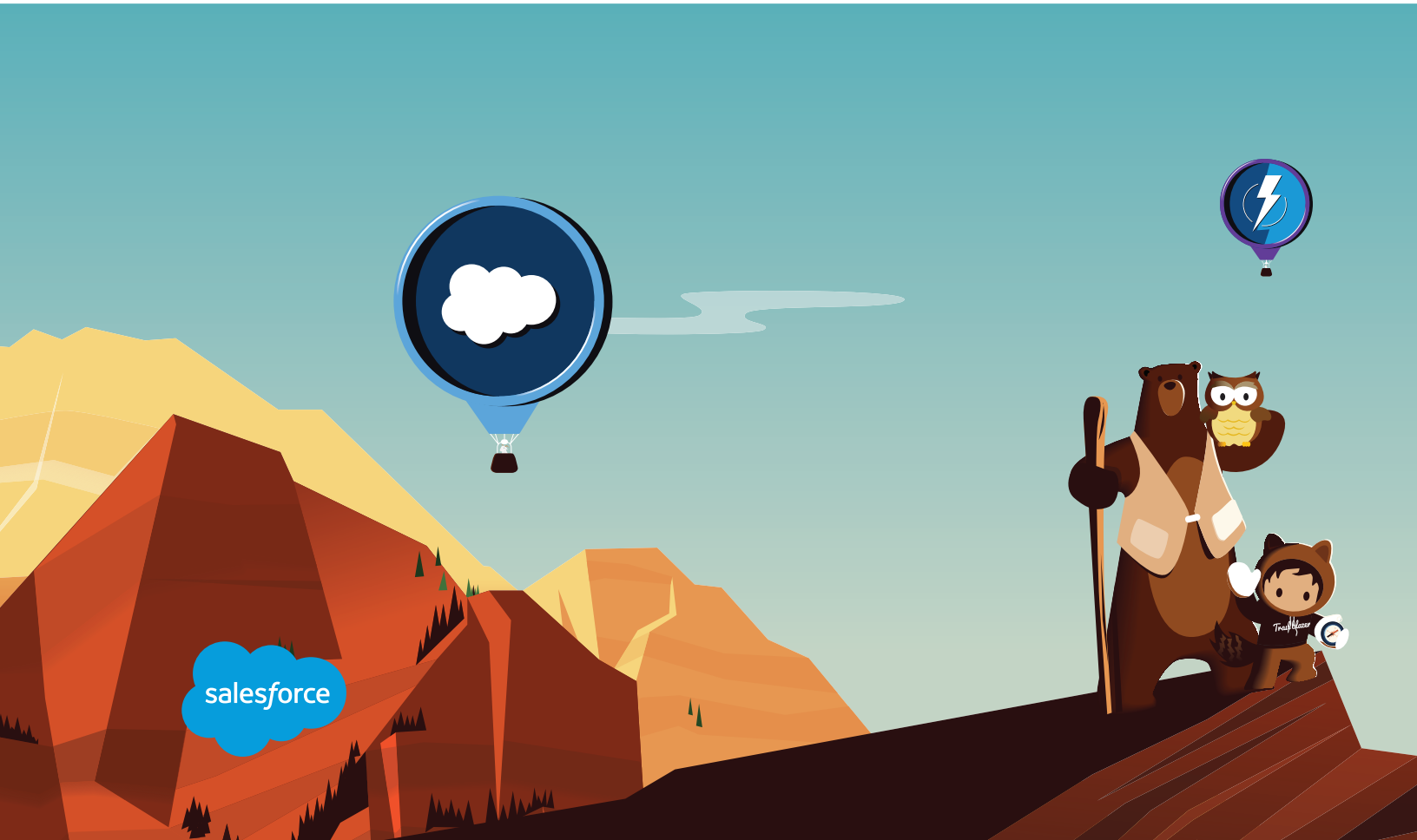
Exercise Guide

Programmatic Development Using Apex and Visualforce

DEV450-S18-V1-EG



1-1: Explore the Certification App 3
1-2: Prepare Your Training Org 5
1-3: Create a Sandbox 7
1-4: Download the Apex Developer's Guide 8
2-1: Create a Custom Object 9
2-2: Create Custom Fields 12
2-3: Create Relationship Fields 16
3-1: Create a Formula Field 21
3-2: Create a Roll-Up Summary Field 23
3-3: Create a Formula Field that References Roll-Up Summary Fields 25
3-4: Understand Record Types 27
4-1: Logging into a Sandbox 28
4-2: See Apex in Action 30
4-3: Create and Use an Apex Class 32
4-4: Observe the Effects of Versioning 33
4-5: Take a Quick Tour of Apex 35
4-6: Examine Implicit Operations 37
4-7: Profile Limits Using the Developer Console 38
4-8: Work with a Custom Object 39
4-9: Use Record IDs to Access a Contact in the UI 40
5-1: Create and Run a Query in the Developer Console 41
5-2: Write a SOQL Query that Uses a WHERE Clause 43
5-3: Write and Execute a SOQL Query in Apex 45
5-4: Write a Dynamic Query in Apex 46
6-1: Write and Test Child-to-Parent Relationship Queries 48
6-2: Query Accounts and Related Contacts 50
7-1: Execute DML Commands 51
7-2: Handle DML Errors and Exceptions 53
8-1: Define a Trigger 54
8-2: Define the Trigger's Business Logic 55
9-1: Refactor a Trigger Using an Apex Class 56
10-1: Explore the Implicit Firing of Triggers 58
10-2: View the Events that Occur During a Rollback 60
10-3: See the Save Order of Execution in Action 62

11-1: Make Test Data Available to Test Methods 64
11-2: Write and Run an Apex Test 65
12-1: Explore Code Coverage 66
13-1: Refactor a Trigger to Avoid SOQL Limits 67
13-2: Refactor a Trigger to Avoid DML Limits 69
14-1: Create a Formula Field to Eliminate a Query 70
14-2: Create Fields for Counting Certification Elements 71
14-3: Create Collections to Filter the Query 73
14-4: Use a Map to Aggregate Results 75
14-5: Create Certification Held Records 76
14-6: Use a Workflow to Avoid Creation of Duplicate Records (Optional) 77
15-1: Create a Simple Visualforce Page 80
15-2: Display Data in a Visualforce Page 81
16-1: Create a Simple Technician Status Page 83
16-2: Refine Your Page and Add Navigational Links 85
17-1: Reference a Controller Extension in a Visualforce Page 86
17-2: Create a Simple Read-Only Property 88
17-3: Creating a Read/Write Property in a Custom Controller 90
17-4: Implement the Search Button 92
17-5: Redirecting to a Results Page 93
17-6: Handle Basic Save Errors in Your Method 95
18-1: Create a Page to Display a List of Records 96
18-2: Integrate SOSL Search in a Visualforce Page 97
18-3: Create a Simple Search Page 98
19-1: Determine Whether a Declarative Solution Exists 100
19-2: Defend Against SOQL Injection 102
19-3: Create a Custom Button that Uses JavaScript (Optional) 104
20-1: Write the Test Method for the Constructor 106
20-2: Write Unit Tests for Action Methods 108
20-3: Write Unit Tests for Getters and Setters 109

1-1: Explore the Certification App

Goal:

Familiarize yourself with the custom certification app.

Tasks:

1. Locate the correct Service Vendor account.
2. Create a new technician record.
3. Sign your new technician up for training.
4. Add a certification attempt for your technician.
5. Document that your technician has earned the certification.

Time:

5 minutes

Instructions:

1. Locate the correct Service Vendor account.
   A. Select the **Certification** app from the Force.com App Picker.
   B. Select the **Accounts** tab.
   C. Choose **Service Vendor Accounts** from the list views menu, and click **Go!** if necessary.
   D. Select **Windy City Network Solutions**.
2. Create a new technician record.
   A. In the Contacts related list, click **New Contact**.
   B. Select the **Technician** record type, and click **Continue**.
   C. Enter the following information:

First Name	`Jonas`
Last Name	`Whittier`

   D. Click **Save**.
3. Sign your new technician up for training.
   A. Click the **Courses** tab.
   B. Click **Go!** to view all courses.
   C. Select **[101] AWCA Server**.
   D. Scroll down to the Course Deliveries related list, and select the upcoming delivery located in Chicago.
   E. In the Course Attendees related list, click **New Course Attendee**.
   F. Click the magnifying glass next to the Student field to open the lookup window, then select **Jonas Whittier**.
   G. Set the status field to **Enrolled**.
   H. Click **Save**.

4. Add a certification attempt for your technician.
   A. Click **Jonas** in the Course Attendees related list to return to the contact record.
   B. Scroll down to the Certification Attempts related list, and click **New Certification Attempt**.
   C. Ensure that Multiple Choice is selected, then click **Continue**.
   D. Enter the following information:

Certification Element	`AWCA Server Multiple Choice`
Certification Candidate (should be pre-populated)	`Jonas Whittier`
Attempt Date	Today's Date
Status	`Complete/Pass`

   E. Click **Save**.
5. Add a Certification Held record for your technician.
   A. Click **AWCA Server Multiple Choice** to view the Certification Element record.
   B. Click **AWCA Server** to view the Certification record.
   C. Scroll down to the Certifications Held related list, and click **New Certification Held**.
   D. Enter the following information:

Certification (should be pre-populated)	`AWCA Server`
Certified Professional	`Jonas Whittier`
Date Achieved	Today's Date

   E. Click **Save**.

1-2: Prepare Your Training Org

Goal:

Prepare your org for classroom activities and access after class.

Tasks:

1. Log in to the training org and reset your user details.
2. Confirm email address.
3. Download your lab files from the Documents tab.
4. Verify the Developer Console settings.

Time:

5 minutes

Instructions:

1. Log in to the training org and reset your user details.

URL:	`login.salesforce.com`
User Name:	`(username)`
Password:	`(password)`

   A. Click **Log in to Salesforce**.
   B. Click **Your Name | My Settings | Personal | Advanced User Details**.
   C. Click **Edit** and change these values.

First Name:	`(your first name)`
Last Name:	`(your last name)`
Email:	`(an email you can access from class)`
Development Mode:	**Select**

   D. Click **Save**.

2. Download your lab files from the Documents tab.
   A. Click the **Documents** tab.
   B. Ensure that **Shared Documents** is selected for the Folder option, and click **Go!**
   C. Click **View** next to LabFiles.
   D. Download the zip file to your Desktop.

E. From Windows Explorer, right-click the zip file and click **7-Zip | Extract Here**.
F. If you do not see 7-Zip as an option, then you can download and install it from http://www.7-zip.org/.

3. Verify the Developer Console settings.

A. Click **Your Name | Developer Console**.
B. In the Developer Console, click **Debug | Perspective Manager**.
C. Set the default perspective to **Log Only (Predefined)**, if it is not already the default.
D. Close the Developer Console.

