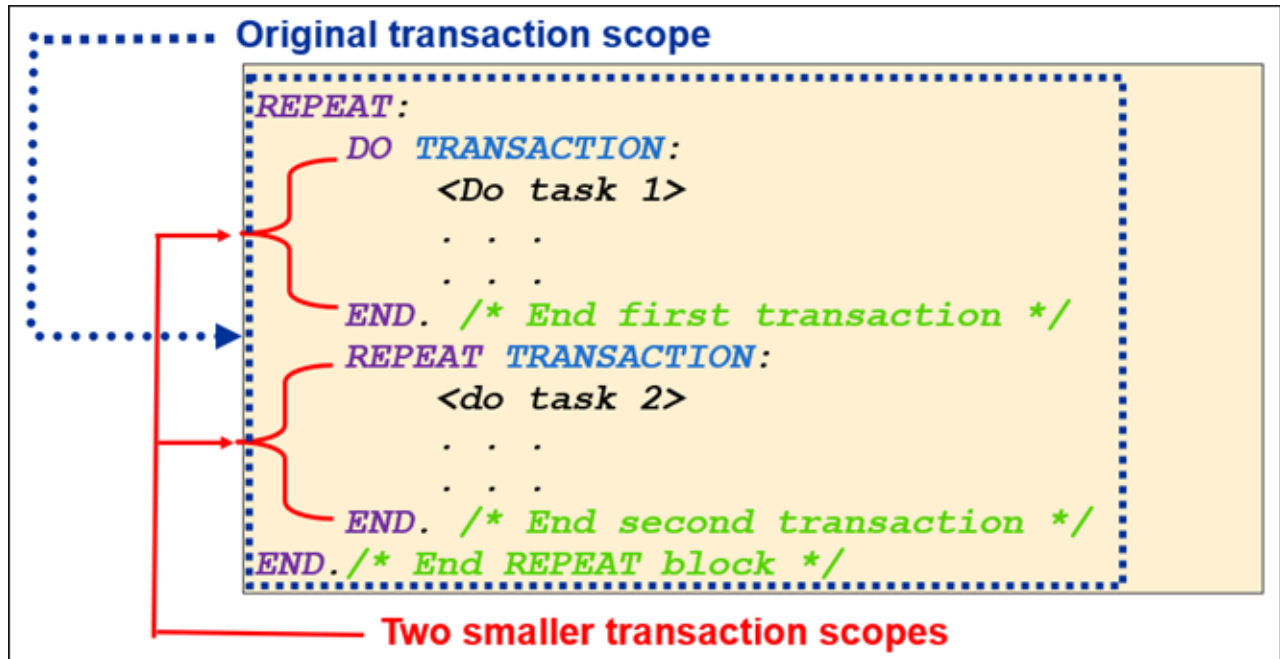
- The `DO TRANSACTION` block transaction scope includes the modification of an order record.
- The inner `REPEAT` block scopes a transaction that includes the update of one `OrderLine` record.

```

/* eTrans5.p */

REPEAT:
    DO TRANSACTION: /*first transaction */
        PROMPT-FOR Order.OrderNum.
        FIND Order WHERE OrderNum = INPUT Order.OrderNum.
        DISPLAY OrderNum CustNum PO SalesRep.
        SET Order.PO.
    END. /* End DO TRANSACTION block (first transaction) */
    REPEAT TRANSACTION: /* second transaction */
        FIND NEXT OrderLine OF order.
        DISPLAY LineNum ItemNum Qty Price LABEL "Unit price".
        SET Qty.
        DISPLAY Qty * Price LABEL "Total price".
    END. /* End REPEAT TRANSACTION block (second transaction) */
END. /* REPEAT block */

```



## Part 1—Add some Order data to the database and simulate a crash

You now repeat the same test on the procedure where the transaction scope is reduced.

Test to see which data is saved and which data is rolled back.

- Enter an order number for each order.
- Enter data for three orders and their order lines.
- Save the first two orders.
- Press **Ctrl+Enter** to stop the procedure before saving the data for the third order.

Run the procedure, `eTrans5.p`, and record the data you enter in the following table:

OrderNum	PONum	Qty

**Part 2—Test for saved data with reduced transaction scope**

Run `eCheckOrders.p` to retrieve order records by order number.

- 1. Run the procedure `eCheckOrders.p`.
- 2. Enter the order number from the preceding table.
- 3. Verify that the data matches what you entered.
- 4. Repeat steps 2 and 3 for the second and third orders.

You should see that every record is saved except the last order line for the third order.

**Guided Exercise 5.6: Running a sub-procedure within a transaction**





## Introduction

If you start a transaction in a main procedure, then that transaction remains active even while the main procedure runs a subprocedure. Any database modifications in the subprocedure are part of the transaction started in the main procedure.

**Note:** If a transaction spans across a procedure, then ensure that the execution path is well defined and that the transaction is kept as short as possible.

In this exercise, you run the procedure, `eRunTrans.p` that starts a transaction and calls the subprocedure that carries out a subtransaction. You enter order data, simulate a crash, and then check to see what data was saved.

The procedure, `eRunTrans.p`, calls the procedure, `eUpdOrder.p` within the scope of an active transaction.

**Note:** `(BUFFER Customer)` is a parameter passed from the parent procedure to the subprocedure. Parameter passing is covered later in this course.

## Procedure `eRunTrans.p`

```

1  /* eRunTrans.p */
2  REPEAT:
3    PROMPT-FOR Customer.CustNum.
4    FIND Customer USING CustNum.
5    DISPLAY Name FORMAT "x(20)" SalesRep CreditLimit FORMAT ">>>>9".
6    SET CreditLimit.
7    MESSAGE "Do you want to do order processing?" VIEW-AS ALERT-BOX
8      QUESTION BUTTONS YES-NO UPDATE lAnswer AS LOGICAL.
9    IF lAnswer THEN DO:
10     RUN eUpdOrder.p(BUFFER Customer).
11   END.
12 END.
13

```

## Procedure `eUpdOrder.p`

```

1  /* eUpdOrder.p */
2  DEFINE PARAMETER BUFFER Customer FOR Customer.
3  DEFINE VARIABLE iItem AS INTEGER NO-UNDO.
4  FOR EACH Order OF Customer:
5    DISPLAY Order.CustNum Order.Ordernum Carrier.
6    SET Carrier.
7    FOR EACH OrderLine OF Order:
8      ASSIGN OrderLine.ItemNum = 0 Qty = 0.
9      DISPLAY LineNum.
10     SET OrderLine.ItemNum Qty.
11     FIND ITEM WHERE OrderLine.ItemNum = ITEM.ItemNum.
12     ASSIGN Orderline.Price = ITEM.Price.
13     DISPLAY Item.Price LABEL "Unit price"
14       Qty * ITEM.Price LABEL "Total price".
15   END.
16 END.

```

When `eUpdOrder.p` is invoked, a transaction is already active in `eRunTrans.p`. The work done in the subprocedure is part of this transaction. If a system error occurs while processing orders, all the order processing work done for the customer and any changes made to the customer record are undone.

### Part 1—Start the main procedure and enter some data

Run the procedure, `eRunTrans.p`, and record the input data in the following table:

CustNum	Credit Limit	PONum	Qty

1. Enter a customer number.
2. Enter a credit limit amount.
3. Select **Yes** to do order processing. This invokes the subprocedure `eUpdOrder.p`.
4. Enter a **PO** number.
5. Enter a **quantity**.
6. Repeat step 5 twice, adding two items.
7. Repeat steps 4 and 5 for a second order for the same customer.
8. Use the **END-ERROR** key to undo the last order line and to exit the subprocedure (`eUpdOrder.p`).
9. Run `eCheckOrderUpdates.p` to verify which orders and order lines were saved.  
You should see every order line except the last saved.

### Part 2—Enter some more data and stop the subprocedure

1. Repeat steps 1 through 7 in the previous section with different data.
2. This time press **Ctrl+Break** to stop the procedure before saving the data for the third order.
3. Run the procedure, `eCheckOrderUpdates.p`.

This time you should see that none of your data was saved because the entire transaction was rolled back.

## Try It 5.2: Modify the transaction scope



Based on business or technical requirements you can increase or decrease the scope of a transaction. In this Try It, you will decrease the transaction scope to increase the application's performance. This Try It has one part and should take 10 minutes to complete.

### Before you begin

Before you begin, you must:

1. Know how to identify transaction scope.
2. Complete Try It 5.1.

#### Location of solution code:

`/progress_education/openedge/ABLEssentials/src/Solutions/Lesson05`

### Modify the transaction scope

1. Reduce the scope of the transaction in `lTrans2.p` (which you used in Try It 5.1).
2. Save your changes as `tTrans3.p`.
3. Verify your answer with a compile listing.

### Wrap-up

In this Try It, you reduced the scope of a transaction in the procedure `lTrans2.p` and then verified the new transaction scope by generating a compile listing.

## Try It 6.1: Identify the record scope and transaction duration



In this Try It, you will understand that there is more to programming than accessing the database and displaying records. As a developer, you must consider the behavior of your code as well. Use the following steps to determine the duration of a record lock:

1. Identify the transaction scope in your procedure.
2. Identify the record scope in your procedure.
3. Assess the duration of the record locks in the procedure.

Remember that the scope of a record and the scope of the transaction together determine the duration of a record lock.

This Try It has three parts and should take approximately 30 minutes to complete.

### Before you begin

Before you begin, you must:

1. Know how to simulate a multi-user environment, see *Procedure 6.1: Simulate a multi-user session*.
2. Create a compile listing, see *Guided Exercise 5.1: Generate a compile listing*.
3. Use the compile listing to identify a record scope, see *Procedure 6.3: Use a COMPILE listing to identify the record scope*.

#### Location of solution code:

/progress\_education/openedge/ABLEssentials/src/Solutions/Lesson06

### Part 1—Identify the transaction scope

To identify the transaction scope in the procedure, `lLock.p`.

1. Open procedure, `lLock.p`.
2. Review the code.
3. Identify the transaction scope and note it.

## Part 2—Identify the scope of each record buffer

1. Identify the scope of each record buffer.
2. Identify the record scope of each record buffer used in the procedure and update the following table.

This record buffer...	Is scoped to this block...

## Part 3—Identify the type and duration of the lock on each record

In the following table, record the lock type and duration of each lock.

**Note:** The customer record has two separate locks.

Record	Lock type	Block in which the lock is acquired	Upgraded or released in
Customer			
Order			
OrderLine			
Item			

### Questions

Consider these questions and note your answers:

1. How does the locking strategy of this procedure affect other users' access to the database?
2. How does the locking strategy of this procedure affect the execution of this procedure?

### Wrap-up

In this Try It, you identified the transaction scope, the record scope, and the type and duration of lock on each record.

## Try It 6.2: Implement optimistic locking



In this Try It, you will modify the procedure, `lChangeDept1.p`, to use the optimistic locking strategy. You will simulate a multi-user environment and observe its behavior in different scenarios. The scenarios include checking and not checking for the availability of the locked record and observing the results after each run.

This Try It has three parts and should take approximately 60 minutes to complete.

### Before you begin

Before you begin, you must:

1. Know how to simulate a multi-user environment, see *Procedure 6.1*.
2. Complete *Try It 6.1*.

#### Location of solution code:

`/progress_education/openedge/ABLEssentials/src/Solutions/Lesson06`

### Part 1—Run a simple procedure in a multi-user environment

1. In Progress Developer Studio, compile `lChangeDept1.p`.
2. Start `lChangeDept1.p` twice so it runs in two sessions.
3. This should display a dialog box for each session.
4. Type the same department number in both sessions and press **Enter**.
5. The department details should display in the dialog boxes.
6. Modify the department name in one of the dialog boxes.
7. Press **Enter**.

**Question:** What is the result?

8. Click **Cancel** to exit the **Database Status** dialog. Then exit both sessions.

## Part 2—Simple optimistic locking without checking availability

In this part, you will write code to employ simple optimistic locking without checking if the record is available after fetching the record again. Copy the code from `lChangeDept1.p` into a new procedure called `tChangeDept2.p`. Modify the code to perform the following:

1. Retrieve the department record using the `NO-LOCK` option and allow the user to make changes in the screen buffer.
2. Fetch the same department record again with `EXCLUSIVE-LOCK`.
3. Verify that the record did not change (`CURRENT-CHANGED`). If the record did not change, then commit the changes to the database. If the record changed, then display a message to the user and allow the user to apply the changes again.

### Test 1

Open two window sessions in Progress Developer Studio. Perform the following test:

First Session	Second Session
Open and run <code>tChangeDept2.p</code> .	
	Open and run <code>tChangeDept2.p</code> .
Type a department number. (100, 200, ..., 700) and hit <b>Enter</b> . The department name is displayed.	
Modify the department name but do not hit <b>Enter</b> .	
	Type the same department number used in Session 1 and press <b>Enter</b> . The department name is displayed..
	Modify the department name and hit <b>Enter</b> . The department name is changed and you are prompted with a new row.
Hit <b>Enter</b> to submit the modified department name.	

**Question:** What is the result of this test?

Click **OK** in the message box and close both the sessions.

### Test 2

Now you will test what happens when optimistic locking rules are not followed by everyone. You will run `lChangeDept1.p`, which does not use optimistic locking and `tChangeDept2.p` at the same time. Try to anticipate what will happen before you do the following steps:

First Session	Second Session
Open and run <code>lChangeDept1.p</code> .	
	Run <code>tChangeDept2.p</code> .

First Session	Second Session
Type a department number. (100, 200, ..., 700) and hit <b>Enter</b> . The department name is displayed.	
Modify the department name but do not hit <b>Enter</b> .	
	Type the same department number used in Session 1 and press <b>Enter</b> . The department name is displayed..
	Modify the department name and hit <b>Enter</b> .

**Question:** What is the result of this test?

Click **Cancel** in the **Database Status** dialog and close both the sessions.

### Part 3 – Complete optimistic locking including check for availability

In this part, you will write code to employ complete optimistic locking, and include checking to see if the record is available after fetching the same record again. Copy the code from `tChangeDept2.p` into a new procedure called `tChangeDept3.p`. Modify the code to perform the following:

1. Add `NO-ERROR` to all statements that retrieve records from the database.
2. Test to see if the record is available.
3. Test to see if the record is locked by another user.
4. Include messages, where appropriate.
5. Close all active sessions, if any remain open and then perform the following tests:

#### Test 1

Session 1	Session 2
Open and run <code>tChangeDept3.p</code> .	
Type a department number (100, 200, ..., 700) and hit <b>Enter</b> . The department name is displayed.	
Modify the department name but do not hit <b>Enter</b> .	
	Open and run <code>lDeleteDept.p</code> .
	Type the same department number used in Session 1 and press <b>Enter</b> .
	Click <b>OK</b> in the confirmation dialog to delete the department.
Press <b>Enter</b> after the department name in the first session, to submit your changes.	



**Question:** What is the result of the this test?

Click **OK** in the message dialog and close both the sessions.

### Test 2

Session 1	Session 2
Open and run <code>tChangeDept3.p</code> .	
	Open and run <code>tChangeDept3.p</code> .
	Type a department number (100, 200, ..., 700) and hit <b>Enter</b> . The department name is displayed.
	Modify the department name but do not hit <b>Enter</b> .
Type the same department number used in Session 1 and press <b>Enter</b> .	
Modify the department name and hit <b>Enter</b> .	
	Hit <b>Enter</b> to save the modified name.

**Question:** What is the result of the preceding test?

### Wrap-up

In this Try It, you implemented the optimistic locking strategy and observed how it works in a multi-user session. You also observed how it works when you check and do not check for the availability of the locked record.

## Try It 7.1: Overriding the default error handling



In this Try It, you will override the default error handling when the FIND statement fails to find the specified record.

This Try It should take approximately 10 minutes to complete.

## Before you begin

Before you begin, you must have a working development environment. If you do not have it, then perform the following steps:

1. Complete *Guided Exercise 1.1: Setting up your development environment*.

### Location of solution code:

/progress\_education/openedge/ABLEssentials/src/Solutions/Lesson07

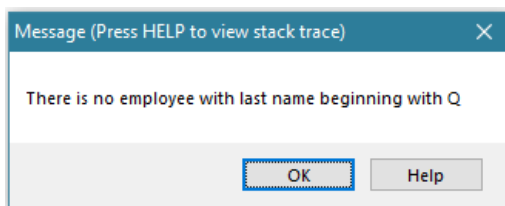
## Part 1—Run the procedure with default error handling

Complete the following steps:

1. Open the procedure `lFindEmp.p`. This procedure asks the user to input the beginning letters of an employee's last name. It then displays fields from the employee's record.
2. Run the procedure.
3. Enter the name 'Smith'. What happens?
4. Enter "We". What happens?
5. Enter the letter "Q". What happens?

## Part 2—Override the default error handling

1. Override default error handling for the FIND statement. If the record is not available, display a message to that effect. It should look something like this:



2. Test your procedure by repeating steps 3-5 in *Part 1—Run the procedure with default error handling*.
3. Save your procedure as `tFindEmp2.p`.

## Wrap-up

In this Try It, you overrode the default error handling and provided your own error message, when the FIND statement fails to find the specified record.

## Try It 7.2: Checking for errors



In this Try It, you rewrite the delete trigger to return a character string instead of using the message statement in the trigger. Replace the existing trigger with your new trigger and write a short procedure to test your trigger.

This Try It should take approximately 30 minutes to complete.

### Before you begin

Before you begin, you must complete *Try It 7.1*.

#### Location of solution code:

/progress\_education/openedge/ABLEssentials/src/Solutions/Lesson07

### Part 1—Modify the DELETE trigger

In this Try It, you rewrite the DELETE trigger for a customer, using the RETURN ERROR statement. Follow these steps to modify the trigger:

1. Open `lDelCust.p`. This is a modified copy of the DELETE trigger. Copy the code into a new procedure called `tDelCust2.p`.
2. Comment all the MESSAGE statements.
3. Modify the RETURN ERROR statements to include the text of the messages you commented out.

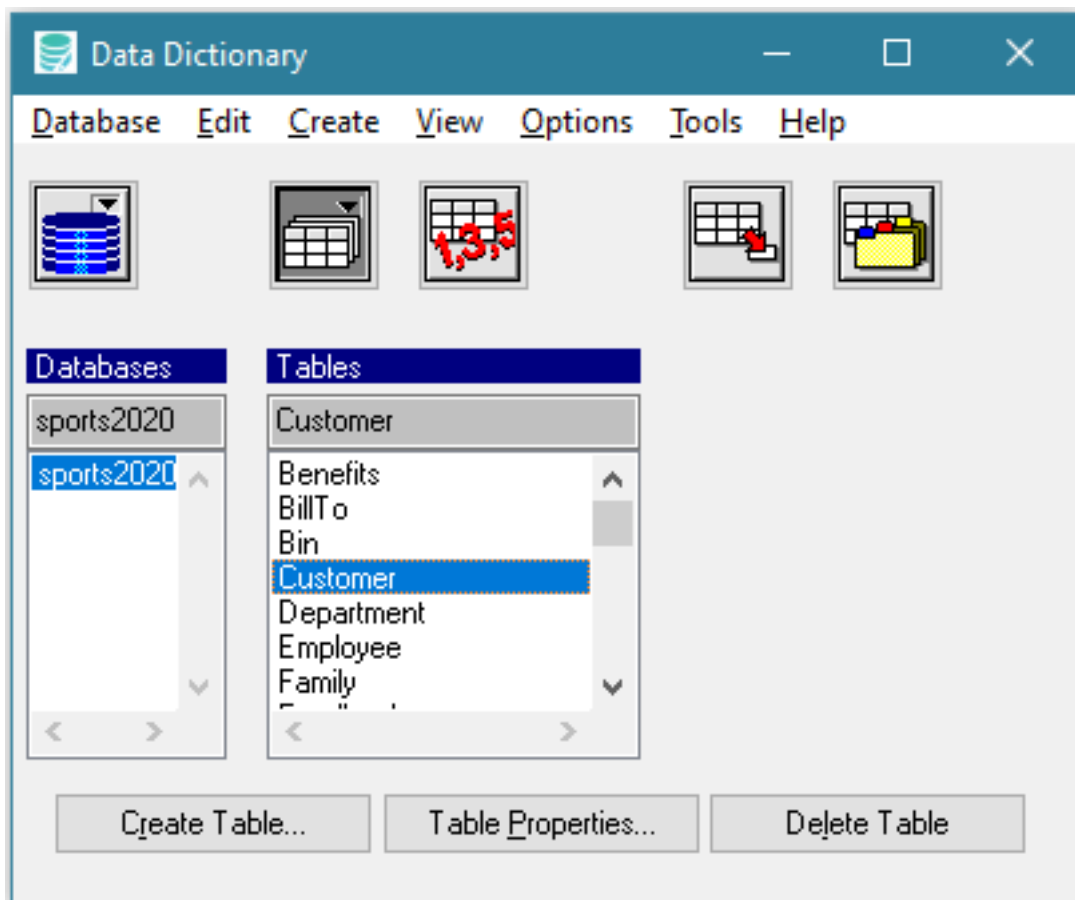
---

**Note:** The `tDelCust2.p` file is a trigger and will not run independently.

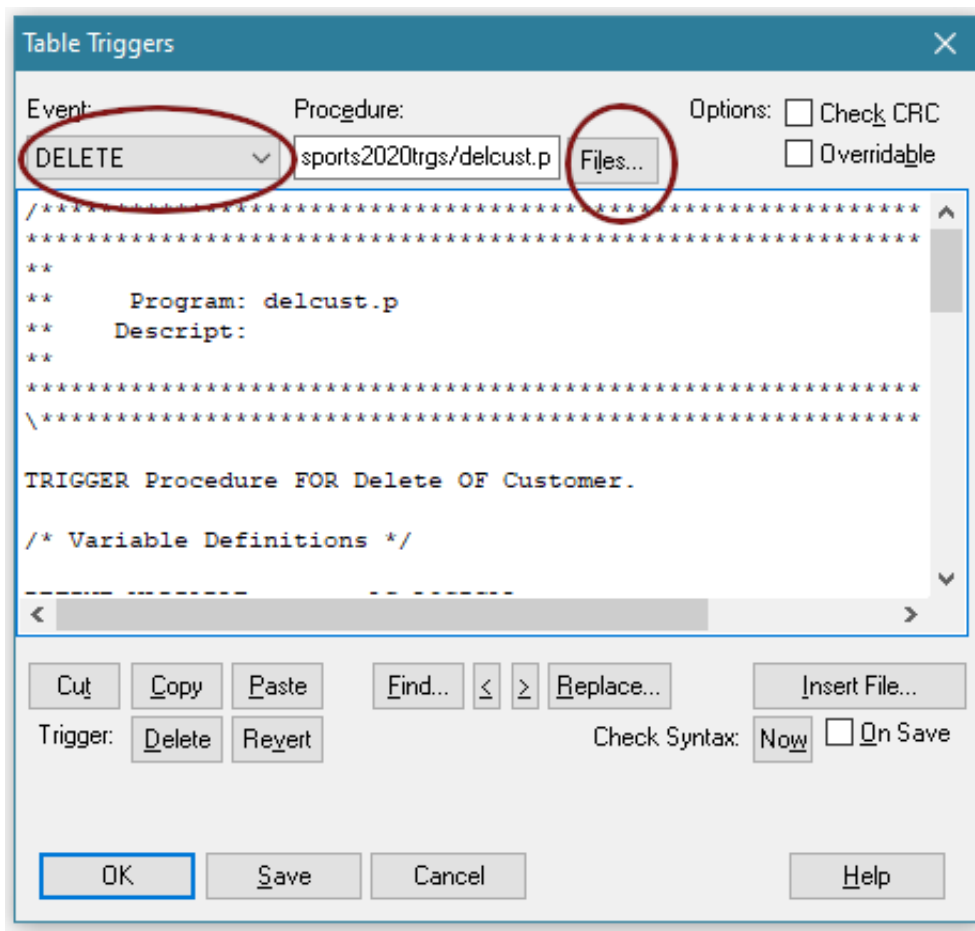
---

### Part 2—Replace the DELETE trigger in the Data Dictionary

1. In Progress Developer Studio, select **OpenEdge > Admin > Data Dictionary**. The **Data Dictionary** window is displayed.



2. Click tables (📊). The Tables list and table related buttons are displayed.
3. Select `Customer` in the table list.
4. Click **Table Properties**. The **Table Properties** dialog is displayed.
5. In the **Table Properties** window, click **Triggers....** The **Table Triggers** dialog is displayed.



6. Select **DELETE** from the **Event** drop-down list. The trigger procedure associated with the delete event is displayed.
7. Click **Files ....** The **Find Trigger Procedure** dialog is displayed.
8. Browse and select `tDelCust2.p` and click **Open**.
9. Click **Save**. A "Trigger Saved" message is displayed above the **OK** button.
10. Click **OK** to exit the **Table Triggers** dialog.
11. Click **OK** to exit the **Table Properties** dialog.
12. Close the **Data Dictionary** and click **Yes** in the **Warning** dialog to commit your changes.

### Part 3—Test the new DELETE trigger

1. Write a procedure to test your trigger. It should do the following:
  - a. Prompt the user for a customer number.
  - b. Delete the customer using the `NO-ERROR` option.
  - c. Test the `ERROR-STATUS:ERROR` handle to see if an error occurred.
  - d. `DISPLAY` the error string returned from the trigger.
2. Save your procedure as `tTestTrigger.p`.

3. Test your procedure with different customer numbers. You should get appropriate error messages. For example, try customer number 3030.
4. You can also do the following:
  - a. Create a new customer.

---

**Note:** You can use `eCreateCustomer.p` to create a new customer and note the new customer number.

---

- b. Delete the customer record using `tTestTrigger.p`.
  - c. You should not see an error message as the new record does not have any child records.

### Wrap-up

In this Try It, you rewrote the delete trigger, replaced the existing trigger with your new trigger and wrote a short procedure to test your delete trigger.

## Try It 7.3: Using ON-ERROR



In this Try, you will use an `ON ERROR` phrase to continue processing when an error occurs. This Try It should take approximately 20 minutes to complete.

### Before you begin

Before you begin, you must complete *Try It 7.2*.

#### Location of solution code:

`/progress_education/openedge/ABLEssentials/src/Solutions/Lesson07`

### Overview

Customers are assigned sales representatives who are listed in the `SalesRep` table. An error will be generated if a customer is assigned a sales rep who is not listed in the `SalesRep` table.

The procedure `lDispCustRep.p` displays Customer number and name, the name of the `Salesrep` associated with the `Customer`, and the first invoice, if any, of the `Customer`.

In this Try It, your task is to write a procedure that uses the `ON ERROR` phrase to leave the block if a customer does not have a salesrep. Your procedure should then continue on to display the invoice amount for the customer without a salesrep and the details for other customers.