1-3: Create a Sandbox

Goal:
Create a sandbox to configure and test changes separate from the production environment.

Task:
Create a full sandbox named Dev.

Time:
5 minutes

Instructions:

1. Create a full sandbox named Dev.
   A. Click **Setup.**
   B. Type `sand` in the Quick Find textbox.
   C. Click **Sandboxes** under Deploy.
   D. Click **New Sandbox** and enter the sandbox information.

Name:	`Dev`
Description:	`This is a development environment.`

   E. In the Full column, click **Next**. (Warning: Make sure you select this correctly!)
   F. Select the sandbox options.

Object Data Included:	**All**
Include Field Tracking History Data:	**Deselect**
Include Chatter Data:	**Select**

   G. Click **Create**.

   Note: The time it takes to complete the full copy will depend on the length of the queue at the time requested. You will receive an email letting you know when it has been completed.

1-4: Download the Apex Developer's Guide

Goal:
Download the Apex Developer's Guide to use as a resource.

Task:
Download the Apex Developer's Guide.

Time:
5 minutes

Instructions:

1. Download the Apex Developer's Guide.
   A. In a new browser tab, navigate to https://developer.salesforce.com.
   B. Type `What is Apex?` in the search window, and click the magnifying glass to search.
   C. Select **What is Apex? | Apex Developer Guide ...**
   D. Right click the **PDF** button in the upper left hand corner and select **Save Link As…**
   E. Choose the Desktop as the location, and click **Save** to download the guide as a PDF.

2-1: Create a Custom Object

Goal:

Create a custom object to track customer success stories.

Tasks:

1. View the Account and Contact standard objects in Schema Builder.
2. Create a custom object from Schema Builder.
3. View the object detail page in the Setup menu.
4. Edit the object permissions on the Sales User and Marketing User profiles.
5. View the organization-wide default setting for the new object.

Time:

15 minutes

Instructions:

1. View the Account and Contact standard objects in Schema Builder.
   A. Click **Setup**.
   B. Type `schema` in the Quick Find textbox.
   C. Click **Schema Builder** under Build.
   D. Click **Clear All**.
   E. Select the **Account** and **Contact** objects.
   F. Rearrange the objects so they are visible in the canvas.
   G. Click **View Options | Display Element Names** if it is an available option.
2. Create a custom object from Schema Builder.
   A. Click the **Elements** tab.
   B. Drag and drop **Object** from the palette and enter the object details.

Label:	`Customer Story`
Plural Label:	`Customer Stories`
Starts With:	**Consonant**
Object Name:	`Customer_Story`
Description:	`Used to track customer success stories.`
Context-Sensitive Help Setting:	**Open the standard Salesforce.com Help & Training window**
Record Name:	`Customer Story Name`

Data Type:	**Text**
Allow Reports:	**Select**
Allow Activities:	**Deselect**
Track Field History:	**Deselect**
In Development:	**Deployed**
Add Google Docs, Notes, and Attachments related list to default page layout:	**Deselect**

C. Click **Save**.
D. Rearrange the Customer Story object so it is visible in the canvas.

3. View the object detail page in the Setup menu.

   A. Click the **arrow** next to the gear icon on the Customer Story object to open the settings menu, then select **View Object**.

   What standard fields were automatically created?

   _____

   _____

   B. View the other sections of the Custom Object Definition Detail page.

   What other entities were automatically created?

   _____

   _____

4. Edit the object permissions on the Sales User and Marketing User profiles.

   A. Click **Setup**.
   B. Type `profiles` in the Quick Find textbox.
   C. Click **Profiles** under Manage Users.
   D. Click **Sales User**, which is a custom profile, in the Profile Name column.
   E. Click **Object Settings**.
   F. Click **Customer Stories**.
   G. Click **Edit**.
   H. Select **Enabled** for the Read, Create, and Edit permissions.
   I. Click **Save**.
   J. Click **Setup**.
   K. Type `profiles` in the Quick Find textbox.

L. Click **Profiles** under Manage Users.
M. Click **General Marketing User** in the Profile Name column.
N. Click **Object Settings**.
O. Click **Customer Stories**.
P. Click **Edit**.
Q. Select **Enabled** for the Read, Create, and Edit permissions.
R. Click **Save**.

5. View the organization-wide default setting for the new object.

   A. Click **Setup.**
   B. Type `sharing` in the Quick Find textbox.
   C. Click **Sharing Settings** under Security Controls.

   What is the organization-wide default setting for the Customer Story object?

   _____

2-2: Create Custom Fields

Goal:

Create custom fields on the Customer Stories object to track the story description, products, and installation time.

Tasks:

1. Using Schema Builder, add a text field to track the story description and view the field-level security.
2. Using the Setup menu, add the field to the page layout.
3. Using the Setup menu, add a multi-select picklist field to track products and a number field to track installation time.

Time:

15 minutes

Instructions:

1. Using Schema Builder, add a text field to track the story description and view the field-level security.
   A. Click **Setup**.
   B. Type `schema` in the Quick Find textbox.
   C. Click **Schema Builder** under Build.
   D. From the Elements tab, drag and drop **Text** onto the Customer Story object.
   E. Enter the custom field details.

Field Label:	`Story Description`
Field Name:	`Story_Description`
Description:	`Description of the customer story`
Help Text:	`Brief overview of the customer success story`
Length:	`255`
Default Value:	Leave blank
Required:	**Deselect**
Unique:	**Deselect**
External ID:	**Deselect**

F. Click **Save**.
G. Hover over the Story Description field, right click, and select **Manage Field Permissions**.

   Which users can view and edit this field?

   How would you restrict users from editing this field?

   How would you restrict users from viewing this field?

H. Click **Cancel**.

2. Using the Setup menu, add the field to the page layout.
   A. Click the **settings** arrow on the Customer Story object and select **View Page Layouts**.
   B. Click **Edit** next to Customer Story Layout.
   C. Drag and drop **Story Description** from the palette to below the Customer Story Name field (in the Information section).
   D. Click **Save**.
3. Using the Setup menu, add a multi-select picklist field to track products and a number field to track installation time.
   A. In the Custom Fields & Relationships section, click **New**.
   B. Select **Picklist (Multi-Select)** as the data type.
   C. Click **Next** and enter the details.

Field Label:	`Products`
Values	**Enter values, with each value separated by a new line**

Enter values, with each value separated by a new line.	`Desktops`
	`Laptops`
	`Printers`
	`Accessories`
	`Networking Equipment`
	`Servers`

Display values alphabetically, not in the order entered:	**Select**

Use first value as default value:	**Deselect**
# Visible Lines:	`4`
Field Name:	`Products`
Description:	`Products bought by the customer`
Help Text:	`Select one or more products`

D. Click **Next**.
E. Leave the field-level security as is and click **Next**.
F. Choose to automatically add the field to the page layout and click **Save and New**.
G. Select **Number** as the data type.
H. Click **Next** and enter the details.

Field Label:	`Installation Time (days)`
Length:	`3`
Decimal Places:	`0`
Field Name:	`Installation_Time`
Description:	`Installation time`
Help Text:	`Number of days from purchase to installation complete`
Required:	**Deselect**
Unique:	**Deselect**
External ID:	**Deselect**
Default Value:	Leave blank

I. Click **Next**.
J. Leave the field-level security as is and click **Next**.
K. Choose to automatically add the field to the page layout and click **Save**.
L. Close the Custom Object tab.
M. Refresh the browser tab to see the new fields in Schema Builder.

What are the differences between creating a field using the Setup menu and creating a

field using Schema Builder?

2-3: Create Relationship Fields

Goal:

Relate the Customer Stories object to the Account and Contact objects.

Tasks:

1. Create a Master-Detail relationship field and add a filter to limit the records available to users.
2. Create a Lookup relationship field and view the lookup options.
3. Add the Customer Stories related list to the account page layout.
4. Create a customer story record to verify the object was configured properly.

Time:

20 minutes

Instructions:

1. Create a Master-Detail relationship field and add a filter to limit the records available to users.
   A. Click **Setup**.
   B. Type `schema` in the Quick Find textbox.
   C. Click **Schema Builder** under Build.
   D. From the Elements tab, drag and drop **Master-Detail** onto the Customer Story object.
   E. Enter the custom field details.

Field Label:	`Account`
Field Name:	`Account`
Description:	`The Account whose success story this is.`
Help Text:	Leave blank
Related To:	`Account`
Child Relationship Name:	`Customer_Stories`
Sharing Setting:	**Read/Write: Allows users with at least Read/Write access to the Master record to create, edit, or delete related Detail records.**
Related List Label:	`Customer Stories`
Reparentable Master Detail:	**Deselect**

F. Click **Save**.
G. Hover over the Account field, right click, and select **View Field in New Window**.
H. Click **Edit**.
I. Click **Show Filter Settings** and enter the filter details.

Field:	`Account: Account Record Type`
Operator:	**equals**
Value / Field:	**Value**
Value:	`Customer`
Filter Type:	**Required. The user-entered value must match filter criteria.**
If it doesn't, display this error message on save:	`Select a customer account.`
Lookup Window Text:	Leave blank
Active:	**Select**

J. Click **Save**.
K. Click **Setup**.
L. Type `sharing` in the Quick Find textbox.
M. Click **Sharing Settings** under Security Controls.

What is the organization-wide default setting for the Customer story object?

N. Close the Sharing Settings tab.

2. Create a Lookup Relationship field and view the lookup options.

A. From the Elements tab, drag and drop **Lookup** onto the Customer Story object.
B. Enter the custom field details.

Field Label:	`Primary Contact`
Field Name:	`Primary_Contact`
Description:	`The Primary Contact of the Account whose success story this is.`

Help Text:	Leave blank
Related To:	`Contact`
Child Relationship Name:	`Customer_Stories`
Related List Label:	`Customer Stories`

C. Click **Save**.
D. Hover over the Primary Contact field, right click, and select **View Field in New Window**.
E. Click **Edit**.

What are the lookup options?

F. Click **Cancel**.
G. Click **Back to Customer Story**.
H. In the Page Layouts section, click **Edit** next to Customer Story Layout.
I. Drag and drop **Account** to below Currency.
J. From the palette, drag and drop **Primary Contact** to below Account.
K. Click **Save**.

3. Add the Customer Stories related list to the account page layout.

A. Click **Setup**.
B. Type `account` in the Quick Find textbox.
C. Click **Page Layouts** under Customize Accounts.
D. Click **Edit** next to Customer Account Layout.
E. From the palette, click **Related Lists**.
F. Drag and drop **Customer Stories** to below the Cases related list.
G. Click **Save**.
H. Click **Yes**.
I. Close the Account Page Layout tab.