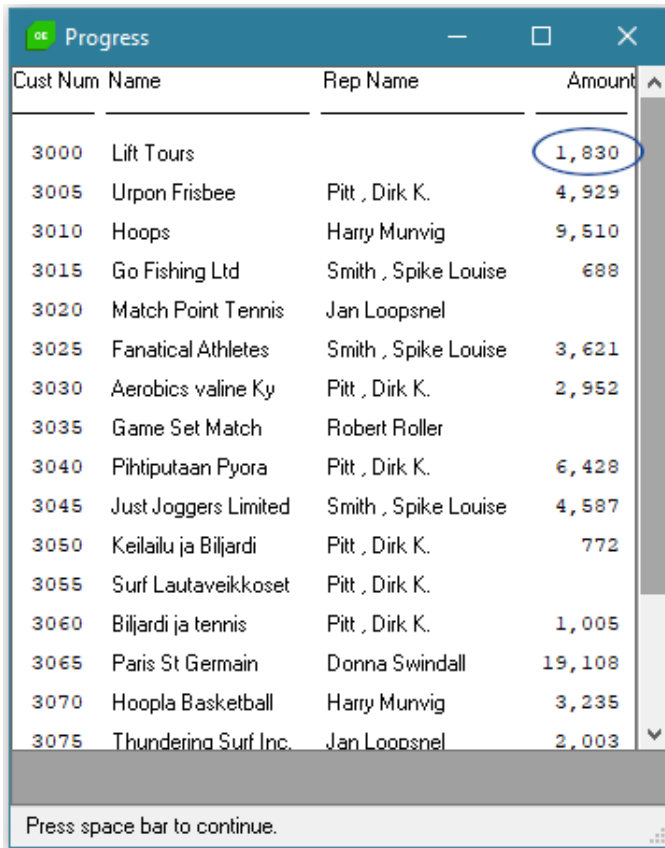
### Part 1—Simulate an error and observe the default error handling

1. Run `lDispCustRep.p` to see how it behaves normally.
2. The Sports2020 database does not have any inconsistencies that would cause an error so you must create one. Run `lMakeError.p`, which assigns an invalid `Salesrep`, which then generates an error.
3. Run `lDispCustRep.p` again to see how it behaves with default error handling. Notice that no invoice amount displays for **Lift Tours**.

### Part 2—Customize the error handling, save, and test

Create a procedure called `tDispCustRep.p` with the following changes to the code in `lDispCustRep.p`.

1. Add a `DO ON ERROR UNDO, LEAVE` statement around the `FIND` and `DISPLAY Salesrep` statement.
2. Save your procedure.
3. Run the procedure again. You should see that the invoice amount is displayed for **Lift Tours**.

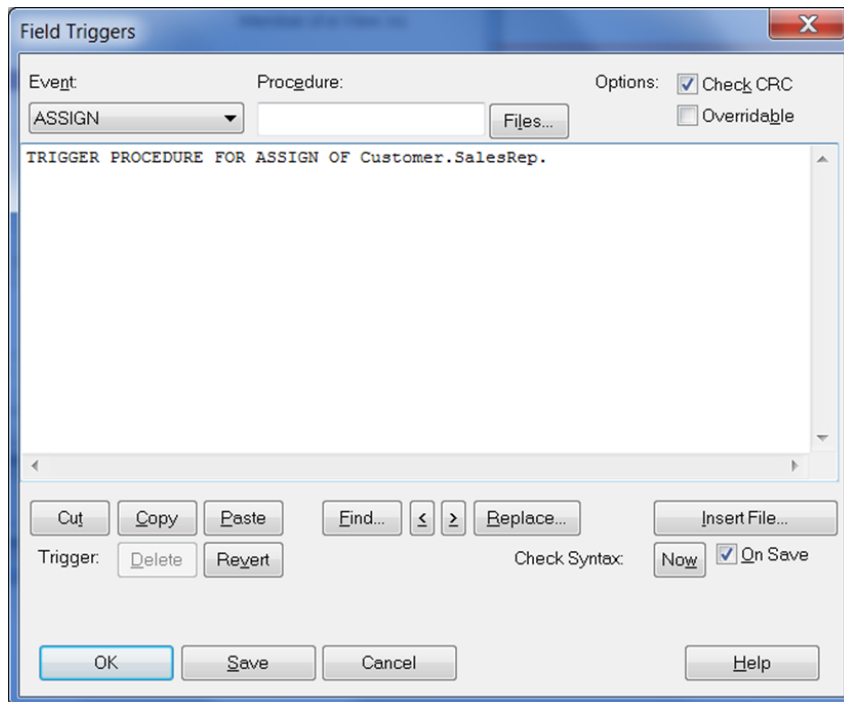


Cust Num	Name	Rep Name	Amount
3000	Lift Tours		1,830
3005	Urpon Frisbee	Pitt , Dirk K.	4,929
3010	Hoops	Harry Munvig	9,510
3015	Go Fishing Ltd	Smith , Spike Louise	688
3020	Match Point Tennis	Jan Loopsnel	
3025	Fanatical Athletes	Smith , Spike Louise	3,621
3030	Aerobics valine Ky	Pitt , Dirk K.	2,952
3035	Game Set Match	Robert Roller	
3040	Pihtiutaan Pyora	Pitt , Dirk K.	6,428
3045	Just Joggers Limited	Smith , Spike Louise	4,587
3050	Keilailu ja Biljardi	Pitt , Dirk K.	772
3055	Surf Lautaveikkoset	Pitt , Dirk K.	
3060	Biljardi ja tennis	Pitt , Dirk K.	1,005
3065	Paris St Germain	Donna Swindall	19,108
3070	Hoopla Basketball	Harry Munvig	3,235
3075	Thundering Surf Inc.	Jan Loopsnel	2,003

Press space bar to continue.

### Part 3—Bonus - Code a trigger procedure on ASSIGN

Write a trigger procedure for adding a new salesrep.




---

**Note:** This is a bonus practice exercise. Solution is not provided.

---

## Wrap-up

In this Try It, you used an ON ERROR phrase to continue processing when an error occurred.

## Try It 8.1: Parameters and functions



In this Try It, you will:

- Pass parameters to a procedure.



- Define parameters for a procedure.
- Write and invoke a user-defined function.

This Try It should take approximately 10 minutes to complete.

## Before you begin

Before you begin, you should:

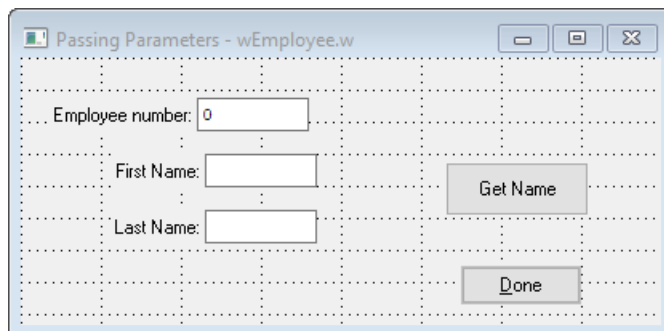
1. Complete *Guided Exercise 1.1: Setting up your development environment*.
2. Know how to create UI procedure in AppBuilder perspective, see *Procedure 8.1: Create an ABL UI procedure in the AppBuilder perspective*.
3. Know how to add trigger code to a widget, see *Procedure 8.2: Add trigger code to a widget*.
4. Know how to edit the trigger code, see *Procedure 8.3: Edit the triggers in a widget*.
5. Know how to define a function, see *Procedure 8.4: Define a function*.

### Location of solution code:

/progress\_education/openedge/ABLEssentials/src/Solutions/Lesson08

## Part 1—Pass parameters

The following user interface was created for you. Add code to this interface to retrieve an employee's name when you enter the employee's number.

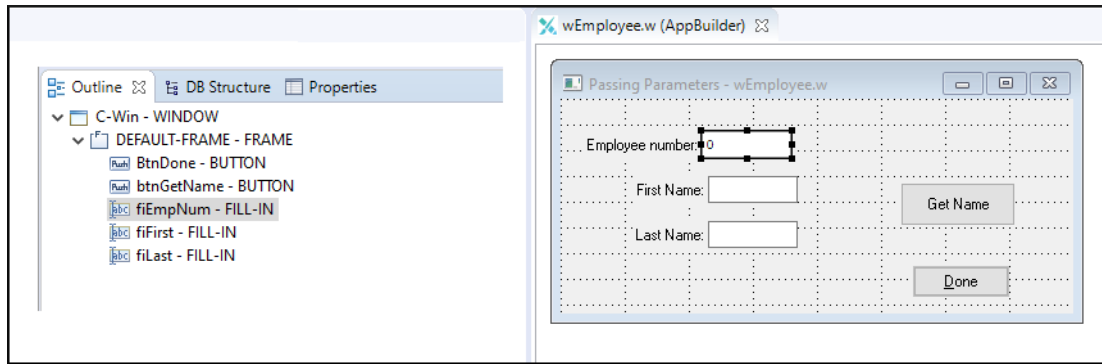


Perform the following steps:

1. Set the PROPATH to point to the directories: src\Try Its\User Interface and src\Try Its\Business Logic.
2. Open the procedure, src\Try Its\Business Logic\lGetEmpName.p. What type of parameters does it expect?
3. Open src\Try Its\User Interface\wEmployee.w.

The three text fields on the screen have labels, and you need their object names to manipulate them.

- a. Select **Window > Show View > Outline**. The **Outline** view opens in the lower left panel.
- b. Expand the nodes **C-Win - WINDOW > DEFAULT FRAME - FRAME**.
- c. Click a widget node to see it highlighted in the **UI Design** view.



- d. Note the object name associated with a UI element.. The nodes in the **Outline** view are listed in the format *Object name – Widget type*. In the preceding figure, **Employee number** is listed as **fiEmpNum – FILL IN**, and the object name for the text field is **fiEmpNum**.
- e. Enter the object names for the widgets in the following table:

Label	Object Name
Employee number	fiEmpNum
First name	
Last name	
Get name	

4. Write the trigger code that fetches the employee names. This code executes when the user clicks the **Get Name** button.
- Right click on the design view and select **View source** or press **F9** on the keyboard. The source for `wEmployee.w` procedure is displayed in the code editor.
  - Add a trigger for the **Get Name** button. Right click on the source view and select **Source > Add Trigger**. The **Add Trigger** dialog is displayed.
  - Select `btnGetName [BUTTON] DEFAULT-FRAME` for the **Widget** field, `CommonEvents` for the **Event Category** field and `CHOOSE` for the **Event** field from the drop-down lists.
  - Click **Generate**. The trigger template code block for the `btnGetName` widget is inserted.

```

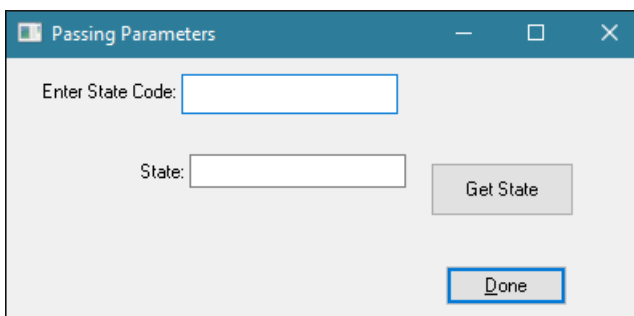
201     RUN dispatch IN THIS-PROCEDURE ('exit').
202     &ELSE
203     RUN exitObject.
204     &ENDIF
205     &ELSE
206     APPLY "CLOSE":U TO THIS-PROCEDURE.
207     &ENDIF
208 END.
209
210 /* _UIB-CODE-BLOCK-END */
211 &ANALYZE-RESUME
212
213
214 &Scoped-define SELF-NAME btnGetName
215 &ANALYZE-SUSPEND _UIB-CODE-BLOCK _CONTROL btnGetName C-Win
216 ON CHOOSE OF btnGetName IN FRAME DEFAULT-FRAME /* Get Name */
217 DO:
218
219 END.
220
221 /* _UIB-CODE-BLOCK-END */
222 &ANALYZE-RESUME
223
224
225

```

- e. Write a RUN statement that executes the lGetEmpName.p procedure, passing the appropriate parameters. The procedure should display the employee's first and last name in the **First Name** and **Last Name** fields.
5. Save the procedure wEmployee.w then click **Run**.
  6. Enter an employee number and test the code. Try employee numbers from 1 to 56 and try other numbers not in the database.

## Part 2—Define parameters

1. Open src\Try Its\User Interface\wState.w. The **Get State** button executes a routine to get the name of a state, based on the two-character state code. The procedure that validates and fetches the state name based on its code is valState.p.

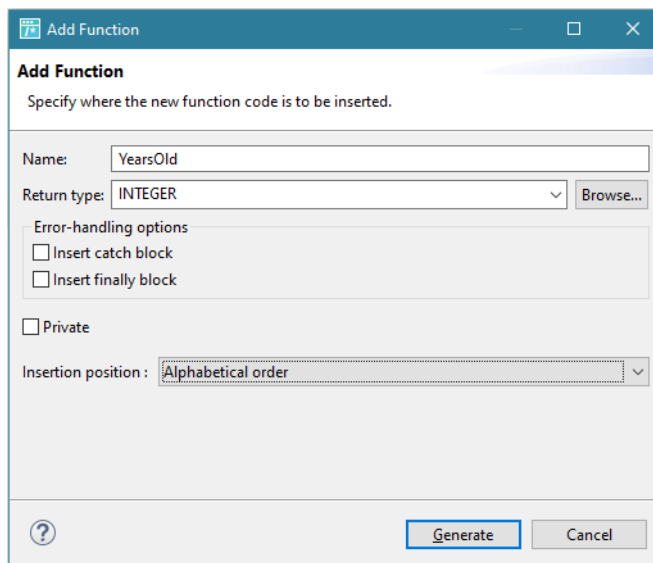


2. Add the trigger template code for the **Get State** button.
  - a. Right click on the design view and select **View source** or press **F9** on the keyboard. The source for wState.w procedure is displayed in the code editor.
  - b. Right click on the source view and select **Source > Add Trigger**. The **Add Trigger** dialog is displayed.