4. Create a customer story record to verify the object was configured properly.

A. In Schema Builder, click **Close**.
B. Click the **Accounts** tab.
C. Click **New** to create a new test account.
D. Leave the Record Type of new record field set to Customer and click **Continue**.
E. Enter `Test Account` in the Account Name field.
F. Click **Save**.
G. From the Contacts related list, click **New Contact**.
H. Leave the Record Type of new record field set to Standard and click **Continue**.

I. Enter `Kate` in the First Name field and `Hanson` in the Last Name field.
J. Click **Save**.
K. Click **Test Account** to return to the account record.
L. From the Customer Stories related list, click **New Customer Story** and enter the details.

Customer Story Name:	`Test Account Customer Story`
Story Description:	`Major win at new customer against top competitor.`

Products Chosen:	`Laptops`
	`Networking Equipment`
	`Servers`

Installation Time (days):	`30`
Currency:	`USD – U.S. Dollar`
Account:	`Test Account`
Primary Contact:	`Kate Hanson`

M. Hover over the question mark icon next to the Installation Time (days) field to view the help text.
N. Click **Save**.
O. Click **Test Account** to return to the account record.
P. From the Contacts related list, click **Del** next to Kate Hanson.
Q. Click **OK**.
R. From the Customer Stories related list, right-click **Test Account Customer Story** and click **Open Link in New Tab**.

In the Customer Story tab, what happened to the value in the Primary Contact field? Why did this happen?

S. Return to the **Test Account** record tab.
T. Click **Delete** to delete the account record.
U. Click **OK**.
V. Return to the **Test Account Customer Story** tab and refresh the page.

What happened to the Test Account Customer Story record? Why did this happen?

3-1: Create a Formula Field

Goal:

Create a formula field on the Course Delivery object to calculate the end date.

Tasks:

1. Using the Setup menu, add a formula field to calculate the course delivery end date.
2. Test the formula field.

Time:

10 minutes

Instructions:

1. Using the Setup menu, add a formula field to calculate the course delivery end date.
   A. Click **Setup**.
   B. Type `objects` in the Quick Find textbox.
   C. Click **Objects** under Create.
   D. Click **Course Delivery**.
   E. In the Custom Fields & Relationships section, click **New**.
   F. Select **Formula** as the data type.
   G. Click **Next** and enter the details.

Field Label:	`End Date`
Field Name:	`End_Date`
Formula Return Type:	**Date**

   H. Click **Next** to enter the formula.
   I. In the Advanced Formula tab, create the following formula:

```
Start_Date__c  +  Course__r.Duration__c  - 1
```

   Note: Do not type the field names or the operators. Click the **Insert Field** and **Insert Operator** buttons instead.

   J. Click **Check Syntax**.
   K. Enter `End date of the course` in the Description field.
   L. Click **Next**.
   M. Leave the field-level security as is and click **Next**. Why is this field read only?
   N. Choose to automatically add the field to the page layout and click **Save**.
2. Test the formula field.
   A. From the Certification app, click the **Courses** tab. How many courses do you see?

B. Click **Go!** to view all course records. How many courses do you see?
C. Click **[401] Data Recovery**.

   What is the course duration?

D. Click **DELIVERY-00012**.

   What is the start date?

   What is the end date?

3-2: Create a Roll-Up Summary Field

Goal:

Create a roll-up summary field on the Course object to count the number of times a course was cancelled.

Tasks:

1. Using the Setup menu, add a roll-up summary field to count the number of cancellations.
2. Test the roll-up summary field.

Time:

10 minutes

Instructions:

1. Using the Setup menu, add a roll-up summary field to count the number of cancellations.
   A. Click **Setup**.
   B. Type `objects` in the Quick Find textbox.
   C. Click **Objects** under Create.
   D. Click **Course**.
   E. In the Custom Fields & Relationships section, click **New**.
   F. Select **Roll-Up Summary** as the data type.
   G. Click **Next** and enter the details.

Field Label:	`# of Courses Cancelled`
Field Name:	`Number_of_Courses_Cancelled`
Description:	`Number of courses cancelled.`

   H. Click **Next** and define the summary calculation.

Summarized Object:	**Course Deliveries**
Select Roll-Up Type:	**Count**
Filter Criteria:	**Only records meeting certain criteria should be included in the calculation**
Field:	`Status`
Operator:	**equals**

Value:	`Cancelled`

I. Click **Next**.
J. Leave the field-level security as is and click **Next**.
K. Choose to automatically add the field to the page layout and click **Save**.

2. Test the roll-up summary field.

A. From the Certification app, click the **Courses** tab.
B. Click **Go!** to view all course records.
C. Click **[102] AWCA Network**.

How many courses were cancelled?

How many courses were delivered?

3-3: Create a Formula Field that References Roll-Up Summary Fields

Goal:

Create a formula field on the Course object to calculate the cancellation rate.

Tasks:

1. Using the Setup menu, add a formula field to calculate the course cancellation rate.
2. Test the formula field.

Time:

10 minutes

Instructions:

1. Using the Setup menu, add a formula field to calculate the course cancellation rate.
   A. Click **Setup**.
   B. Type `objects` in the Quick Find textbox.
   C. Click **Objects** under Create.
   D. Click **Course**.
   E. In the Custom Fields & Relationships section, click **New**.
   F. Select **Formula** as the data type.
   G. Click **Next** and enter the details.

Field Label:	`Course Cancellation Rate`
Field Name:	`Course_Cancellation_Rate`
Formula Return Type:	**Percent**
Decimal Places:	**2**

   H. Click **Next** to enter the formula.
   I. Click the Advanced Formula tab.
   J. From the Functions drop down, choose IF and click **Insert Selected Function**.
   K. Replace `logical_test`, `value_if_true` and `value_if_false` to write a formula that looks like this:

```
IF((Number_of_Courses_Cancelled__c +
Number_of_Courses_Delivered__c)  > 0,
Number_of_Courses_Cancelled__c  /  (
Number_of_Courses_Cancelled__c  +
Number_of_Courses_Delivered__c  ), null)
```

Note: Do not type the field names or the operators. Click the **Insert Field** and **Insert**

Operator buttons instead

L. Click **Check Syntax**.
M. Enter `Cancellation rate of the course` in the Description field.
N. Click **Next**.
O. Leave the field-level security as is and click **Next**.
P. Choose to automatically add the field to the page layout and click **Save**.

2. Test the formula field.

   A. From the Certification app, click the **Courses** tab.
   B. Click **Go!** to view all course records.
   C. Click **[102] AWCA Network**.

   What is the course cancellation rate?

   _____

3-4: Understand Record Types

Goal:

Understand the capabilities of record types.

Tasks:

1. View account records for each record type.
2. Create an account record using the Service Vendor record type.

Time:

10 minutes

Instructions:

1. View account records for each record type.
   A. Click the **Accounts** tab.
   B. Click **Go!** to view all accounts.
   C. Click **ABC Labs**.
   D. Verify that the Industry field, the Opportunities related list, and the Cases related list are displayed on the page.
   E. Double click on the **Support Level** field to edit it. Verify that the picklist options are Silver, Gold, and Platinum.
   F. Click the **Accounts** tab.
   G. Click **Go!** to view all accounts.
   H. Click **Alveswood Technologies**.
   I. Verify that the Industry field, the Opportunities related list, and the Cases related list are not displayed on the page.
   J. Double click on the **Support Level** field to edit it. Verify that the picklist options are Standard Vendor and Premier Vendor.
2. Create an account record using the Service Vendor record type.
   A. Click the **Accounts** tab.
   B. Click **New**.
   C. From the Record Type of new record picklist, select **Service Vendor**.
   D. Click **Continue**.
   E. Enter `Test Service Vendor Account` and click **Save**.

4-1: Logging into a Sandbox

Goal:

Explore your sandbox.

Tasks:

1. Log in in to your sandbox.
2. Review the data.
3. Review user records.
4. Log out of your sandbox.

Time:

10 minutes

Instructions:

1. Log in in to your sandbox.
   A. Open a browser tab and browse to `https://test.salesforce.com`. This will open a page that allows you to log in to your sandbox.
   B. When you created your sandbox, you were asked to enter a name of `Dev`. Your username for the sandbox should be the same as your regular username, followed by a period and the sandbox name. Note: the sandbox name is not case sensitive.

      If your username is:

      `yourname@trg.org`

      then your sandbox username will be:

      `yourname@trg.org.dev`

   C. Type in the same password you used for your production training org.
   D. Click **Log in to Salesforce**.
   E. Once logged in, you should notice a black box in the top right-corner that displays, Sandbox: Dev.
2. Review the data.
   A. Click the **Courses** tab.
   B. Click **Go!**
   C. Notice that all the course data available in your training org is also in this sandbox instance.
3. Review user records.
   A. Click **Setup**.
   B. Type `users` in the Quick Find textbox.
   C. Click **Users** under Manage Users.

D. Click **Edit** for the first user listed.
E. Notice that the email address has been appended with `=example.com` to ensure that the system ignores these email addresses.

4. Log out of your sandbox.

A. In the Salesforce UI in the top right-hand corner, select ***your name*** **| Logout**.
B. Close the browser tab. Note that, unless otherwise stated, all other exercises should be completed in your production org.

4-2: See Apex in Action

Goal:

Execute Apex to create a new Contact record in the database.

Tasks:

1. Open the Developer Console.
2. Open Execute Anonymous.
3. Enter the code to execute.
4. Examine the logs.
5. Examine the result in the user interface.

Time:

15 minutes

Instructions:

1. Open the Developer Console.
   A. In the Salesforce UI in the top right-hand corner, select ***your name*** **| Developer Console**.
   B. Select **Debug | Change Log Levels**.
   C. Select Add/Change under DebugLevel Action.
   D. Set the Database, Apex Code and Profiling entries to **Finest** and System to **Fine**.
   E. Click **Done**.
   F. Click **Done**.
   G. In the Debug menu, deselect **Show My Current Logs Only**.
   H. Select **Debug | Perspective Manager**.
   I. Set the default perspective to **Log Only (Predefined)**, if it is not already the default.
2. Open Execute Anonymous.
   A. Select **Debug | Open Execute Anonymous Window**.
   B. Remove any existing code in the Enter Apex Code window.
   C. Select **Open Log** in the bottom right-hand corner.

   Note: You can expand the window by clicking the **Up** arrow in the top right corner.
3. Enter the code to execute.
   A. Paste the contents of ContactExecuteAnonymous.txt from the Exercises folder into the Execute Anonymous window.
   B. Click **Execute** in the bottom right-hand corner.
4. Examine the logs.
   A. When the Execution Log opens, select **Debug Only**. As a result, only debug statements should be visible. It should display only one log entry with the ID of the inserted record.
   B. Deselect **Debug Only**.
   C. In the input box next to Filter, type `DML` (this must be typed in all caps, as the filter box is

case-sensitive).

D. Notice the limits usage for DML statements and DML rows.
E. Select **File | Close All**.

5. Examine the result in the UI.

   A. In the Salesforce UI, click on the **Contacts** tab.
   B. Confirm that the record inserted using the code appears in the Recent Contacts section.

4-3: Create and Use an Apex Class

Goal:

Create an Apex class called `ContactManager` and define a method within the class to create a Contact record in the database.

Tasks:

1. Create a class and save it.
2. Invoke the class method using code in the Execute Anonymous window.
3. Examine the logs to see the invocation of the class.

Time:

10 minutes

Instructions:

1. Create a class and save it.
   A. In the Developer Console, select **File | New | Apex Class**.
   B. In the New Class dialog box, type `ContactManager` for the name of your new class.
   C. Click **OK**.
   D. Copy and paste the contents of ContactManagerClass.txt from the Exercises folder into the window, overwriting all existing text.
   E. Type `CTRL + S` to save the class.
   F. Note the API Version of the class.
2. Invoke the class method using code in the Execute Anonymous window.
   A. In the Developer Console, type `CTRL + E` to open the Execute Anonymous window.
   B. Remove any existing code in the Enter Apex Code window.
   C. Type this code into the Enter Apex Code window:

```
ID contactID = ContactManager.addContact('Rehman','Areil');
System.debug('Called from Execute Anonymous: ID is: ' +
contactID);
```

   D. Ensure that **Open Log** is selected and click **Execute**.
3. Examine the logs to see the invocation of the class.
   A. In the debug log for your execution, select **Debug Only**. As a result, only debug statements should be visible.
   B. Notice that the newly created record has an ID, which was returned by the method call.
   C. Deselect **Debug Only**.
   D. In the input box next to Filter, type `ContactManager`.
   E. Notice the invocation of the static `addContact()` method of the `ContactManager` class.
   F. Type `CTRL + ALT + /` to close all windows.

4-4: Observe the Effects of Versioning

Goal:

Change the version on an Apex class to see how it affects compilation.

Tasks:

1. Create a new class to test out versioning in Apex.
2. Discover what versions will compile FeedPost.
3. Discover what versions will compile FeedItem.

Time:

5 minutes

Instructions:

1. Create a new class to test out versioning in Apex.
   A. In the Salesforce UI in the top right-hand corner, select **Setup**.
   B. Type `apex` in the Quick Find textbox.
   C. Click **Apex Classes** under Develop.
   D. Click **New** to create a new class.
   E. Enter in the following code to create a public class called ChatterVersion:

```
public class ChatterVersion {

}
```

   F. Click **Save**.

2. Discover what versions will compile FeedPost.

   Background: When Chatter was first introduced, the original sObject that stored Chatter posts was called FeedPost. For example, inserting an sObject of type FeedPost would make a Chatter post to a user's own wall, another user's wall, a Chatter group, or on a record.

   A. Click **Edit** to modify the ChatterVersion class.
   B. Select the **Version Settings** tab.
   C. Change the version to **17.0**.
   D. Click **Quick Save**.
   E. Select the **Apex Class** tab.
   F. Update the class to have a variable called `fp` of type FeedPost:

```
public class ChatterVersion {
    FeedPost fp;
}
```

   G. Click **Quick Save**. You should see the error `Entity is not api accessible`.

H. Select the **Version Settings** tab.
I. Change the version to **18.0**.
J. Click **Save**. You should not see an error.

3. Discover what versions will compile FeedItem.

A. Click **Edit** to modify the ChatterVersion class.
B. Select the **Version Settings** tab.
C. Change the version to **22.0**.
D. Click **Quick Save**. You should see the error `Entity is not api accessible`.
E. Select the **Apex Class** tab.
F. Update the class to have a variable called `fi` of type FeedItem and comment out the line for the FeedPost variable fp:

```
public class ChatterVersion {
    // FeedPost fp;
    FeedItem fi;
}
```

G. Click **Save**. You should not see an error.

4-5: Take a Quick Tour of Apex

Goal:

Quickly learn some Apex fundamentals that will be familiar to you, and check out some simple differences.

Tasks:

1. Set up the classes.
2. Review the test classes.
3. Run the test.
4. View code coverage.

Time:

20 minutes

Instructions:

1. Set up the classes.
   A. In the Developer Console, select **File | New | Apex Class**.
   B. In the New Class dialog box, type `CourseManager`.
   C. Click **OK**.
   D. Copy and paste the contents of `CourseManagerClass.txt` from the Exercises folder into the window, overwriting all existing text.
   E. Complete the TODOs to complete the body of the method. The method should return a `Set` that contains the duplicates that were found in the two lists.
   F. Type `CTRL + S` to save the class.
2. Review the test classes.
   A. Click **File | New | Apex Class**.
   B. In the New Class dialog box, type `CourseManager_Test`.
   C. Click **OK**.
   D. Copy and paste the contents of CourseManager_Test.txt from the Exercises folder into the window, overwriting all existing text. Note the following in the code:
      i. Lines 14 and 15 create two lists of records. There will be 3 overlapping records with names: course3, course4, and course5.
      ii. Line 17 checks whether the code that will be moved to production actually fulfills the business requirement. If the assertion fails, an exception will be caused, which will fail the test.
   E. Type `CTRL + S` to save the class.
3. Run the test.
   A. From the Test menu, click **New Run**.
      i. Select the **CourseManager_Test** class.
      ii. Click **Add Selected**.

iii. Click **Run**.

B. Switch to the **Tests** tab in the bottom panel and locate the most recent TestRun (It is usually at the bottom of the list).
C. Expand the collapsed folder by clicking **+**.
D. Ensure that all your methods executed successfully. A green tick indicates success.

4. View code coverage.

A. Switch to the tab for the `CourseManager` class.
B. Click **Code Coverage:** (next to API Version).

Note: The lines that are covered appear in blue while those not covered appear in red.

C. Type `CTRL + ALT + /` to close all windows.

4-6: Examine Implicit Operations

Goal:

Examine the operations implicitly occurring when you perform a DML operation.

Tasks:

1. Create a Course Attendee record in the UI.
2. View the logs to see operations implicitly invoked due to the DML operation

Time:

10 minutes

Instructions:

1. Create a Course Attendee record in the UI.
   A. Leaving the Developer Console open, switch to the Salesforce UI in another window.
   B. Click the **Course Deliveries** tab.
   C. Click **Go!** if necessary.
   D. Click the link for the Course Delivery named **DELIVERY-00025**.
   E. Click **New Course Attendee**.
   F. Enter `Clara Petit` as the Student and select `Enrolled` as the Status.
   G. Click **Save**.
2. View the logs to see operations implicitly invoked due to the DML operation.
   A. Switch back to the Developer Console.

      Note: You may need to refresh the screen in Developer Console.
   B. Click the **Logs** tab in the bottom panel.
   C. Double-click on the most recent entry. (The Application column should read Browser and the Operation column should read /a02/e).
   D. In the Filter input box, type `Trigger` (the input is case-sensitive).
   E. Confirm that the `CourseAttendeeTrigger` was executed.
   F. Confirm that the `provideAccessLMS` method of the `CourseAttendeeHandler` class was executed.
   G. In the Filter input box, type `WF` (Overwrite any existing text. The input is case-sensitive).
   H. Confirm that the New Course Created workflow rule was evaluated. The ON_CREATE_ONLY entry in the Details column shows that it was not executed because it did not meet the criteria.
   I. Type `CTRL + ALT + /` to close all windows.

4-7: Profile Limits Using the Developer Console

Goal:
Investigate limits using the Developer Console.

Task:
Review the limit profiling information in the log.

Time:
5 minutes

Instructions:

1. Review the limit profiling information in the log.
   A. In the Developer Console, click the **Logs** tab in the bottom panel.
   B. Double-click the most recent entry. (The Application column should read Browser and the Operation column should read /a02/e).
   C. Select **Debug | Perspective Manager | All (Predefined)**.
   D. In the Execution Overview panel, notice the Save Order of the various operations.
   E. Click the **Timeline** tab in the Execution Overview panel. Notice the entries in the Category, Millis, and % columns.
   F. Switch to the **Limits** tab in the Execution Overview panel. Leave the tab open and type `STATEMENT` in the Filter input box of the Execution Log panel.
   G. Clicking the first entry in the Execution Log panel resets the Used so far column in the Limits tab. The Source panel jumps to the line of code that was executed at that point in time.
   H. Click the third-to-last entry in the Execution Log panel. The values for SOQL and SOQL_ROWS Limits Used so Far are updated. The Source panel jumps to the line of code that was executed at that point in time.
   I. Click the last entry in the Execution Log panel. All limits used are now accounted for.
   J. Type `CTRL + ALT + /` to close all windows.

4-8: Work with a Custom Object

Goal:

Work with the fields of the Course object.

Tasks:

1. Write code to insert a record for the Course object.
2. Check the Salesforce UI to ensure that the record was inserted successfully.

Time:

15 minutes

Instructions:

1. Write code to insert a record for the Course object.
   A. In the Developer Console, type `CTRL + E` to open the Execute Anonymous window.
   B. Remove any existing code in the Enter Apex Code window.
   C. Copy and paste the contents of `CreateCustomObjRecord.txt` from the Exercises folder into the window, overwriting all existing text.
   D. Complete the TODOs to create a new Course and insert it into the database.
   E. Select **Open Log** in the bottom right-hand corner.
   F. Click **Execute**.
   G. Ensure that your code inserts the record into the database.
   H. Type `CTRL + ALT + /` to close all windows.
2. Check the Salesforce UI to ensure that the record was inserted successfully.
   A. Switch to the Salesforce UI.
   B. Click the **Courses** tab.
   C. Click **Go**.
   D. Check that your record was created.

4-9: Use Record IDs to Access a Contact in the UI

Goal:

Explore the use of record IDs.

Tasks:

1. Query the ID and name fields for Contact records in the database.
2. Copy the ID from a returned record.
3. Use the ID to display the associated Contact page in the UI.

Time:

5 minutes

Instructions:

1. Query the ID and name fields for Contact records in the database.
   A. In the Developer Console, type `CTRL SHIFT + O` to open the Open Resource window.
   B. In this window, type in `Contact`, select **Contact.obj**, and click **Open**. As a result, you should see a list of the Contact sObject's fields and their datatypes.
   C. Click the **ID** field. Keep the **CTRL** key pressed and click **Name**.
   D. Click the **Query** button at the bottom. The focus will shift to the Query Editor window in the bottom panel.
   E. Click **Execute** at the bottom. The Query Results pane will display all Contact records.
2. Copy the ID from a returned record.
   A. Double-click the **ID** of any record, then select it.
   B. Copy and paste the ID into a text editor such as Notepad++.
   C. In Developer Console, type `CTRL + ALT + /` to close all windows.
   D. Close Developer Console.
3. Use the ID to display the associated Contact page in the UI.
   A. Switch to the Salesforce UI.
   B. Click the **Home** tab.
   C. In the URL, replace `home/home.jsp` with the ID of your record.
   D. Hit **Enter** to display the record.
   E. Determine the length of the ID. Is it 15 or 18 characters long?