- c. Select `btnGetState[BUTTON] DEFAULT-FRAME` for the **Widget** field, `CommonEvents` for the **Event Category** field and `CHOOSE` for the **Event** field from the drop-down lists.
  - d. Click **Generate**. The trigger template code block for the `btnGetState` widget is inserted.
3. Write the trigger code for the **Get State** button that does the following:
  - a. Assigns the value of the **State Code** field.
  - b. Runs the `valState.p` procedure with the correct parameters to send and receive data.
  - c. Displays the returned value of the state name in the appropriate field.
4. Open `src\Try Its\User Interface\valState.p`. This procedure is incomplete and needs to be completed. Do the following:
  - a. Define the parameters this code needs to take the state code and return the state name.
  - b. Uncomment the code that assigns the state name.
  - c. Save `valState.p`.
5. Run `wState.w` and test your code. Here are some state codes to test: MA, TX, CO. Try some non-existent codes such as QW.

### Part 3—User-defined functions

1. Open `src\Try Its\User Interface\wEmpInfo.w`.
2. Right click on the design view and select **View source** or press **F9** on the keyboard. The source for `wEmpInfo.w` procedure is displayed in the code editor.
3. Create a new function called **YearsOld**. The function returns an integer value that is the employee's age.
  - a. Right-click on the source view and select **Source > Add Function**. The **Add Function** dialog is displayed.
  - b. Enter `YearsOld` in the **Name** field, select `INTEGER` as the **Return type** and **Alphabetical Order** for the **Insertion Position**.



- c. Click **Generate**. The template code for the `YearsOld` function is added to the source view of `wEmpInfo.w`.

```

385 ASSIGN
386     iResult = TRUNCATE((TODAY - Employee.StartDate) / 365,0).
387 RETURN iResult. /* Function return value. */
388
389 END FUNCTION.
390
391 /* _UIB-CODE-BLOCK-END */
392 &ANALYZE-RESUME
393
394 &ANALYZE-SUSPEND _UIB-CODE-BLOCK _FUNCTION YearsOld C-Win
395 FUNCTION YearsOld RETURNS INTEGER
396     ( ):
397     /*-----
398     Purpose:
399     Notes:
400     -----
401     DEFINE VARIABLE result AS INTEGER NO-UNDO.
402
403     RETURN result.
404
405 END FUNCTION.
406
407 /* _UIB-CODE-BLOCK-END */
408 &ANALYZE-RESUME
409

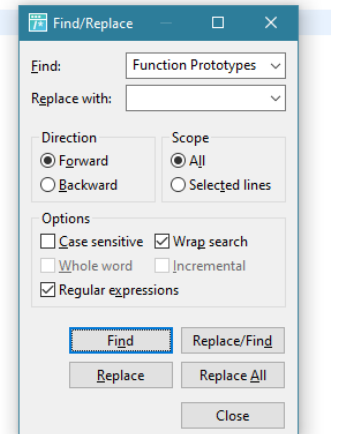
```

- d. Close wEmpInfo.w.
4. To update the function prototype definition, right-click on src\Try Its\User Interface\wEmpInfo.w in the **Project Explorer** and select **Open with Text Editor**. Perform the following steps to modify the function prototype definition for the YearsOld function.
    - a. Press **Ctrl + F** to open the **Find/Replace** dialog. Enter "Function Prototypes" in the **Find** field. Click **Find**.

```

82
83 /* ***** Function Prototypes ***** */
84
85 &ANALYZE-SUSPEND _UIB-CODE-BLOCK _FUNCTION-FORWARD YearsEmployed C-Win
86 FUNCTION YearsEmployed RETURNS INTEGER
87 ( INPUT piEmpNum AS INTEGER ) FORWARD.
88
89 /* _UIB-CODE-BLOCK-END */
90 &ANALYZE-RESUME
91
92 &ANALYZE-SUSPEND _UIB-CODE-BLOCK _FUNCTION-FORWARD YearsOld C-Win
93 FUNCTION YearsOld RETURNS INTEGER
94 ( INPUT iEmpNum AS INTEGER ) FORWARD.
95
96 /* _UIB-CODE-BLOCK-END */
97 &ANALYZE-RESUME
98
99
100
101 /* ***** Control Definitions ***** */
102

```



- b. Update the `YearsOld` function prototype definition with the following code. See the image in the previous step.

```

FUNCTION YearsOld RETURNS INTEGER
( INPUT iEmpNum AS INTEGER ) FORWARD.

```

- c. Save the changes and close `wEmpInfo.w` in the text editor.

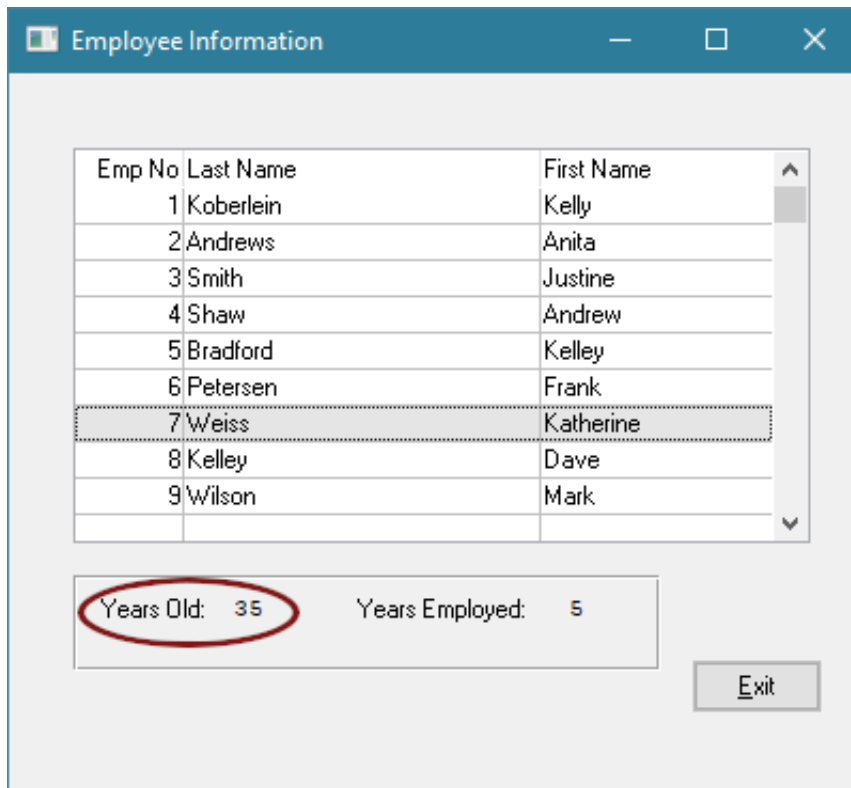
5. Re-open `wEmpInfo.w` in the OpenEdge ABL editor. Add code to the function template to calculate an employee's age. The function takes an employee number as input and returns the employee's age.

---

**Note:** Subtract the employee's birth date from today's date to get the age.

---

6. Find the source for the **ON VALUE- CHANGED** trigger for the **brEmployee** browse object. Add code to call the **YearsOld** function and display the value returned by the function.
7. Run `wEmpInfo.w` and test the function. The **Years Old** field should change value each time you select an employee listed in the browse widget.



The window titled "Employee Information" displays a table with employee data. The table has three columns: Emp No, Last Name, and First Name. The data is as follows:

Emp No	Last Name	First Name
1	Koberlein	Kelly
2	Andrews	Anita
3	Smith	Justine
4	Shaw	Andrew
5	Bradford	Kelley
6	Petersen	Frank
7	Weiss	Katherine
8	Kelley	Dave
9	Wilson	Mark

Below the table, there is a summary section with two labels: "Years Old:" and "Years Employed:". The value "35" under "Years Old:" is circled in red. The value "5" is under "Years Employed:". An "Exit" button is located at the bottom right of the window.

### Wrap-up

In this Try It, passed parameters to a procedure, defined parameters for a procedure, wrote and invoked a user-defined function.

## Try It 8.2: Persistent procedures



In this Try It, you will start and access a persistent procedure.

This Try It should take approximately 30 minutes to complete.

## Before you begin

Before you begin, you must:

1. Complete *Try It 8.1*.
2. Know how to work with persistent procedures, see *Procedure 8.5: Work with persistent procedures*.

### Location of solution code:

/progress\_education/openedge/ABLEssentials/src/Solutions/Lesson08

## Part 1—Persistent procedures

Follow these steps to start a persistent procedure:

1. Open `src\Try Its\User Interface\wRegion.w`. It opens in the Design view. Right click on the design view and select **View source** or press **F9** on the keyboard. The source for `wRegion.w` is displayed in the code editor.

---

**Note:** If you have trouble opening the file in the Design view, right-click the file in the **Project Explorer** pane and select **Open With > OpenEdge ABL Editor** to open the source view.

---

2. Browse to the Definitions section in the source view. Define a variable to hold the handle of a persistent procedure here.

---

**Note:** You can also click the **Definitions** node in the outline view.

---

3. Browse to the Main Block code section in the source view. Write a **RUN** statement in the main block to start `lEmpLibrary.p` as a persistent procedure.

---

**Note:** Use the handle you defined in previous step to store the handle of the persistent procedure

---

4. Add trigger template code for the **Get Region** button.
  - a. Right click on the source view and select **Source > Add Trigger**. The **Add Trigger** dialog is displayed.
  - b. Select **BtnGetRegion[BUTTON] DEFAULT-FRAME** for the **Widget** field, **Portable Mouse Events** for the **Event Category** field and **MOUSE-SELECT-CLICK** for the **Event** field from the drop-down lists.
  - c. Click **Generate**. The trigger template code block for the `BtnGetRegion` widget is inserted in the source view.
5. Add code to the template for the **Get Region** button. It should:
  - a. Assign the input value.
  - b. Run the **valRegion** procedure defined in the persistent procedure, `lEmpLibrary.p`.
  - c. Display the state name and region.
6. Save `wRegion.w`.
7. Run and test `wRegion.w`. Enter the state codes MA, TX, and other values. Does the region name display correctly?



## Part 2 - Bonus, Super procedure

1. Open the source view of the procedure, `wRegion.w`. To open in the source view, right-click the procedure in the Project Explorer and select **Open With > OpenEdge ABL Editor**.
2. Add the procedure, `eEmpLibrary.p`, as super procedure of `wRegion.w`.
3. Run `wRegion.w`.

## Wrap-up

In this Try It, you started and accessed a persistent procedure.

# Try It 9.1: Identify the record scope and transaction duration



In this Try It, you will perform the following:

- Run a procedure from a UI trigger.
- Run a procedure that returns a temp-table with customer data.
- Write a procedure to populate a temp-table for items and provide it as a parameter.

This Try It has five parts and should take approximately 30 minutes to complete.

## Before you begin

Before you begin, you must:

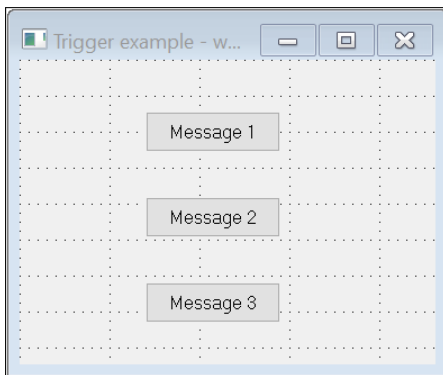
1. Complete *Guided Exercise 1.1: Setting up your development environment*.
2. Know how to create temp-tables, see *Procedure 9.1: Define a temp-table in Progress Developer Studio*.
3. Know how to add trigger code to a widget, see *Procedure 8.2: Add trigger code to a widget*.
4. Know how to edit the trigger code, see *Procedure 8.3: Edit the triggers in a widget*.
5. Know how to define a function, see *Procedure 8.4: Define a function*.

### Location of solution code:

/progress\_education/openedge/ABLEssentials/src/Solutions/Lesson09

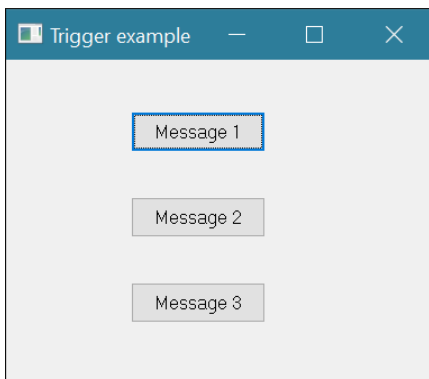
## Part 1—Use a UI trigger

1. Open `wMessageL.w`.
2. Write trigger code for each button:
  - **Message 1** should say "Hello" and your name.
  - **Message 2** should say "Welcome to the last Try It in this course."
  - **Message 3** can be whatever you like.



3. Run and test your trigger code. Right-click the message form and select **Run as > Progress OpenEdge Application**.

The message with the buttons should display.



## Part 2—Run an external Procedure, retrieve, and display the data

In this Try It, you will write code in the user interface file `wCustDisplayL.w`, to call an external procedure **getCustData** that retrieves the customer data. The procedure **GetCustData** is an internal procedure in the library file `src\Examples\Business Logic\DataUtil.p`. You will run `DataUtil.p` as a persistent file and use its handle to call its internal procedure **GetCustData**. When coded correctly, `wCustDisplayL.w` displays customer data in the fill-in fields.

1. Review the exercise files to understand their current state:

- a. Open `Try Its\User Interface\wCustDisplayL.w`. Right-click the design view and select **View Source**.

---

**Note:** If you have trouble opening the procedure in the Design view, right-click the procedure in the **Project Explorer** pane and select **Open With > OpenEdge ABL Editor** to open the source view.

---

- b. Run the procedure. Notice that no data is displayed in the fields. Click the **Next** and **Previous** buttons and observe that nothing happens. Click **Done** to close the form.
  - c. The **GetCustData** internal procedure in `src\Try Its\Business Logic\DataUtil.p` is the procedure that gets the data for the Customer Information form. The procedure is not complete and therefore no data is retrieved and displayed by `wCustDisplayL.w`. Open `DataUtil.p` and view the **GetCustData** internal procedure. Notice that it has several comments directing you to add code. You will add code here later.
2. Open `src\Try Its\User Interface\wCustDisplayL.w`. It opens in the Design view. Right click on the design view and select **View source** or press **F9** on the keyboard. The source for `wCustDisplayL.w` is displayed in the code editor.

---

**Note:** If you have trouble opening the file in the Design view, right-click `wCustDisplayL.w` in the **Project Explorer** pane and select **Open With > OpenEdge ABL Editor** to open the source view.

---

3. The procedure, `wCustDisplayL.w`, has an internal procedure named **InitializeObjects**. Add code to **InitializeObjects** to do the following:
  - a. Define a handle variable for the external procedure.
  - b. Run the external procedure persistently.
  - c. Run the internal procedure **GetCustData** that is in `DataUtil.p`.

---

**Note:** You will need to include an output parameter for the temp-table.

---

4. Next, you modify the internal procedure, `getCustData` in the called procedure, `src\Examples\Business Logic\DataUtil.p`, to receive the passed data and to retrieve the chosen record. Add code to the procedure to **getCustData** to do the following:
  - a. Define a parameter to receive the data passed from the calling procedure.
  - b. Delete the old customer records if they exist.
  - c. Write the code to create the temp-table for the Customer table. Do not forget to include error handling.

---

**Note:** Ensure that `src\Examples\Business Logic\` is the first path in the PROPATH definition.

---

5. Run `wCustDisplayL.w` to test your logic.

### Wrap-up

In this Try It, you wrote UI triggers, passed temp-table parameters to running procedures, and used message boxes to debug your code.

# Solutions

---

This section contains the following topics:

- [Try It 1.1: Using FIND, DISPLAY, and REPEAT, Solutions](#)
- [Try It 1.2: Using FIND, FOR EACH, and WHERE, Solutions](#)
- [Try It 1.3: Using frames, Solutions](#)
- [Try It 1.4: Using BEGINS, CONTAINS, and MATCHES, Solutions](#)
- [Try It 1.5: Sorting records, Solutions](#)
- [Try It 1.6: Displaying messages in a dialog box, Solutions](#)
- [Try It 2.1: Define variables, Solutions](#)
- [Try It 2.2: Dates and numbers, Solutions](#)
- [Try It 2.3: Data types, comparisons, strings, lists, and functions, Solutions](#)
- [Try It 2.4: Conditional logic, Solutions](#)
- [Try It 3.1: Write a simple query using both functions, Solutions](#)
- [Try It 3.2: Join tables in a query, Solutions](#)
- [Try It 3.3: Scroll a query results list, Solutions](#)
- [Try It 4.1: Create a new record, Solutions](#)
- [Try It 4.2: Modify a sequence, Solutions](#)
- [Try It 4.3: Modify a database trigger, Solution](#)
- [Try It 4.4: Modify records and assign values, Solutions](#)
- [Try It 4.5: Delete records, Solutions](#)
- [Try It 5.1: Identify the transaction scope, Solutions](#)
- [Try It 5.2: Modify the transaction scope, Solutions](#)
- [Try It 6.1 : Identify record scope and locking duration, Solutions](#)
- [Try It 6.2: Implement optimistic locking, Solutions](#)
- [Try It 7.1: Overriding the default error handling, Solutions](#)
- [Try It 7.2: Checking for errors, Solutions](#)

- [Try It 7.3: Using ON-ERROR, Solutions](#)
- [Try It 8.1: Parameters and functions, Solutions](#)
- [Try It 8.2: Persistent procedures, Solutions](#)
- [Try It 9.1 : Integrate logic into an application, Solutions](#)

## Try It 1.1: Using FIND, DISPLAY, and REPEAT, Solutions

### Part 1—Write statements to find, display and repeat records

Test your statements by running them in the Progress Developer Studio.

1. Find and display all information in the State table. Save the file as `sState.p`.

```
/* sState.p */

REPEAT:
    FIND NEXT State.
    DISPLAY State.
END.
```

2. Find and display the first name, last name, and position for all employees. Save the file as `sEmployee.p`.

```
/* sEmployee.p */

REPEAT:
    FIND NEXT Employee.
    DISPLAY FirstName LastName Position.
END.
```

3. Find and display all information in the **SalesRep** table except **MonthQuota**. Save the file as `sSalesrep.p`.

```
/* sSalesrep.p */

REPEAT:
    FIND NEXT SalesRep.
    DISPLAY SalesRep EXCEPT MonthQuota.
END.
```

# Try It 1.2: Using FIND, FOR EACH, and WHERE, Solutions

## Part 1—Determine the correct statement

- For each of the following scenarios determine if you would use a FIND or FOR EACH statement to get the data outlined in the scenario.

	Scenario	FIND	FOR EACH
a.	All data for the first item in the Item table.	X	
b.	The name and city for all customers.		X
c.	The name of the last SalesRep in the database SalesRep table. <b>Hint:</b> The SalesRep name field is RepName.	X	
d.	A list of names of all suppliers. <b>Hint:</b> The table name is Supplier.		X

## Part 2—Retrieve and display data

- Write separate procedures to retrieve and display data for each scenario (a – d above). Test your procedures by running them in the Progress Developer Studio. Save them with the following names:

**a.** sFirstItem.p

```
FIND FIRST Item.
DISPLAY Item EXCEPT ItemImage WITH 1 COLUMN.
```

**b.** sCustomers.p

```
FOR EACH Customer:
  DISPLAY Name City.
END.
```

**c.** sLastSalesrep.p

```
FIND LAST SalesRep.
DISPLAY RepName.
```

**d.** sSuppliers.p

```
FOR EACH Supplier:
  DISPLAY Name.
END.
```

## Part 3—Construct ABL procedures

You can use either FOR EACH or FIND and REPEAT for these examples. Both solutions are shown below.

### 1. sNotUSA.p

```
FOR EACH Customer WHERE Country <> "USA":  
    DISPLAY Name Country.  
END.  
  
FindCust:  
REPEAT:  
    FIND NEXT Customer WHERE Country <> "USA".  
    DISPLAY Name Country.  
END. /* FindCust */
```

### 2. sAfter.p

```
FOR EACH Order WHERE ShipDate > 6/1/1998:  
    DISPLAY Order WITH 1 COLUMN.  
END.  
  
FindOrder:  
REPEAT:  
    FIND NEXT Order WHERE ShipDate > 6/1/1998.  
    DISPLAY Order WITH 1 COLUMN.  
END. /* FindOrder */
```

### 3. sCustBalance.p

```
FOR EACH Customer WHERE Balance >= 30000:  
    DISPLAY Name Balance.  
END.  
  
FindCust:  
REPEAT:  
    FIND NEXT Customer WHERE Balance >= 30000.  
    DISPLAY Name Balance.  
END. /* FindCust */
```

## Try It 1.3: Using frames, Solutions

### Part 1—Create and display colored frames

#### sColoredFrames.p

```
/* sColoredFrames.p */  
DEFINE FRAME f1 WITH TITLE "Color 1" WITH BGCOLOR 1.  
DEFINE FRAME f2 WITH TITLE "Color 2" WITH BGCOLOR 2.  
DEFINE FRAME f3 WITH TITLE "Color 3" WITH BGCOLOR 3.  
DEFINE FRAME f4 WITH TITLE "Color 4" WITH BGCOLOR 4.  
DEFINE FRAME f5 WITH TITLE "Color 5" WITH BGCOLOR 5.  
DEFINE FRAME f6 WITH CENTERED WITH TITLE "Color 6" WITH BGCOLOR 6.  
DEFINE FRAME f7 WITH CENTERED WITH TITLE "Color 7" WITH BGCOLOR 7.  
DEFINE FRAME f8 WITH CENTERED WITH TITLE "Color 8" WITH BGCOLOR 8.  
DEFINE FRAME f9 WITH CENTERED WITH TITLE "Color 9" WITH BGCOLOR 9.  
DEFINE FRAME f10 WITH CENTERED WITH TITLE "Color 10" WITH BGCOLOR 10.  
DISPLAY "Color #1" WITH FRAME f1.  
PAUSE.  
DISPLAY "Color #2" WITH FRAME f2.  
PAUSE.  
DISPLAY "Color #3" WITH FRAME f3.
```



```

PAUSE.
DISPLAY "Color #4" WITH FRAME f4.
PAUSE.
DISPLAY "Color #5" WITH FRAME f5.
PAUSE.
DISPLAY "Color #6" WITH FRAME f6.
PAUSE.
DISPLAY "Color #7" WITH FRAME f7.
PAUSE.
DISPLAY "Color #8" WITH FRAME f8.
PAUSE.
DISPLAY "Color #9" WITH FRAME f9.
PAUSE.
DISPLAY "Color #10" WITH FRAME f10.

```

### Questions:

1. Where do frames 1 – 5 appear on the default window?  
**Answer:** On the left side of the default frame.
2. Where do frames 6 – 10 appear on the default window?  
**Answer:** In the center of the default frame.
3. What color is #9?  
**Answer:** Blue.

## Part 2—Display and hide frames

### sHideFrames.p

```

/* sHideFrames.p */

DEFINE FRAME f1 WITH TITLE "Color 1" WITH BGCOLOR 1.
DEFINE FRAME f2 WITH TITLE "Color 2" WITH BGCOLOR 2.
DEFINE FRAME f3 WITH TITLE "Color 3" WITH BGCOLOR 3.
DEFINE FRAME f4 WITH TITLE "Color 4" WITH BGCOLOR 4.
DEFINE FRAME f5 WITH TITLE "Color 5" WITH BGCOLOR 5.
DEFINE FRAME f6 WITH CENTERED WITH TITLE "Color 6" WITH BGCOLOR 6.
DEFINE FRAME f7 WITH CENTERED WITH TITLE "Color 7" WITH BGCOLOR 7.
DEFINE FRAME f8 WITH CENTERED WITH TITLE "Color 8" WITH BGCOLOR 8.
DEFINE FRAME f9 WITH CENTERED WITH TITLE "Color 9" WITH BGCOLOR 9.
DEFINE FRAME f10 WITH CENTERED WITH TITLE "Color 10" WITH BGCOLOR 10.

DISPLAY "Color #1" WITH FRAME f1.
PAUSE.
DISPLAY "Color #2" WITH FRAME f2.
PAUSE.
HIDE FRAME f1.
DISPLAY "Color #3" WITH FRAME f3.
PAUSE.
HIDE FRAME f2.
DISPLAY "Color #4" WITH FRAME f4.
PAUSE.
HIDE FRAME f3.
DISPLAY "Color #5" WITH FRAME f5.
PAUSE.
HIDE FRAME f4.
DISPLAY "Color #6" WITH FRAME f6.
PAUSE.
DISPLAY "Color #7" WITH FRAME f7.
PAUSE.
VIEW FRAME f1.
DISPLAY "Color #8" WITH FRAME f8.
PAUSE.
VIEW FRAME f3.

```

```
DISPLAY "Color #9" WITH FRAME f9.  
PAUSE.  
DISPLAY "Color #10" WITH FRAME f10.
```

**Questions:**

1. Which frames are visible on the left side of the default frame, and what is their sequence?

**Answer:** 1, 2, 3, 4, 5

2. Which frames are visible in the center of the default frame?

**Answer:** 6, 7, 8, 9, 10

## Try It 1.4: Using BEGINS, CONTAINS, and MATCHES, Solutions

### Part 1—Determine the correct use of BEGINS, CONTAINS and MATCHES statements

1. sGo.p

```
FOR EACH Item WHERE ItemName MATCHES "*go*":  
    DISPLAY ItemName.  
END.
```

2. sWater.p

```
FOR EACH Item WHERE CatDescription CONTAINS "water":  
    DISPLAY ItemName CatDescription.  
END.
```

3. sHoo.p

```
FOR EACH Customer WHERE Name BEGINS "Hoo":  
    DISPLAY Name.  
END.
```

4. sTennis.p

```
FOR EACH Customer WHERE NAME MATCHES "*tennis*":  
    DISPLAY NAME.  
END.
```

### Part 1—Bonus Tasks

#### sUSA09.p

```
/* sUSA09.p */  
  
FOR EACH customer  
    WHERE Country = "USA"  
    AND (PostalCode BEGINS "0" OR PostalCode begins "9"):  
    DISPLAY NAME Country Postalcode.  
END.
```

## Try It 1.5: Sorting records, Solutions

Part 1—Sorting records on select fields

### sEmployeeNames.p

```
/* sEmployeeNames.p */

FOR EACH employee BY DeptCode BY LastName BY firstName:
    DISPLAY DeptCode LastName FirstName.
END.
```

### sUSCustomers.p

```
/* sUSCustomers.p */

FOR EACH Customer WHERE Country = "USA" BY State BY City BY Name:
    DISPLAY City FORMAT "x(15)" State FORMAT "x(2)" NAME FORMAT "x(30)".
END.
```

### sBirthdates.p

```
/* sBirthdates.p */

FOR EACH Family BY Birthdate DESCENDING:
    DISPLAY EmpNum Birthdate RelativeName.
END.
```

## Try It 1.6: Displaying messages in a dialog box, Solutions

Part 1—Display messages with different alert types

### sMyMessage.p

```
/* sMyMessage.p */
MESSAGE "Here is my message" VIEW-AS ALERT-BOX.
```

### sDepartment.p

```
/* sDepartment.p */
FIND Department WHERE DeptCode = "700".
MESSAGE "Department 700 is" DeptName VIEW-AS ALERT-BOX.
```

### sComments.p

```
/* sComments.p */

FOR EACH customer WHERE custnum < 3010:
    MESSAGE NAME ":" Comments VIEW-AS ALERT-BOX.
END.
```

### sSportsSupplier.p

```
/* sSportsSupplier.p */
```

```
FOR EACH supplier WHERE NAME MATCHES "*Sports*":  
  MESSAGE "Supplier" SupplierIDNum "is" NAME VIEW-AS ALERT-BOX.  
END.
```

# Try It 2.1: Define variables, Solutions

## Solution, Part 1—Specify the data type

Specify ABL data types you would use to hold the following values:

Value	Data type
151	INTEGER
151.19	DECIMAL
One hundred fifty two	CHARACTER
True	LOGICAL
10/10/04	DATE
NO	LOGICAL
January, February, March	CHARACTER
December 31, 2001	CHARACTER
365	INTEGER

**Solution, Part 2—Define variables**

1. Define a variable to be used as a counter that starts from zero.

**Answer:** DEFINE VARIABLE iCounter AS INTEGER INITIAL 0 NO-UNDO.

2. Define a variable to store monthly sales total with two decimal places.

**Answer:** DEFINE VARIABLE dSalesTot AS DECIMAL FORMAT "->>, >>9.99" NO-UNDO.

3. Define a variable to store a value of a country with the initial value USA.

**Answer:** DEFINE VARIABLE cCountry AS CHARACTER INITIAL "USA" NO-UNDO.

4. Define a variable to store a value of either Yes or No.

**Answer:** DEFINE VARIABLE lOk AS LOGICAL NO-UNDO.

5. Define a variable to store an employee's date of birth.

**Answer:** DEFINE VARIABLE dtBirthDate AS DATE NO-UNDO.

6. Define a variable to store a name with the initial value as your name. Specify a display format that is long enough to hold your first and last names.