5-1: Create and Run a Query in the Developer Console

Goal:
Retrieve Cases using the Query Editor in the Developer Console.

Task:
Run a query in the Query Editor in the Developer Console.

Time:
5 minutes

Instructions:

1. Run a query in the Query Editor in the Developer Console.
   A. In the Salesforce UI in the top right-hand corner, click **your name** and then click **Developer Console**.
   B. Open the Case object.
      i. In the Developer Console, type `CTRL + SHIFT + O` to open the Open Resource window.
      ii. In this window, type in `Case`, select **Case.obj**, and click **Open**. As a result, you should see a list of the Case sObject's fields and their datatypes.
      iii. Click the **Name** header to sort the fields by name.
   C. Create a query.
      i. Holding the **CTRL** key, select the fields listed in the following table from the Case.obj window.

CaseNumber
ClosedDate
Id
IsClosed
Status
Subject

      ii. Click **Query**. As a result, a query should populate in the query pane.
   D. Run a query.
      i. Ensure the Use Tooling API checkbox is not selected.
      ii. Click **Execute**. As a result, the Query Results window populates with the data retrieved by running the query.

E. Explore row-level options.
   i. Select a **single case** in the Query Results window.
   ii. Click **Open Detail Page**. As a result, the selected case's detail page should appear.
   iii. Notice other actions available from the Query Results window, such as creating, deleting, and updating sObject records.

F. Close all open by windows in the Developer Console.
G. Close the Developer Console.

5-2: Write a SOQL Query that Uses a WHERE Clause

Goal:

Retrieve Cases that meet specific criteria.

Tasks:

1. Run a query in the Query Editor in the Developer Console to retrieve all closed Cases.
2. Retrieve all Cases that do not have a specified type.
3. Retrieve all high-priority Cases that involve particular products.
4. Retrieve all Cases with a subject containing the word “printer.”

Time:

20 minutes

Instructions:

1. Retrieve all closed Cases.
   A. In the Salesforce UI in the top right-hand corner, click on **your name** and then click **Developer Console**.
   B. Open the Case object.
      i. In the Developer Console, type `CTRL + SHIFT + O` to open the Open Resource window.
      ii. In this window, type in `Case`, select **Case.obj**, and click **Open**. As a result, you should see a list of the Case sObject’s fields and their datatypes.
      iii. Click the **Name** header to sort the fields by name.
   C. Create a query.
      i. Holding the **CTRL** key, select the fields listed in the following table from the Case.obj window.

CaseNumber	**Priority**
Id	**Status**
IsClosed	**Subject**
Product_Category__c	**Type**

      ii. Click **Query**. As a result, a query should populate in the query pane.
   D. Run the query.
      i. Ensure the Use Tooling API checkbox is not selected.
      ii. Click **Execute**. As a result, the Query Results window populates with the data retrieved by running the query.

iii. In the Query Results window, click **Status** in the header row to sort the results by status. Notice that at least one Case's status is 'Closed'.

E. Write and **Execute** a query with a WHERE clause.

i. Modify the query in the query pane by manually adding a WHERE clause that specifies that only closed Cases should be retrieved. HINT: Review the WHERE clause operators in your Student Guide.
ii. Click **Execute**. As a result, the Query Results window should populate with the data retrieved by running the query. Verify the results of your query.

F. Close the Query Results window.

2. Retrieve all Cases that do not have a specified type.

A. In the Query Editor, modify the query in the query pane so it looks for Cases that have no specified type. HINT: Type is a Nullable field, so a Case without a Type will have a NULL Type.
B. Click **Execute**. As a result, the Query Results window populates with the data retrieved by running the query. Verify the results of your query.
C. Close the Query Results window.

3. Retrieve all high-priority Cases that involve particular products.

A. In the Query Editor, modify the query in the query pane so it finds Cases that meet both of the criteria listed in the following table. HINT: Remember a WHERE clause supports multiple filter criteria using the logical operators: AND, OR.

Field	Criteria
Priority	'High'
Product_Category__c	'Printers'

B. Click **Execute**. As a result, the Query Results window populates with the data retrieved by running the query. Verify the results of your query.
C. Close the Query Results window.

4. Retrieve all Cases with a subject containing the word "printer."

A. In the Query Editor, modify the query in the query pane so it finds Cases that have the word "printer" included somewhere in the subject. The word "printer" can appear at the beginning, middle, or end of the Case subject. HINT: Remember a WHERE clause supports the LIKE operator and wildcards when searching for string values.
B. Click **Execute**. As a result, the Query Results window populates with the data retrieved by running the query. Verify the results of your query.
C. Close all open by windows in the Developer Console.
D. Close the Developer Console.

5-3: Write and Execute a SOQL Query in Apex

Goal:

Print cases into the debug log.

Tasks:

1. Print a single case into the debug log.
2. Print multiple cases into the debug log.

Time:

10 minutes

Instructions:

1. Print a single case into the debug log. When debugging, it can be helpful to query an sObject by its Id (but never do this in production code as an sObject's Id changes depending on the environment; i.e. an sObject's Id in a Sandbox is not the same in Production).
   A. Prepare the lab.
      i. In the Salesforce UI in the top right-hand corner, click on **your name** and then click **Developer Console**.
      ii. In the Developer Console, type `CTRL + E` to open the Enter Apex Code window.
      iii. In this window, ensure the **Open Log** checkbox is selected.
      iv. Replace any existing text with the code for Task 1 in SOQLQueriesInApex.txt.
   B. Find a Case's Id.
      i. In the Salesforce UI, view any case.
      ii. From the URL, copy the ID of the case into a text editor.
   C. Use the Id to print the Case to the debug log.
      i. In the Developer Console's Enter Apex Code window, complete the TODOs.
      ii. Click **Execute**.
      iii. In the debug log for your execution, select the **Debug Only** checkbox. As a result, only debug statements should be visible.
      iv. Verify the results of your code using the debug log.
2. Print multiple cases into the debug log.
   A. In the Enter Apex Window, replace any existing text with the code for Task 2 in SOQLQueriesInApex.txt.
   B. Complete the TODOs.
   C. Click **Execute**.
   D. In the debug log for your execution, select the **Debug Only** checkbox. As a result, only debug statements should be visible. Verify the results of your code using the debug statements.
   E. Close all open by windows in the Developer Console.
   F. Close the Developer Console.

5-4: Write a Dynamic Query in Apex

Goal:

Retrieve cases that meet criteria specified at run time.

Tasks:

1. Use a bound variable to specify filter criteria.
2. Use Database.query() to repair the code from Task 1.
3. (Optional) Use a bound variable to query by record type.

Time:

10 minutes

Instructions:

1. Use a bound variable to specify filter criteria.
   A. Review considerations for working with Date Values in your Student Guide. In particular, notice that SOQL supports the use of date literals, such as LAST_N_DAYS in filter criteria for date values. You can find out more about date literals in the Apex documentation online.
   B. In the Salesforce UI in the top right-hand corner, click on **your name** and then click **Developer Console**.
   C. In the Developer Console, type `CTRL + E` to open the Enter Apex Code window.
   D. In this window, ensure the **Open Log** checkbox is selected.
   E. Replace any existing text with the code from Task 1 in SOQLDynamicQueryInApex.txt. Note there are no TODOs in the code for this task.
   F. Click **Execute**. Note that this causes an error because the bind variable is not supported with LAST_N_DAYS in a bracketed SOQL query.
   G. Click **OK** to continue.
2. Use Database.query() to repair the code from Task 1.
   A. In the Enter Apex Code window, replace any existing text with the code from Task 2 in SOQLDynamicQueryInApex.txt.
   B. Complete the TODOs.
   C. Click **Execute**.
   D. In the debug log for your execution, select the **Debug Only** checkbox. As a result, only debug statements should be visible. Verify that cases were retrieved.
   E. Close all open windows by typing `CTRL + ALT + /`
3. (Optional) Use a bound variable to query by record type.
   A. Replace any existing text with the code from Task 3 in SOQLDynamicQueryInApex.txt.
   B. Complete the TODOs.
   C. Click **Execute**.
   D. In the debug log for your execution, select the **Debug Only** checkbox. As a result, only debug statements should be visible. Verify that cases were retrieved.

E. Close all open windows by typing `CTRL + ALT + /`
F. Close the Developer Console.

6-1: Write and Test Child-to-Parent Relationship Queries

Goal:

Write child-to-parent relationship queries that explore relationships among sObjects in the Certification application.

Tasks:

1. Select all Contacts and their related Accounts.
2. Select Courses that have related Certifications.

Time:

20 minutes

Instructions:

1. Select all Contacts and their related Accounts.

   A. View the API names of fields in sObjects.

      i. In the Salesforce UI in the top right-hand corner, click on **your name** and then click **Developer Console**.
      ii. In the Developer Console, navigate to **File | Open Resource**.
      iii. In the Open Resource window, type `Contact.obj`. Click **Open** to open the highlighted sObject and view its fieldnames and their data types.
      iv. Using the steps you just completed as a guide, open the Account object (`Account.obj`).

   B. Use the Query Editor to write a query that selects all Contacts and their related Accounts.

      i. Select the **Query Editor** tab, and erase any text in the query pane.
      ii. Write a single query that selects all Contacts. You should select the following fields: the Name field from Contact and the related Name field from Account. The FROM clause of the query should be: `FROM Contact`.

   C. Run the query.

      i. Ensure that the **Use Tooling API** checkbox is not selected.
      ii. Click **Execute**. As a result, the Query Results window should populate with a list of the Contacts' names, along with the related Account name.
      iii. When you're done viewing the query results, click on the **Query Results** pane and type `CTRL + /`. As a result, the Query Results pane should close.

   D. Move the query into Apex, using a SOQL `for` Loop.

      i. In the Developer Console, select the **Perspective manager** from the Debug menu. Set the default perspective to **Log Only (Predefined)**, if it is not already the default.
      ii. Type `CTRL + E` to open the Enter Apex Code window. You can expand the window by clicking the **Up** arrow in the top right corner.
      iii. Remove any existing code in the window.

iv. Write anonymous Apex code that uses a SOQL `for` loop, which uses an iterator whose data type is `List<Contact>`, to print to the debug log the names of each Contact and related Account. Use the same SOQL query that you created in the previous step. HINT: If you have trouble remembering how to write SOQL for loops, consult module 5 in your Student Guide.

An example of what you need to print to the log follows:

```
Erica Neumann is related to the following Account:
IntronCorporation
```

v. Ensure that the **Open Log** checkbox is selected and click **Execute**.
vi. In the debug log for your execution, select the **Debug Only** checkbox. As a result, only debug statements should be visible.
vii. When you're done viewing the debug log results, click on the **Log Results** pane and type `CTRL + /`. As a result, the Log Results pane should close.