**Answer:** DEFINE VARIABLE cFullName AS CHARACTER FORMAT "x(25)" INITIAL "John Doe" NO-UNDO.

7. Define a variable to store decimal values. Specify a display format for three positions before and two positions after the decimal point.

**Answer:** DEFINE VARIABLE dNumber AS DECIMAL FORMAT ">>9.99" NO-UNDO.

## Try It 2.2: Dates and numbers, Solutions

**Part 1—Date functions**

1. What date expression would return the month that this employee began work?

**Answer:** MONTH(StartDate)

2. How would you find out the year the employee started work?

**Answer:** YEAR(StartDate)

## Part 2—Numbers

1. Write an expression for each of the following. Write `DISPLAY` or `MESSAGE` statements to view the results.

Description	ABL Expression
Display 256.1234 with two decimal places.	<code>TRUNCATE(256.1234,2)</code>
Convert 15.23 to an integer value.	<code>INTEGER(15.23)</code>
Calculate the remainder when you divide 29 by 3.	<code>29 MODULO 3</code>

2. What does `dAmount` represent in the following code sample?

```
DEFINE VARIABLE dAmount AS DECIMAL NO-UNDO.
ASSIGN dAmount = (orderLine.Price * Orderline.Qty)
               * ((100 - Orderline.Discount) / 100).
```

**Answer:** Extended price including any discount calculations.

3. Write a procedure named `tToday.p` to display the following in a single column:

- Today's date
- The day of the month
- The month
- The year

**Answer:** The following code will produce the desired output.

```
/* tToday.p */

DISPLAY TODAY          LABEL "Today is"
      DAY(TODAY)       LABEL "Day"
      MONTH(TODAY)     LABEL "Month"
      YEAR(TODAY)      LABEL "Year" FORMAT "9999"
      WITH 1 COLUMN.
```

4. Write a procedure that calculates the number of days until the end of the month.

**Answer:** The following code will produce the desired output.

```
/* sEndMonth.p */

DEFINE VARIABLE dtToday AS DATE NO-UNDO.
DEFINE VARIABLE dtEndMonth AS DATE      NO-UNDO .
DEFINE VARIABLE cEndMonth AS CHARACTER NO-UNDO .
DEFINE VARIABLE iNumDays  AS INTEGER   NO-UNDO.
ASSIGN
    dtToday    = TODAY
    dtEndMonth = 11/30/2021
    iNumDays   = dtEndMonth - dtToday .

MESSAGE "There are" iNumDays "days until the end of the month" VIEW-AS ALERT-BOX.
```

---

**Note:** You will have to hard code the **last date** of the **current month** of the **current year** in your procedure.

---

### Part 3—Dates and numbers (Bonus questions)

1. Write a procedure to calculate how many seconds until midnight tonight. Display the results to the screen. Save the file as `tMidnight.p`.
2. Modify `tMidnight.p` to display the number of hours, minutes, and seconds to midnight.

**Answer:** The following code will produce the desired output.

```
/* sMidnight.p */

DEFINE VARIABLE Hour      AS INTEGER NO-UNDO.
DEFINE VARIABLE Minute    AS INTEGER NO-UNDO.
DEFINE VARIABLE Sec       AS INTEGER NO-UNDO.
DEFINE VARIABLE Timeleft  AS INTEGER NO-UNDO.

ASSIGN
    Timeleft = (24 * 60 * 60) - TIME
    Sec      = Timeleft MODULO 60 /* seconds till next midnight */
    Timeleft = (Timeleft - Sec) / 60
    Minute   = Timeleft MODULO 60 /* minutes till next midnight */
    Hour     = (Timeleft - Minute) / 60 /* hours till next midnight */.

MESSAGE "Time to midnight:" SKIP
    "Total seconds:" Timeleft skip
    Hour "hours" Minute "minutes" Sec "seconds" VIEW-AS ALERT-BOX.
```

## Try It 2.3: Data types, comparisons, strings, lists, and functions, Solutions

### Part 1—Compare values

1. The customer's credit limit is less than or equal to the customer's balance.
2. The customer's sales representative is JAL and the customer's country is not the USA.
3. The customer's number is less than or equal to 10.
4. The customer's city is either Boston or Bedford.

**Answer:** `CreditLimit <= balance`

**Answer:** `salesrep = "JAL" AND country <> "USA"`

**Answer:** `CustNum <= 10`

**Answer:** `City = "Boston" OR city = "Bedford"`

### Solution, Part 2—Strings and lists

#### sPhoneNum.p

```
/* sPhoneNum.p */

DEFINE VARIABLE cPhoneNum AS CHARACTER
    FORMAT "(999)999-9999" INITIAL "7812802000" NO-UNDO.
DEFINE VARIABLE cAreaCode AS CHARACTER NO-UNDO.

cAreaCode = SUBSTRING(cPhoneNum,1,3).
DISPLAY "My area code is" cAreaCode FORMAT "x(3)" NO-LABEL.
```

## Solutions, Part 3—Access items in a list

This code will satisfy the exercise requirements:

```
/* sList.p */
/* This is a suggested list. */
/* You can list any items you like */

DEFINE VARIABLE cWine AS CHARACTER
  INITIAL "Zinfandel,Sake,Port,Chardonnay,Chianti,Merlot,Shiraz" NO-UNDO.

MESSAGE SUBSTITUTE("There are &1 entries in the list", NUM-ENTRIES(cWine)) SKIP(1)
  SUBSTITUTE("The third entry is &1", ENTRY(3,cWine)) VIEW-AS ALERT-BOX.
```

## Try It 2.4: Conditional logic, Solutions

### Solution, Part 1—Scenario 1

```
/* sWeekday.p */

CASE WEEKDAY(TODAY):
  WHEN 2 OR WHEN 3
    THEN DISPLAY "It's the beginning of the week".
  WHEN 4
    THEN DISPLAY "It's Wednesday".
  WHEN 5 or WHEN 6
    THEN DISPLAY "The weekend is coming soon".
  OTHERWISE DISPLAY "It's the weekend".
END CASE.
```

### Solution, Part 1—Scenario 2

```
/* sFamily.p */

DEFINE VARIABLE cGender AS CHARACTER NO-UNDO.
DEFINE VARIABLE cRelation AS CHARACTER NO-UNDO.
FOR EACH family
  WHERE EmpNum <= 20
    AND (Relation = "Son" OR Relation = "Daughter"):
    ASSIGN cGender = IF Relation = "Son"
      THEN "Male"
      ELSE "Female".
  DISPLAY EmpNum Relation RelativeName cGender.
END.
```

### Solution, Part 3—Scenario 3

#### Using DO WHILE

```
/* sCountingWhile.p */

DEFINE VARIABLE iCounter AS INTEGER INITIAL 2 NO-UNDO.
DO WHILE iCounter < 21:
  MESSAGE "Even numbers from 2 to 20:" iCounter VIEW-AS ALERT-BOX.
  ASSIGN iCounter = iCounter + 2.
END.
```



**Using DO . . . BY**

```
/* sCountingDo.p */

DEFINE VARIABLE iCounter AS INTEGER INITIAL 2 NO-UNDO.
DO iCounter = 2 TO 20 BY 2:
    MESSAGE "Even numbers from 2 to 20:" iCounter VIEW-AS ALERT-BOX.
END.
```

**Solution, Part 4—Scenario 4**

This code will satisfy the exercise requirements:

```
/* sFrench.p */

DEFINE VARIABLE lChoice AS LOGICAL NO-UNDO.
MESSAGE "Do you speak French?"
VIEW-AS ALERT-BOX QUESTION BUTTONS YES-NO
TITLE "Language Choice " UPDATE lChoice.
IF lChoice THEN
    MESSAGE "Bonjour. C'est un beau jour."
    VIEW-AS ALERT-BOX TITLE "Réponse française".
ELSE
    MESSAGE "Hello. it's a beautiful day."
    VIEW-AS ALERT-BOX TITLE "English response".
```

## Try It 3.1: Write a simple query using both functions, Solutions

**Write simple queries**

1. Create a procedure that defines a query for the Item table, displaying the item number, name, price and amount on hand.

1. Use the IF NOT AVAILABLE syntax for the query.

2. Save the procedure as tQueryItem.p.

```
/* sQueryItem.p */

DEFINE QUERY qItem FOR ITEM.

OPEN QUERY qItem FOR EACH ITEM.

REPEAT:
    GET NEXT qItem.
    IF NOT AVAILABLE Item THEN LEAVE.
    DISPLAY ItemNum ItemName Price Onhand.
END.
CLOSE QUERY qItem.
```

2. Create a procedure that defines a query for every employee in department 500. Display the employee number, first name, last name, and department.

1. Use the QUERY-OFF-END syntax.
2. Save the procedure as tQueryEmployee.p.

```
/* sQueryEmployee.p */

DEFINE QUERY qEmp FOR Employee.

OPEN QUERY qEmp FOR EACH Employee WHERE DeptCode = "500".

REPEAT:
  GET NEXT qEmp.
  IF QUERY-OFF-END("qEmp") THEN LEAVE.
  DISPLAY EmpNum FirstName LastName DeptCode.
END. /* REPEAT */
CLOSE QUERY qEmp.
```

## Try It 3.2: Join tables in a query, Solutions

1. Create a query that:
  1. Finds the orders for each customer.
  2. Sorts the results list by customer number and order date.
  3. Displays the customer number and name, the order number, and the order date.
4. Save the procedure as lMult.p.

```
/* sMult.p */

DEFINE QUERY qOrder FOR Customer, Order.

OPEN QUERY qOrder FOR EACH Customer,
                      EACH Order OF Customer
                      BY Customer.CustNum
                      BY OrderDate.

REPEAT:
  GET NEXT qOrder.
  IF NOT AVAILABLE Order THEN LEAVE.
  DISPLAY Customer.CustNum NAME OrderNum OrderDate.
END.
CLOSE QUERY qOrder.
```

2. Modify the code in lMult.p so that you show only the first order for each customer.
- Save the procedure as lMult2.p.

```
/* sMult2.p */

DEFINE QUERY qOrder FOR Customer, Order.

OPEN QUERY qOrder FOR EACH Customer,
                      FIRST Order OF Customer
                      BY Customer.CustNum
                      BY OrderDate.

REPEAT:
  GET NEXT qOrder.
  IF NOT AVAILABLE(Order) THEN LEAVE.
  DISPLAY Customer.CustNum NAME OrderNum OrderDate.
END.
CLOSE QUERY qOrder.
```

## Try It 3.3: Scroll a query results list, Solutions

### Update a scrolling query

The First and Last buttons will function correctly with the following code. /\* sEmpFam.w \*/

```
/* sEmpFam.w */
. . .
ON CHOOSE OF BtnFirst IN FRAME DEFAULT-FRAME
DO:
  GET FIRST brEmployee.
  REPOSITION brEmployee TO ROWID ROWID(Employee).
END.

ON CHOOSE OF BtnLast IN FRAME DEFAULT-FRAME
DO:
  GET LAST brEmployee.
  REPOSITION brEmployee TO ROWID ROWID(Employee).
END.
. . .
```

## Try It 4.1: Create a new record, Solutions

### Solution, Create a record

#### sNewLineNum.p

```
/* sNewLineNum.p */

DEFINE VARIABLE iOrderLine AS INTEGER.

FIND LAST Order.
FIND LAST Orderline OF Order.
IF AVAILABLE OrderLine THEN
  ASSIGN iOrderLine = Orderline.LineNum + 1.
IF NOT AVAILABLE OrderLine THEN
  ASSIGN iOrderLine = 1.

CREATE Orderline.
ASSIGN Orderline.LineNum = iOrderLine
      Orderline.OrderNum = Order.OrderNum
      ItemNum = 7
      Price = 1.20
      Qty = 6.
DISPLAY orderline WITH 1 COLUMN.
```

## Try It 4.2: Modify a sequence, Solutions

### Solution, Modify a sequence

The following procedure satisfies the requirements for `sNextCustNum.p`.

```
/* sNextCustNum.p */

FIND LAST Customer.
display Custnum Name.
PAUSE.
CREATE Customer.
ASSIGN Name = "Weinberg".
DISPLAY Custnum Name.
```

To modify the sequence do the following:

1. Open the Data Dictionary.
2. Click the **Sequences** icon.
3. Select **NextCustNum**.
4. Select the **Sequence Properties** button.
5. Change the value in the *Increment by* field from 5 to 10.
6. Save your changes, click **OK**, and exit from the Data Dictionary.
7. Press **Yes** in the warning dialog to commit your changes.