2. Select Courses that have related Certifications.

A. View the API names of fields in sObjects.

i. Using Step 1.A in this exercise as your guide, open `Course__c.obj` to view the Course__c sObject's fieldnames and their data types.
ii. Using Step 1.A in this exercise as your guide, open `Certification__c.obj` to view the Certification__c sObject's fieldnames and their data types

B. Use the Query Editor to write and test a query that selects all Courses and their related Certifications.

i. In the Developer Console, select the **Query Editor** tab and erase any text in the query pane.
ii. Write a query that selects only Courses with related Certifications. You should select the following fields: the Name from Course and the Name field from Certification. The FROM clause of the query should be: `FROM Course__c`.

C. Run the query.

i. Ensure that the **Use Tooling API** checkbox is not selected.
ii. Click **Execute**. As a result, the Query Results window should populate with a list of the Course names, along with the related Certification name.
iii. Close all open windows in the Developer Console.
iv. Close the Developer Console.

6-2: Query Accounts and Related Contacts

Goal:

Write parent-to-child relationship queries that explore relationships among sObjects in the Certification application.

Tasks:

1. Select all Accounts and their related Contacts.
2. Select all Certifications that have a related Course.

Time:

10 minutes

Instructions:

1. Select all Accounts and their related Contacts.
   A. In the Salesforce UI in the top right-hand corner, click on **your name** and then click **Developer Console**.
   B. Select the **Query Editor** tab, and erase any text in the query pane.
   C. Write a single query that selects all Accounts and displays their names, along with the last names of any related Contacts. The FROM clause of the query should be: `FROM Account`.
   D. Run the query.
      i. Ensure that the **Use Tooling API** checkbox is not selected.
      ii. Click **Execute**. As a result, the Query Results window should populate with a list of the Account names, along with the last names of any related Contacts.
      iii. When you're done viewing the query results, click on the **Query Results** pane and type `CTRL + /`. As a result, the Query Results pane should close.
2. Select all Certifications that have a related Course.
   A. In the Salesforce UI in the top right-hand corner, click on **your name** and then click **Developer Console**.
   B. Select the **Query Editor** tab, and erase any text in the query pane.
   C. Write a single query that selects only Certifications with an associated Course. Be sure to also select the Name of the associated Course. The FROM clause of the query should be: `FROM Certification__c`. Hint: You will need to use two nested queries.
   D. Run the query.
      i. Ensure that the **Use Tooling API** checkbox is not selected.
      ii. Click **Execute**. As a result, the Query Results window should populate with a list of the Certification names, along with the names of related Courses.
      iii. Close all open windows in the Developer Console.
      iv. Close the Developer Console.

7-1: Execute DML Commands

Goal:

Create and save Contacts.

Tasks:

1. Write Apex code to insert Contacts using a standalone insert statement.
2. Write Apex code to insert Contacts using a `Database` class method.
3. Test your code.

Time:

25 minutes

Instructions:

1. Write Apex code to insert Contacts using a standalone insert statement.
   A. Open the Developer Console.
      i. In the Salesforce UI in the top right-hand corner, select ***your name*** **| Developer Console**.
      ii. Click **Debug** and deselect **Show My Current Logs Only**.
      iii. Select the **Perspective Manager** from the Debug menu. Set the default perspective to **Log Only (Predefined)**, if it is not already the default.
   B. Create a class and save it.
      i. Click **File | New | Apex Class**.
      ii. In the New Class dialog box, type `ContactsDML` for the name of your new Class.
      iii. Click **OK**.
      iv. Copy and paste the contents of ContactsDML.txt from the Exercises folder into the window, overwriting all existing text.
      v. Complete the TODOs in the `standaloneDML` method.
      vi. Type `CTRL + S` to save the class.
2. Write Apex code to insert Contacts using a `Database` class method.
   A. Complete the TODOs in the `databaseMethodDML` and `databaseMethodDMLAllOrNone` methods.
   B. Type `CTRL + S` to save the class.
3. Test your code.
   A. Create a Test class.
      i. Click **File | New | Apex Class**.
      ii. In the New Class dialog box, type `ContactsDML_Test`.
      iii. Click **OK**.
      iv. Copy and paste the contents of ContactsDMLTest.txt from the Exercises folder into the window, overwriting all existing text.

v. Complete the TODOs in the `standaloneDMLTest`, `databaseMethodDMLTest` and `databaseMethodDMLAllOrNoneTest` methods.
vi. Type `CTRL + S` to save the class.

B. Run the test.

i. From the Test menu, click **New Run**.
ii. Click `ContactsDML_Test` and click on **Add Selected** button.
iii. Click the **Run** button.
iv. Switch to the **Tests** tab in the bottom panel.
v. Expand the collapsed folder by clicking on **+**.
vi. Ensure that all your methods executed successfully. A green tick indicates success.
vii. Close all the open windows in the Developer Console.
viii. Close the Developer Console.

7-2: Handle DML Errors and Exceptions

Goal:

Handle errors when inserting Contacts.

Tasks:

1. Print the list of reasons why Contacts could not be inserted into the database.
2. Test your code.

Time:

15 minutes

Instructions:

1. Print the list of reasons why Contacts could not be inserted into the database
   A. In the Salesforce UI in the top right-hand corner, select ***your name*** **| Developer Console**. The Developer Console should open.
   B. Type `CTRL + O`. Choose **Classes** for the Entity Type. Double-click the **ContactsDML** class.
   C. Copy and paste the contents of ContactsDMLException.txt from the Exercises folder into the window, overwriting all existing text.
   D. Complete the TODOs in the `exceptionsDML` method.
   E. Type `CTRL + S` to save the class.
2. Test your code.
   A. Complete the Test class.
      i. Type `CTRL + O`. Choose **Classes** for the Entity Type. Double-click the **ContactsDML_Test** class.
      ii. Copy and paste the contents of ContactsDMLExceptionTest.txt from the Exercises folder into the window, overwriting all existing text.
      iii. Complete the TODO in the `exceptionsDMLTest` method.
      iv. Type `CTRL + S` to save the class.
   B. Run the test.
      i. From the Test menu, click **New Run**.
      ii. Move the `ContactsDML_Test` class to the Selected Test Classes list by clicking on **>**.
      iii. Click the **Run** button.
      iv. Ensure that all your methods executed successfully. A green tick indicates success.
      v. Observe the outputs of the `exceptionsDML` method in the debug log.
      vi. Close all the open windows in the Developer Console.
      vii. Close the Developer Console.

8-1: Define a Trigger

Goal:
Define a trigger on the Course_Delivery__c sObject.

Task:
Define a trigger.

Time:
5 minutes

Instructions:

1. Define a trigger.
   A. In the Salesforce UI in the top right-hand corner, select ***your name*** **| Developer Console**. The Developer Console should open.
   B. In the Developer Console, click **File | New | Apex Trigger**. In the dialogue that pops up, type in `CourseDeliveryTrigger` and select **Course_Delivery__c** as the sObject. Click **Submit**. A trigger definition should be created for you.
   C. Modify the trigger definition so that it runs before updates as well.
   D. Type `CTRL + S` to save the trigger.

8-2: Define the Trigger's Business Logic

Goal:

Define the business logic of a trigger that only allows a Course Delivery to be saved if is not scheduled to start on a holiday.

Tasks:

1. Create a Holiday.
2. Create a trigger.
3. Test the trigger's logic.

Time:

15 minutes

Instructions:

1. Create a Holiday.
   A. Click **Setup**.
   B. Type `holidays` in the Quick Find textbox.
   C. Click **Holidays** under Company Profile.
   D. Create a new Holiday listing for New Year's Eve. It should occur on December 31 of the current year, and it should not repeat.
2. Create a trigger.
   A. In the Developer Console, ensure that CourseDeliveryTrigger is open.
   B. Copy and paste the contents of CourseDeliveryTrigger.txt from the Exercises folder into the window, overwriting all existing text.
   C. Complete the TODOs.
   D. Save the trigger. Fix any syntax errors before continuing.
   E. Close all the open windows in the Developer Console.
   F. Close the Developer Console.
3. Test the trigger's logic.
   A. In Salesforce, create and save a new Course Delivery with a start date of December 31 of the current year. Because its Start Date falls on a holiday, you should not be able to save the Course Delivery.
   B. In Salesforce, create and save a new Course Delivery with a start date of January 1 of the upcoming year. Because its Start Date does not fall on a known holiday, you should be able to save the Course Delivery. Verify that the Course Delivery is saved (you can do this programmatically using SOQL or declaratively, by finding the Course Delivery in the UI).
   C. In Salesforce, find the Course Delivery you create in step 3.B. Update it so that its start date is on December 31 of the current year. You should not be able to update this Course Delivery's start date to this day because it is a Holiday.

9-1: Define an Apex Class

Goal:

Make a trigger easy to read and maintain by creating a helper class for it.

Tasks:

1. Create an Apex class.
2. Invoke the class from the trigger.
3. Test the trigger.

Time:

15 minutes

Instructions:

1. Create an Apex class.
   A. Open the Developer Console.
      i. In the Salesforce UI in the top right-hand corner, select ***your name*** **| Developer Console**.
      ii. Click **Debug** and deselect **Show My Current Logs Only**.
      iii. Select the **Perspective Manager** from the Debug menu. Set the default perspective to **Log Only (Predefined)**, if it is not already the default.
   B. Create a class and save it.
      i. Click **File | New | Apex Class**.
      ii. In the New Class dialog box, type `CourseDeliveryTriggerHandler` for the name of your new Class.
      iii. Click **OK**.
      iv. Copy and paste the contents of CourseDeliveryTriggerHandler.txt from the Exercises folder into the window, overwriting all existing text.
      v. Complete the TODOs, as required.
      vi. Type `CTRL + S` to save the class.
2. Invoke the class from the trigger.
   A. Type `CTRL + SHIFT +O` to open the Open Resource Window
   B. At the top of the Open Resource window, type in `CourseDeliveryTrigger.` Move the blue selection highlight bar to CourseDeliveryTrigger.apxt. If you do not have this trigger, please complete labs 8.1 and 8.2, and then restart this exercise at step 2.
   C. Click **Open**.
   D. Copy and paste the contents of CourseDeliveryTriggerRefactored.txt from the Exercises folder into the window, overwriting all existing text.
   E. Complete the TODOs as required.
   F. Type `CTRL + S` to save your changes.