**Answer:** The new customer number should be incremented by 10.

## Try It 4.3: Modify a database trigger, Solution

### Solution, Modify a database trigger

The following procedure will create the new order and display the order date and promise date:

```
/* sCreateOrder.p */

CREATE Order.
DISPLAY Ordernum OrderDate PromiseDate.
```

The following procedure is the modified database trigger for creating orders:

```
/* *****
   Program: CrOrd.p
   ***** */

TRIGGER PROCEDURE FOR Create OF Order.

/* Automatically Increment Order Number using NextOrdNum Sequence */
ASSIGN Order.OrderNum = NEXT-VALUE(NextOrdNum)

/* Set Order Date to TODAY, Promise Date to 1 week from TODAY */
Order.OrderDate = TODAY
Order.PromiseDate = TODAY + 7.
```

**Answer:** The promise date should be in 7 days.

## Try It 4.4: Modify records and assign values, Solutions

### Modify a record

The following code should satisfy the specified requirements.

```
/* sIncreasePrice.p */
DEFINE VARIABLE dPrice AS DECIMAL NO-UNDO.

FOR EACH orderline WHERE ordernum = 10000:
    ASSIGN dPrice = Price
    Price = Price * 1.25.
    DISPLAY Linenum
        Price LABEL "New price"
        dPrice LABEL "Old price" FORMAT "$>, >>9.99".
END.
```

## Try It 4.5: Delete records, Solutions

### Delete a record

#### lDelFail.p

```
FIND Customer WHERE Customer.CustNum = 3255.
DELETE Customer.
```

#### sDelParentChild.p

```
/* sDelParentChild.p */

DEFINE VARIABLE iCustNum      AS INTEGER INITIAL 3255    NO-UNDO.
DEFINE VARIABLE lYesno        AS LOGICAL                NO-UNDO.

/* Delete Children then Parent */
FOR EACH Invoice WHERE Invoice.CustNum = iCustNum:
    MESSAGE "Are you sure you want to delete invoice#" invoice.invoicenum
        "of customer #" invoice.custnum VIEW-AS ALERT-BOX BUTTONS YES-NO
    UPDATE lYesno.

    IF lYesno THEN
        DELETE Invoice.
    ELSE DO: /* DO statement #1 */
        MESSAGE "Cancelled deletion of invoice #" invoice.invoicenum .
        RETURN NO-APPLY.
    END. /* End DO statement #1 */
END. /* End FOR EACH Invoice3 */

FOR EACH Order WHERE Order.CustNum = iCustNum:
    MESSAGE "Are you sure you want to delete order #" Order.Ordernum
        "of customer #" Order.CustNum VIEW-AS ALERT-BOX BUTTONS YES-NO UPDATE lYesno.
    IF lYesno THEN
        DELETE Order.
    ELSE DO: /* DO statement #2 */
        MESSAGE "Cancelled deletion of order #" Order.Ordernum .
        RETURN.
    END. /* End DO statement #2 */
```

```
END. /* End FOR EACH Order */

FIND Customer WHERE Customer.CustNum = iCustNum.
  MESSAGE " Are you sure you want to delete customer #" customer.custnum
    VIEW-AS ALERT-BOX BUTTONS YES-NO UPDATE lYesno.
IF lYesno THEN
  DELETE Customer.
ELSE DO: /* DO statement #3 */
  MESSAGE "Cancelled deletion of customer #" customer.custnum .
  RETURN.
END. /* DO statement #3 */
```

## Try It 5.1: Identify the transaction scope, Solutions

### Part 1—Identify the transaction scope 1

The RepOrdbl block and the ForCustblk block both determine transaction scope. The Repordlineblk block is a subtransaction block within an existing transaction, RepOrdbl.

**Listing file:**

```

{} Line Blk
-----
1      /* lTrans1.p */
2      DEFINE VARIABLE iLineNum AS INTEGER NO-UNDO.
3      RepOrdblk:
4      1 REPEAT:
5      1   CREATE Order.
6      1   DISPLAY OrderNum.
7      1   UPDATE Order.OrderDate Order.CustNum Order.Carrier.
8      1   FIND Customer OF Order.
9      1   iLineNum = 0.
10     1 RepOrdlineblk:
11     2 REPEAT:
12     2   CREATE OrderLine.
13     2   ASSIGN OrderLine.OrderNum = Order.OrderNum
14     2       iLineNum = iLineNum + 1
15     2       OrderLine.LineNum = iLineNum.
16     2   ASSIGN Qty = 0.
17     2   DISPLAY LineNum.
18     2   UPDATE OrderLine.ItemNum Qty.
19     1   END. /* RepOrdlineblk */
20     END. /* RepOrdblk */
21     ForCustblk:
22     1 FOR EACH Order OF Customer:
23     1   DISPLAY Order.CustNum Order.OrderNum.
24     1   UPDATE Customer.SalesRep.
24     END. /* ForCustblk */

```

File Name	Line Blk.	Type	Tran	Blk. Label
...lTrans1.p	0	Procedure	No	
Buffers: sports2020.Customer sports2020.Order				
...lTrans1.p	4	Repeat	Yes	RepOrdblk
Frames: Unnamed				
...lTrans1.p	11	Repeat	Yes	RepOrdlineblk
Buffers: sports2020.OrderLine Frames: Unnamed				
...lTrans1.p	22	For	Yes	ForCustblk
Frames: Unnamed				

## Part 2—Identify the transaction scope 2

The Repblk block determines the transaction scope. The DO blocks do not have the transaction property associated with them.

**Listing file:**

```

1      /* lTrans2.p */
2      Repblk:
3  1 REPEAT:
4  1     PROMPT-FOR ItemName.
5  1     FIND item USING ItemName NO-ERROR.
6  1     IF NOT AVAILABLE item THEN
7  1         AskQuestion:
8  2         DO:
9  2             MESSAGE "The item" INPUT ItemName "does not exist."
10 2             "Do you wish to add this item?"
11 2             VIEW-AS ALERT-BOX QUESTION BUTTON YES-NO
12 2             UPDATE lAnswer AS LOGICAL.
13 3             DO:
14 3             IF lAnswer THEN
15 3                 CREATE ITEM.
16 3                 ASSIGN ItemName.
17 3                 SET ITEM EXCEPT ItemNum.
18 2             END. /* AskQuestion */
19 1             END.
20 1     DISPLAY Item.ItemName WITH 1 COLUMN.
20 END. /* Repblk */

```

File Name	Line	Blk. Type	Tran	Blk. Label
...\1Trans2.p	0	Procedure	No	
...\1Trans2.p	3	Repeat	Yes	Repblk
Buffers: sports2020.Item				
Frames: Unnamed				
...\1Trans2.p	8	Do	No	AskQuestion
...\1Trans2.p	13	Do	No	

## Try It 5.2: Modify the transaction scope, Solutions

## Part 1—Modify the transaction scope

The transaction scope can be decreased with a `DO TRANSACTION` in the `CreateItem` block.

sTrans3.p



```

1  /* sTrans3.p */
2  Repblk:
3  REPEAT:
4      PROMPT-FOR ItemName.
5      FIND item USING ItemName NO-ERROR.
6      IF NOT AVAILABLE item THEN
7          AskQuestion:
8          DO :
9              MESSAGE "The item" INPUT ItemName "does not exist."
10             "Do you wish to add this item?"
11             VIEW-AS ALERT-BOX QUESTION BUTTON YES-NO
12             UPDATE lAnswer AS LOGICAL.
13             IF lAnswer THEN
14                 CreateItem:
15                 DO TRANSACTION:
16                     CREATE ITEM.
17                     ASSIGN ItemName.
18                     SET ITEM EXCEPT ItemNum.
19                 END. /* CreateItem */
20             ELSE LEAVE.
21         END. /* AskQuestion */
22     DISPLAY Item.ItemName WITH 1 COLUMN.
23 END. /* Repblk */

```

Listing file for sTrans3.lst:

```
{ } Line Blk
-----
1      /* sTrans3.p */
2      Repblk:
3      1 REPEAT:
4      1      PROMPT-FOR ItemName.
5      1      FIND item USING ItemName NO-ERROR.
6      1      IF NOT AVAILABLE item THEN
7      1      AskQuestion:
8      2      DO :
9      2      MESSAGE "The item" INPUT ItemName "does not exist."
10     2      "Do you wish to add this item?"
11     2      VIEW-AS ALERT-BOX QUESTION BUTTON YES-NO
12     2      UPDATE lAnswer AS LOGICAL.
13     2      IF lAnswer THEN
14     2      CreateItem:
15     3      DO TRANSACTION:
16     3      CREATE ITEM.
17     3      ASSIGN ItemName.
18     3      SET ITEM EXCEPT ItemNum.
19     2      END. /* CreateItem */
20     2      ELSE LEAVE.
21     1      END. /* AskQuestion */
22     1      DISPLAY Item.ItemName WITH 1 COLUMN.
23     END. /* Repblk */
...esson05\sTrans3.p      11/05/2021 20:03:30 PROGRESS(R) Page 2
```

File Name	Line	Blk.Type	Tran	Blk. Label
...esson05\sTrans3.p	0	Procedure	No	
...esson05\sTrans3.p	3	Repeat	No	Repblk
Buffers: sports2020.Item				
Frames: Unnamed				
...esson05\sTrans3.p	8	Do	No	AskQuestion
...esson05\sTrans3.p	15	Do	Yes	CreateItem

# Try It 6.1 : Identify record scope and locking duration, Solutions

## Solution, Part 1—Identify the transaction scope

**Answer:** The scope of the transaction is the outer repeat block, repblk.

## Solution, Part 2—Identify the scope of each record buffer

This record buffer...	Is scoped to this block...
Customer	repblk
Order	forblk1

This record buffer...	Is scoped to this block...
OrderLine	forblk2
Item	forblk2

### Solution, Part 3—Identify the type and duration of the lock on each record

Record	Lock type	Block in which the lock is acquired	Upgraded or released in.
Customer	SHARE-LOCK	repblk	Upgraded
	EXCLUSIVE-LOCK	repblk	repblk
Order	SHARE-LOCK	forblk1	repblk
OrderLine	SHARE-LOCK	forblk2	repblk
Item	SHARE-LOCK	forblk2	repblk

### Questions

Consider these questions and note your answers:

1. How does the locking strategy of this procedure affect other users' access to the database?

**Answer:** If every OrderLine record for every order for the customer in question is share locked then the records are unavailable for update to other users until the end of the transaction (end of REPEAT block).

2. How does the locking strategy of this procedure affect the execution of this procedure?

**Answer:** The AVM might suspend execution of the inner FOR EACH block if the procedure must wait for an upgrade from SHARE-LOCK to EXCLUSIVE-LOCK for a Customer record.

---

**Note:** Correct responses are not limited to the answers presented here.

---

## Try It 6.2: Implement optimistic locking, Solutions

### Solution, Part 1—Run a simple procedure in a multi-user environment

**Answer:** You get a message saying the record is locked (Department is in use) by another user.

## Solution, Part 2—Simple optimistic locking without checking availability

### sChangeDept2.p

```

/* sChangeDept2.p */

DEFINE VARIABLE lChanged AS LOGICAL NO-UNDO.

DISPLAY "Department codes are: 100, 200, 300, 400, 500, 600, 700".

REPEAT:
  PROMPT-FOR Department.DeptCode.
  FIND Department NO-LOCK USING Department.DeptCode.
  Retry-blk:
  DO ON ERROR UNDO, RETRY:
    DISPLAY Department.DeptName.
    PROMPT-FOR DeptName.
    Update-blk:
    DO TRANSACTION:
      FIND CURRENT Department EXCLUSIVE-LOCK.
      IF NOT CURRENT-CHANGED Department THEN ASSIGN DeptName.
      ELSE DO:
        MESSAGE "The data has changed while you were working." SKIP
          "Please try again." VIEW-AS ALERT-BOX.
        UNDO retry-blk, RETRY retry-blk.
      END. /* Data changed, retry */
    END. /* Transaction ends */
  END. /* Retry-blk */
END. /* REPEAT */

```

**Test 1:** You are notified that the department record has been changed by another user and you are requested to try again.

---

**Note:** There is **never** lock contention when running lChangeDept2.p in both sessions because it uses optimistic locking..

---

**Test 2:** The user is notified that the record is in use by another user.

This is because lChangeDept1.p does not use optimistic locking, so you will still get lock contention. To avoid lock contention, all processes accessing a record should use optimistic locking.

## Solution, Part 3 – Complete optimistic locking including a check for availability

### sChangeDept3.p

```

/* sChangeDept3.p */

DEFINE VARIABLE lChanged AS LOGICAL NO-UNDO.
DISPLAY "Department codes are: 100, 200, 300, 400, 500, 600, 700".

REPEAT:
  PROMPT-FOR Department.DeptCode.
  FIND Department USING Department.DeptCode NO-LOCK NO-ERROR.
  IF AVAILABLE Department THEN
  DO: /* There is such a Department */
    DISPLAY Department.DeptName.
    PROMPT-FOR DeptName.
    Update-blk:
    DO TRANSACTION:
      FIND CURRENT Department EXCLUSIVE-LOCK NO-WAIT NO-ERROR.
      IF AVAILABLE Department THEN DO:
        IF NOT CURRENT-CHANGED Department THEN
          ASSIGN DeptName.
        ELSE
          lChanged = YES. /* Indicate data was changed */
      END.
    END.
  END.