3. Test your code.
   A. Create a Test class.
      i. Click **File | New | Apex Class**.
      ii. In the New Class dialog box, type `CourseDeliveryTriggerRefactored_Test`.
      iii. Click **OK**.
      iv. Copy and paste the contents of CourseDeliveryTriggerRefactored_Test.txt from the Exercises folder into the window, overwriting all existing text.
      v. Type `CTRL + S` to save the class.
   B. Run the test.
      i. From the Test menu, click **New Run**.
      ii. Select CourseDeliveryTriggerRefactored_Test.
      iii. Click **Add Selected**.
      iv. Click the **Run** button.
      v. Switch to the **Tests** tab in the bottom panel.
      vi. Expand the collapsed folder by clicking **+**.
      vii. Ensure that all your methods executed successfully. A green tick indicates success.
      viii. Close all the open windows in the Developer Console.
      ix. Close the Developer Console.

10-1: Explore the Implicit Firing of Triggers

Goal:

Determine the actions that occur when a Course record is saved.

Tasks:

1. Open the Developer Console.
2. Update the status of a Course record to Retired in the UI.
3. Review the results in the Developer Console.

Time:

10 minutes

Instructions:

Note: If you do not have a working copy of `CourseDeliveryTrigger`:

Create or replace the trigger `CourseDeliveryTrigger` in your org with the version 8.2-CourseDeliveryTrigger.txt from the Solutions folder.

1. Open the Developer Console.
   A. In the Salesforce UI in the top right-hand corner, select ***your name*** **| Developer Console**.
   B. In the Debug menu, deselect **Show My Current Logs Only**.
2. Update the status of a Course record to Retired in the UI.
   A. Leaving the Developer Console open, switch to the Salesforce UI in another window.
   B. Click the **Courses** tab.
   C. Click **Go!**
   D. Click **Edit** for the course [401] Data Recovery.
   E. Change the Status field to **Retired**.
   F. Click **Save**.
3. Examine the results in the Developer Console.
   A. Switch back to the Developer Console window.
   B. Click the **Logs** tab in the bottom panel.
   C. Double-click the most recent log entry.
   D. Select **Debug | Perspective Manager**.
   E. Double-click **All (Predefined)**.
   F. In the Filter input box of the Execution Log panel, type `START` (Please note that the input box is case-sensitive).
   G. Notice that twice an `AfterUpdate` event was fired for `CourseTrigger`, followed by a `BeforeUpdate` event for `CourseDeliveryTrigger`, followed by the execution of a workflow rule.
   H. Looking at the Save Order tab in the Execution Overview pane also shows the triggers firing, as well as a workflow rule.
   I. Click the **Executed Units** tab in the Execution Overview panel and click the entry for the

`checkStatus` method.

J. Notice the various operations called during the execution of this method in the Execution Log panel.
K. Close the Developer Console.

10-2: View the Events that Occur During a Rollback

Goal:

Determine the events that occur when an update action is rolled back.

Tasks:

1. Open the Developer Console.
2. Update the status of a Course record to “Retired” in the UI.
3. Review the results in the Developer Console and the UI.

Time:

10 minutes

Instructions:

1. Open the Developer Console.
   A. In the Salesforce UI in the top right-hand corner, select ***your name*** **| Developer Console**.
      Note: You may need to refresh the screen if the Developer Console was already open.
   B. Select **Debug | Change Log Levels**.
   C. In the General Trace Settings for You section, click **Add/Change** to bring up the Change Log Levels window.
   D. Set the Database, Apex Code and Profiling entries to **Finest** and System to **Fine**.
   E. Click **Done**.
   F. Click **Done**.
   G. In the Debug menu, deselect **Show My Current Logs Only**.
2. Update the status of a Course record to Retired in the UI.
   A. Leaving the Developer Console open, switch to the Salesforce UI in another window.
   B. Click the **Courses** tab.
   C. Click **Go!**
   D. Click **Edit** for the course [102] AWCA Network.
   E. Change the Status field to **Retired**.
   F. Click **Save**. You should see the error `"Course has enrolled students"` in the UI.
3. Examine the results in the Developer Console and the UI.
   A. Switch back to the Developer Console window.
   B. Click the **Logs** tab in the bottom panel.
   C. Double-click the latest log entry.
   D. Select **Debug | Switch Persepctive | All (Predefined)**.
   E. Notice in the Execution Overview in the Save Order tab we see the same triggers and workflow that fired in the last exercise are firing again (although more often).
   F. In the Filter input box of the Execution Log panel, type `Retired` (Please note that the input box is case-sensitive).
   G. Double-click on the first entry that lists `triggerNew`.

i. In the popup window, validate that the value of `Status__c` field is Retired.
ii. Click **OK**. Note that this value was never committed to the database.

H. Click the **Executed Units** tab in the Execution Overview panel and click the entry for the `addError` method. The entry in the log confirms that the `addError` method was executed.
I. Close the Developer Console.
J. Go back to the UI and click **Cancel**.
K. Click the Course named [102] AWCA Network.
L. Notice that the 5 Course Delivery objects have not changed status and the Course still has the status Active, showing the save was rolled back

10-3: See the Save Order of Execution in Action

Goal:

When saving a record, see and discuss the events that occur.

Tasks:

1. Open the Developer Console.
2. Create a Course Attendee record in the UI.
3. Using the log file generated by the interaction with the UI, answer the questions.
4. Run the test for the `CourseAttendeeTrigger`.
5. Using the log file generated by running the test class, answer the questions.

Time:

10 minutes

Instructions:

1. Open the Developer Console.
   A. In the Salesforce UI in the top right-hand corner, select ***your name*** **| Developer Console**.

      Note: You may need to refresh the screen if the Developer Console was already open.
   B. Select **Debug | Change Log Levels**.
   C. Set the Apex Code entry to **Debug** and Profiling to **Fine**.
   D. Click **Done**.
   E. In the Debug menu, deselect **Show My Current Logs Only**.
2. Create a Course Attendee record in the UI.
   A. Leaving the Developer Console open, switch to the Salesforce UI in another window.
   B. Click the **Course Deliveries** tab.
   C. Click **Go!** if necessary.
   D. Click a **Course Delivery Number** where the Status is Scheduled.
   E. Click **New Course Attendee**.
   F. Provide values for the Student and Status fields.
   G. Click **Save**.
3. Using the log file generated by the interaction with the UI, answer the questions.
   A. Return to the Developer Console.
   B. Click the **Logs** tab in the bottom panel.
   C. Double-click the **most recent log file** to open it.
   D. Use the log file to answer the questions. You may also want to look at the actual code or configuration in your org as well.
      i. What DML operation occurred?

ii. What type of Trigger is the `CourseAttendeeTrigger` (before/after, insert/update/delete)?

---

4. Run the test for the CourseAttendeeTrigger trigger.
   A. Select **File | Open | Classes | CourseAttendeeTrigger_Test**.
   B. Click **Open**.
   C. Click **Run Test**.
   D. Look in the Test pane to verify the test executed successfully.
5. Using the log file generated by running the test class, answer the questions.
   A. Click the **Logs** tab in the bottom panel.
   B. Double-click the **most recent log file** to open it.
   C. Using the logs and your understanding of the Save Order of Execution and Triggers, answer these questions:
      i. How many times does the `CourseAttendeeTrigger` fire?

      ---

      ii. What is causing the `CourseAttendeeTrigger` to fire multiple times?

      ---

      iii. How often is the workflow firing? What determines the number of times it fires?

11-1: Make Test Data Available to Test Methods

Goal:

Create test data to test the Certification application.

Tasks:

1. Create test data using a Static Resource.
2. Create test data programmatically.

Time:

10 minutes

Instructions:

1. Create test data using a Static Resource.
   A. In the Salesforce UI in the top right-hand corner, select ***your name*** **| Developer Console**. The Developer Console should open.
   B. In the Developer Console, click **File | New | Static Resource.** In the dialogue that pops up, type in `Test_Holidays` as the Name and select **"text/plain"** as the MIME type. Click **Submit**, and a blank text editor should appear in the Developer Console.
   C. Copy and paste the contents of the HolidaysForStaticResource.txt. Notice that the first line in a Static Resource definition has field names; subsequent lines contain data.
   D. Save the file.
   E. Type `CTRL + ALT + /` to close all open windows in the console.
2. Create test data programmatically.
   A. Create a test class to test inserting Courses and Course Deliveries.
      i. Click **File | New | Apex Class**.
      ii. In the New Class dialog box, type `CourseDeliveryTriggerHandler_Test2`.
      iii. Click **OK**.
      iv. Copy and paste the contents of CourseDeliveryTriggerHandler_Test2.txt from the Exercises folder into the window, overwriting all existing text.
      v. Complete the TODOs.
      vi. Save the class. Fix any syntax errors before continuing. Notice that when you save a test class, an additional Run Test button appears on the top right hand corner of the class editor.
   B. Run the test.
      i. Click **Run Test**.
      ii. Verify that your test run log contains debug statements that display the test data.
      iii. Close all the open windows in the Developer Console.
      iv. Close the Developer Console.

11-2: Write and Run an Apex Test

Goal:

Determine if the Certification application properly prevents a course delivery from being scheduled on a holiday.

Tasks:

1. Write a test that tests the business logic of a trigger and a class.
2. Run the test.

Time:

10 minutes

Instructions:

1. Write a test that tests the business logic of a trigger and a class.
   A. In the Salesforce UI in the top right-hand corner, select ***your name*** **| Developer Console**. The Developer Console should open.
   B. Type `CTRL + SHIFT + O`. Type `CourseDelivery` and then open the CourseDeliveryTriggerHandler_Test2 class.
   C. Copy and paste the contents of CourseDeliveryTriggerHandler_Test2_TestMethods.txt from the Exercises folder into the window, overwriting all existing text.
   D. Complete the TODOs.
   E. Save the class. Fix any syntax errors before continuing.
2. Run the test.
   A. Click **Run Test**.
   B. Verify that your test run log contains debug statements that verify the business logic executed as expected.
   C. Close all the open windows in the Developer Console.
   D. Close the Developer Console.

12-1: Explore Code Coverage

Goal:

Explore code coverage using the Developer Console.

Tasks:

1. Prepare the lab.
2. Run the test class.
3. View code coverage.

Time:

10 minutes

Instructions:

1. Prepare the lab.
   A. This lab requires the CourseDeliveryTrigger. If you haven't already completed the exercises in the Trigger Essentials module, please do so now.
   B. This lab requires the static resource Test_Holidays and the test class CourseDeliveryTriggerHandler_Test2. If you haven't already completed the exercises in the Testing Essentials module, please do so now.
2. Run the test class.
   A. Click **Test | Clear Test Data** to clear any existing test statistics.
   B. Click **Test | New run**.
   C. In the Select Tests dialog box, select **CourseDeliveryTriggerHandler_Test2** from the Test Classes list. Click on the **Add Selected** button to move it to the Selected Test Classes list.
   D. Click **Run**.
3. View code coverage.
   A. If the CourseDeliveryTriggerHandler is not open already, open it by typing `CTRL + SHIFT + O`, selecting it from the provided list, and then selecting **Open**.
   B. In the top left-hand corner of the CourseDeliveryTriggerHandler window, from the Code Coverage drop-down box, select **CourseDeliveryTriggerHandler_Test2.insertCourseDeliverySuccess**. This should cause the red and blue coverage highlighting to display. Notice that the code that was not covered by the test class (i.e., highlighted in red) is related to 'bad data' – i.e. a Course Deliveries that is scheduled on a Holiday.
   C. In the top left-hand corner of the CourseDeliveryTriggerHandler window, from the Code Coverage drop-down box, select **None** to remove the code highlighting.