```

```

ELSE DO: /* Record not available */
  IF LOCKED Department THEN DO: /* The record is locked */
    MESSAGE "The Department record is locked." SKIP
    "Choose 'OK' to try again; 'Cancel' to select another Department."
    VIEW-AS ALERT-BOX BUTTONS OK-CANCEL
    UPDATE lRetry AS LOGICAL.
    IF lRetry THEN UNDO Update-blk, RETRY Update-blk. /* try again */
  END. /* The record is locked */
ELSE /* The record was deleted */
  MESSAGE "Department record has been deleted by another user!"
  VIEW-AS ALERT-BOX.
END. /* Record not available */
END. /* End transaction */

/* Re-read to remove the lingering SHARE-LOCK */
IF AVAILABLE Department THEN
  FIND CURRENT Department NO-LOCK.
/* Show the message outside the transaction after the
lock has been removed */
IF lChanged THEN DO:
  MESSAGE "The data has changed while you were working." SKIP
  "Please try again." VIEW-AS ALERT-BOX.
  lChanged = NO. /* Reset the counter */
END. /* if record changed */
END. /* found a Department */

ELSE /* bad Department number */
  MESSAGE "No such department. Please re-enter number." VIEW-AS ALERT-BOX.
END. /* REPEAT */

/* Re-read to remove the lingering SHARE-LOCK */
IF AVAILABLE Department THEN
  FIND CURRENT Department NO-LOCK.
/* Show the message outside the transaction after the
lock has been removed */
IF lChanged THEN DO:
  MESSAGE "The data has changed while you were working." SKIP
  "Please try again." VIEW-AS ALERT-BOX.
  lChanged = NO. /* Reset the counter */
END. /* if record changed */
END. /* found a Department */

ELSE /* bad Department number */
  MESSAGE "No such department. Please re-enter number." VIEW-AS ALERT-BOX.
END. /* Repeat-blk */

```

**Test 1:** You will see your custom error message for a deleted record which you created in tChangeDept3.p.

**Test 2:** You will see your custom error message for changes in a locked record which you created in tChangeDept3.

## Try It 7.1: Overriding the default error handling, Solutions

### Solution, Part 1—Run the procedure with default error handling

Complete the following steps:

1. Enter the name "Smith". What happens?

**Answer:** The record for the employee named Smith is displayed, and a prompt appears for the next record.

2. Enter the letters "We". What happens?

**Answer:** The record for the employee named Weiss is displayed, and a prompt appears for the next record.

3. Enter the letter "Q". What happens?

**Answer:** An error message is displayed with the following message:

```
FIND FIRST/LAST failed for table Employee. (565)
```

## Solution, Part 2—Override the default error handling

1. Override default error handling for the FIND statement. If the record is not available, display a message to that effect.

```
/* sFindEmp2.p */

REPEAT:
  PROMPT-FOR Employee.LastName LABEL "Last name begins:".
  FIND FIRST Employee
    WHERE Employee.LastName BEGINS INPUT Employee.LastName NO-ERROR.
  IF AVAILABLE Employee THEN
    DISPLAY EmpNum FirstName LastName DeptCode WITH 1 COLUMN.
  ELSE DO:
    MESSAGE "There is no employee with last name beginning with"
      INPUT LastName VIEW-AS ALERT-BOX.
  END.
END.
```

# Try It 7.2: Checking for errors, Solutions

## Solution, Part 1—Modify the DELETE trigger

This is the modified trigger.

```
/* sDelCust2.p */

TRIGGER PROCEDURE FOR DELETE OF Customer.
DEFINE VARIABLE answer AS LOGICAL.

/* Customer record cannot be deleted if outstanding invoices are found */
FIND FIRST invoice OF customer NO-ERROR.
IF AVAILABLE invoice THEN DO:
  IF invoice.amount <= invoice.totalpaid + invoice.adjustment THEN DO:
    MESSAGE "Invoice OK, looking for Orders..."
      VIEW-AS ALERT-BOX.
    FIND FIRST order OF customer NO-ERROR.
    IF NOT AVAILABLE order THEN DO:
      RETURN.
    END.
  ELSE DO:
    MESSAGE "Open orders exist for Customer. Cannot delete."
      VIEW-AS ALERT-BOX.
    RETURN ERROR "Open orders exist for Customer. Cannot delete.".
  END.
END.
ELSE DO:
  MESSAGE "Outstanding Unpaid Invoice Exists. Cannot Delete"
    VIEW-AS ALERT-BOX.
  RETURN ERROR "Outstanding Unpaid Invoice Exists. Cannot Delete".
```

```

END.
END.
ELSE DO:
  FIND FIRST order OF customer NO-ERROR.
  IF NOT AVAILABLE order THEN DO:
    RETURN.
  END.
ELSE DO:
  MESSAGE "Open orders exist for Customer. Cannot delete."
    VIEW-AS ALERT-BOX.
  RETURN ERROR "Open orders exist for Customer. Cannot delete.".
END.
END.

```

---

**Note:** The old MESSAGE code is commented out.

---

### Solution, Part 3—Test the new DELETE trigger

```

/* sTestTrigger.p */

PROMPT-FOR Customer.CustNum.
FIND Customer WHERE CustNum = INPUT Customer.CustNum.
DELETE Customer NO-ERROR.

IF ERROR-STATUS:ERROR THEN
  MESSAGE RETURN-VALUE VIEW-AS ALERT-BOX.

```

## Try It 7.3: Using ON-ERROR, Solutions

### Solution, Part 1—Simulate an error and observe the default error handling

1. Run the procedure lDispCustRep.p.

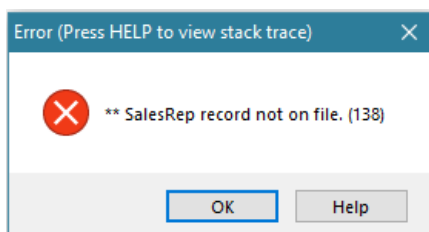
It displays each customer with its SalesRep's name and any existing invoices. It assumes that it will always find a SalesRep.

2. Run the procedure lMakeError.p.

It changes the salesrep value of the first customer record to a value not in the SalesRep table.

3. Run lDispCustRep.p again.

- a. It will produce the following error:



- b. Click **OK**.

- c. You will see the following output. Notice that the **Rep Name** and invoice **Amount** is not displayed for **Lift Tours**.

Cust Num	Name	Rep Name	Amount
3000	Lift Tours		
3005	Urpon Frisbee	Pitt, Dirk K.	4,929
3010	Hoops	Harry Munvig	9,510
3015	Go Fishing Ltd	Smith, Spike Louise	688
3020	Match Point Tennis	Jan Loopsnel	
3025	Fanatical Athletes	Smith, Spike Louise	3,621
3030	Aerobics valine Ky	Pitt, Dirk K.	2,952
3035	Game Set Match	Robert Roller	
3040	Pihtiputaan Pyora	Pitt, Dirk K.	6,428
3045	Just Joggers Limited	Smith, Spike Louise	4,587
3050	Keilailu ja Biljardi	Pitt, Dirk K.	772
3055	Surf Lautaveikkoset	Pitt, Dirk K.	
3060	Biljardi ja tennis	Pitt, Dirk K.	1,005
3065	Paris St Germain	Donna Swindall	19,108
3070	Hoopla Basketball	Harry Munvig	3,235
3075	Thundering Surf Inc.	Jan Loopsnel	2,003
3080	High Tide Sailing	Smith, Spike Louise	
3085	Antin Metsastysase	Pitt, Dirk K.	4,743
3090	Buffalo Shuffleboard	Harry Munvig	2,421
3095	Espoon Pallokeskus	Pitt, Dirk K.	

## Solution, Part 2—Customize the error handling, save, and test

Code for the modified procedure sDispCustRep.p:

```
/* sDispCustRep.p */

FOR EACH Customer NO-LOCK:
    DISPLAY Customer.CustNum FORMAT ">>>9"
    Customer.NAME FORMAT "x(20)".

/* We enclose this code in a DO ON ERROR block to make sure that
   if the FIND raises an error, we still will be able to execute
   the Invoice FIND logic.

   To see the difference in behavior, remove the DO ON ERROR code
   (and the corresponding END statement)

   We also need to add IF AVAILABLE (SalesRep)
   so that we can avoid the invalid Salesrep problem */

DO ON ERROR UNDO, LEAVE:
    FIND SalesRep NO-LOCK OF Customer NO-ERROR.
    IF AVAILABLE SalesRep THEN
        DISPLAY SalesRep.RepName FORMAT "x(20)".
    END.
    FIND FIRST Invoice NO-LOCK OF Customer NO-ERROR.
    IF AVAILABLE(Invoice) THEN
        DISPLAY Invoice.Amount FORMAT ">>, >>9".
    END.
```



## Try It 8.1: Parameters and functions, Solutions

### Solution, Part 1—Pass parameters

#### Step 2

```
INPUT piEmpNum (INTEGER) - Employee number
OUTPUT pcFName (CHARACTER) - First Name
OUTPUT pcLName (CHARACTER) - Last Name
```

#### Step 3

Label	Object Name
Employee number	fiEmpNum
First Name	fiFirst
Last name	fiLast

#### Step 4

```
ON CHOOSE OF btnGetName IN FRAME DEFAULT-FRAME /* Get Name */
DO:
  ASSIGN fiEmpNum.
  RUN lGetEmpName.p (INPUT fiEmpNum, OUTPUT fiFirst, OUTPUT fiLast).
  DISPLAY fiFirst fiLast WITH FRAME {&FRAME-NAME}.
END.
```

### Solution, Part 2—Define parameters

#### Step 3

```
ON CHOOSE OF btnGetState IN FRAME DEFAULT-FRAME /* Get State */
DO:
  ASSIGN fiStateCode.
  RUN valState.p (INPUT fiStateCode, OUTPUT fiStateName).
  DISPLAY fiStateName WITH FRAME {&FRAME-NAME}.
END.
```

#### Step 4

```
/*----- sValState.p -----*/

DEFINE INPUT PARAMETER pcStateCode AS CHARACTER NO-UNDO.
DEFINE OUTPUT PARAMETER pcStateName AS CHARACTER NO-UNDO.

FIND State NO-LOCK
  WHERE State.State = pcStateCode NO-ERROR.
IF AVAILABLE(State) THEN
  ASSIGN pcStateName = State.StateName.
```

### Solution, Part 3—User-defined functions

#### YearsOld function:

```
FUNCTION YearsOld RETURNS INTEGER
( INPUT iEmpNum AS INTEGER ) :
```

```
/*-----  
Purpose: Function design to return employee's age.  
-----*/  
DEFINE VARIABLE iResult AS INTEGER NO-UNDO.  
  
FIND Employee NO-LOCK  
WHERE Employee.EmpNum = iEmpNum NO-ERROR.  
IF AVAILABLE(Employee) THEN  
    ASSIGN iResult = TRUNCATE((TODAY - Employee.Birthdate) / 365,0).  
  
RETURN iResult. /* Function return value. */  
  
END FUNCTION.
```

**Step 6:**

```
ON VALUE-CHANGED OF brEmployee IN FRAME DEFAULT-FRAME  
DO:  
    ASSIGN fiYrsEmployed = YearsEmployed(Employee.EmpNum)  
        fiYrsOld = YearsOld(Employee.EmpNum).  
    DISPLAY fiYrsEmployed fiYrsOld WITH FRAME {&FRAME-NAME}.  
END.
```

## Try It 8.2: Persistent procedures, Solutions

**Solution, Part 1–Persistent procedures****Step 2**

Variable to hold the handle of a persistent procedure.

```
DEFINE VARIABLE hEmp AS HANDLE NO-UNDO.
```

**Step 3**

In the Main Block, start the persistent procedure:

```
RUN lEmpLibrary.p PERSISTENT SET hEmp.
```

**Step 5**

In the Control Triggers section, for the **Get Region** button, run the persistent procedure:

```
ASSIGN fiStateCode.  
RUN valRegion IN hEmp (INPUT fiStateCode,  
    OUTPUT fiStateName,  
    OUTPUT fiRegion).  
DISPLAY fiStateName fiRegion WITH FRAME {&FRAME-NAME}.
```

## Try It 9.1 : Integrate logic into an application, Solutions

### Solution, Part 1—Use a UI trigger

The following code is for the second trigger, **Message2**. Use the same code for the other two with appropriate text strings.

```
ON CHOOSE OF btnMsg2
DO:
    MESSAGE "Welcome to the last lab in this course." VIEW-AS ALERT-BOX.
END.
```

### Solution, Part 2: Run an external Procedure, retrieve and display the data

#### Step 3

Add the following code to the `InitializeObjects` internal procedure in `wCustDisplayL.w`.

```
DEFINE VARIABLE hDataUtil AS HANDLE NO-UNDO.
RUN DataUtil.p PERSISTENT SET hDataUtil.
RUN GetCustData IN hDataUtil (OUTPUT TABLE ttCustomer).
```

#### Step 4

Add the following code to the `GetCustData` internal procedure in `DataUtil.p`:

```
DEFINE OUTPUT PARAMETER TABLE FOR ttCustomer.

EMPTY TEMP-TABLE ttCustomer.

FOR EACH Customer NO-LOCK:
    IF AVAILABLE Customer THEN
        DO:
            CREATE ttCustomer.
            BUFFER-COPY Customer TO ttCustomer NO-ERROR.
        END.
    ELSE RETURN ERROR "Customer records are not available".
END.
```