13-1: Refactor a Trigger to Avoid SOQL Limits

Goal:

Refactor the code for the Course Trigger to avoid the “Too many SOQL queries” error when running a test with several records.

Tasks:

1. Examine the current Course trigger.
2. Test the trigger with a single record in the Salesforce user interface.
3. Test the trigger with several records using unit test code.
4. Refactor the code to avoid a SOQL error.

Time:

15 minutes

Instructions:

1. Examine the current Course trigger.
   A. In the Salesforce user interface, in the top right-hand corner, click on **your name** and then click **Developer Console**.
   B. In the Developer Console, click **File | Open**.
   C. Select **Classes** as the Entity Type and **CourseTriggerHandler** as the Entity and click **Open**.
   D. Examine the code and try to determine what would cause it to receive a “Too many SOQL queries” error if run with multiple records.

2. Test the trigger with a single record in the Salesforce user interface.
   A. In the Salesforce user interface, navigate to the **Courses** tab and click **Go**.
   B. Click **Edit** next to one of the active Courses**.**
   C. Change the status to `Retired` and click **Save**.
   D. If the Course does not have any enrolled Attendees, then it should allow you to change the status. If it does have enrolled attendees, then you will get a message telling you that the “Course has enrolled students.” Either outcome indicates that the trigger was able to complete execution.

3. Test the trigger with several records using unit test code.
   A. In the Developer Console, click **File | Open**.
   B. Select **Classes** as the Entity Type and **CourseTrigger_Test** as the Entity and click **Open**.
   C. Click **Run Test**.
   D. Switch to the **Tests** tab and wait for the test to complete. You should see it fail with a red x, and once you do, double-click the entry to bring up the results. Notice what error appears in the Errors column.

4. Refactor the code to avoid a SOQL error.
   A. In the Developer Console, return to the tab that displays the code for the CourseTriggerHandler.
   B. Delete the entire contents of the current code and copy the contents from the CourseTriggerHandler.txt file located in the Exercises folder into the content window.
   C. Complete the TODO sections to refactor the code.
   D. Click **Save**.
   E. Select the **CourseTrigger_Test** tab.
   F. Click **Run Test**.
   G. Switch to the **Tests** tab and wait for the test to complete. You should see it succeed with a green checkmark, and once you do, double-click the entry to bring up the results. Verify that your code runs correctly.
   H. Close the Developer Console.

13-2: Refactor a Trigger to Avoid DML Limits

Goal:

Refactor the code for the Course Trigger to avoid the "Too many DML rows" error when running a test with several records.

Tasks:

1. Examine the current Course trigger.
2. Refactor the code to avoid a DML error.
3. Re-test the trigger with several records using unit test code.

Time:

15 minutes

Instructions:

1. Examine the current Course trigger.
   A. In the Salesforce user interface, in the top right-hand corner, click on **your name** and then click **Developer Console**.
   B. In the Developer Console, click **File | Open**.
   C. Select **Classes** as the Entity Type and **CourseTriggerHandler** as the Entity and click **Open**.
   D. Examine the code and try to determine what would cause it to receive an error if run with multiple records.

2. Refactor the code to avoid a DML error.
   A. Delete the entire contents of CourseTriggerHandler and copy the contents from the CourseTriggerHandler.txt file located in the Exercises folder into the content window.
   B. Complete the TODO sections to refactor the code.
   C. Click **Save**.

3. Re-test the trigger with several records using unit test code.
   A. In the Developer Console, return to the tab for the CourseTrigger_Test.
   B. Click **Run Test**.
   C. Switch to the **Tests** tab and wait for the test to complete. You should now see it pass with a green checkmark.
   D. Type `CTRL + ALT + /` to close all open windows.
   E. Close the Developer Console.

14-1: Create a Formula Field to Eliminate a Query

Goal:
Create a formula field so that the certification Id is available on the Certification Attempt object.

Task:
Create a hidden formula field on the Certification Attempt object.

Time:
5 minutes

Instructions:

1. Create a hidden formula field on the Certification Attempt object.
   A. In the Salesforce UI in the top right-hand corner, select **Setup**.
   B. Type `objects` in the Quick Find textbox.
   C. Click **Objects** under Create.
   D. In Custom Objects, click the link for **Certification Attempt**.
   E. Under Custom Fields & Relationships, click **New**.
   F. Under Step 1: Choose the field type.
      i. Data Type: **Formula**
      ii. Click **Next**.
   G. Under Step 2: Choose output type.
      i. Field Label: `Certification Id`
      ii. Field Name: `Certification_Id`
      iii. Formula Return Type: **Text**
      iv. Click **Next**.
   H. Under Step 3: Enter Formula.
      i. Certification Id (Text):
      `CASESAFEID(Certification_Element__r.Certification__c)`
      ii. Click **Check Syntax**. Verify that no syntax errors occur.
      iii. Click **Next**.
   I. Under Step 4: Establish field-level security.
      i. Click **Next**.
   J. Under Step 5: Add to Page Layouts.
      i. Click the checkbox next to **Add Field** to deselect all page layouts
      ii. Click **Save**.

14-2: Create Fields for Counting Certification Elements

Goal:

Implement fields that will make the number of Certification Elements associated with a Certification available on the Certification Attempt Element.

Tasks:

1. Create a roll-up summary field on the Certification object.
2. Create a hidden formula field on the Certification Attempt object.

Time:

5 minutes

Instructions:

1. Create a roll-up summary field on the Certification object.
   A. In the Salesforce UI in the top right-hand corner, select **Setup**.
   B. Type `objects` in the Quick Find textbox.
   C. Click **Objects** under Create.
   D. In Custom Objects, click the link for **Certification**.
   E. Under Custom Fields & Relationships, click **New**.
   F. Under Step 1: Choose the field type.
      i. Data Type: **Roll-Up Summary**
      ii. Click **Next**.
   G. Under Step 2: Enter the details.
      i. Field Label: `Number of Certification Elements`
      ii. Field Name: `Number_of_Certification_Elements`
      iii. Click **Next**.
   H. Under Step 3: Define the summary calculation.
      i. Summarized Object: `Certification_Elements`
      ii. Select Roll-Up Type: **Count**.
      iii. Click **Next**.
   I. Under Step 4: Establish field-level security.
      i. Click **Next**.
   J. Under Step 5: Add to Page Layouts.
      i. Choose to automatically add the field to the page layout.
      ii. Click **Save**.

2. Create a hidden formula field on the Certification Attempt object.
   A. In the Salesforce UI in the top right-hand corner, select **Setup**.

B. Type `objects` in the Quick Find textbox.
C. Click **Objects** under Create.
D. In Custom Objects, click the link for **Certification Attempt**.
E. Under Custom Fields & Relationships, click **New**.
F. Under Step 1: Choose the field type.
    i. Data Type: **Formula**
    ii. Click **Next**.
G. Under Step 2: Choose output type.
    i. Field Label: `Number of Elements for Cert`
    ii. Field Name: `Number_of_Elements_for_Cert`
    iii. Formula Return Type: **Number**
    iv. Options – Decimal Places: **0**
    v. Click **Next**.
H. Under Step 3: Enter Formula.
    i. Number of Elements for Cert (Number):
    `Certification_Element__r.Certification__r.Number_of_Certification_Elements__c`
    ii. Click **Check Syntax.** Verify that no syntax errors occur.
    iii. Click **Next**.
I. Under Step 4: Establish field-level security.
    i. Click **Next**.
J. Under Step 5: Add to Page Layouts.
    i. Click the checkbox next to **Add Field** to deselect all page layouts
    ii. Click **Save**.

14-3: Create Collections to Filter the Query

Goal:

Create the query that will bring back Certification Attempt records.

Tasks:

1. Create the collections needed to filter the query.
2. Write a SOQL for loop that performs the query.
3. Change code to call the logic for the new class handler method.
4. Test your new trigger logic.

Time:

20 minutes

Instructions:

1. Create the collections needed to filter the query.
   A. In the Salesforce user interface, in the top right-hand corner, click on **your name** and then click **Developer Console**.
      i. Click **File | New | Apex Class**.
      ii. In the New Class dialog box, type `CertificationAttemptTriggerHandler`.
      iii. Click **OK**.
   B. Delete the default code and copy the contents of the CertificationAttemptTriggerHandler.txt file located in the Exercises folder into the content window.
   C. Complete the first two TODO sections to create collections needed to filter the query and then loop through the results, only adding records to the collections that have changed to 'Complete/Pass.'
   D. Click **Save**.

2. Write a SOQL for loop to that performs the query.
   A. In the tab for the CertificationAttemptTriggerHandler, complete the TODO sections to write a SOQL for loop that performs a query for Certification Attempt objects and uses an inner loop to write to the system debug log the values retrieved.
   B. Click **Save**.

3. Change code to call the logic for the new class handler method.
   A. Click **File | New | Apex Trigger**.
   B. In the New Trigger dialog box, type `CertificationAttemptTrigger` and select **Certification_Attempt__c** as the sObject.
   C. Click **OK**.
   D. Delete the default code and copy the contents of the CertificationAttemptTrigger.txt file located in the Exercises folder into the content window.
   E. Click **Save**. Leave Developer Console open.

4. Test your new trigger logic.
   A. In the Salesforce user interface, navigate to the **Certifications** tab and click **Go**.
   B. Click the link for the **AWCA Network** Certification.
   C. Click the link for the **AWCA Network Multiple Choice** Certification Element.
   D. Under the Certification Attempts related list, click the **New Certification Attempt** button.
   E. Click **Continue**.
   F. Enter `Clara Petit` as the Certification Candidate and select **Complete/Pass** as the Status.
   G. Click **Save**.
   H. In Developer Console, switch to the **Logs** tab.
   I. Double-click the **most recent entry** in the list to load the log file.
   J. Select the **Debug Only** checkbox. You should see several entries appear. Look for an entry that indicates that the `createCertificationHeld` method was called, followed by an entry that shows what Certification Attempt record was retrieved. This indicates that the new trigger logic was fired successfully.
   K. Type `CTRL + ALT + /` to close all open windows.

14-4: Use a Map to Aggregate Results

Goal:

Aggregate the results of the query for use in determining if a candidate has passed all elements of a certification.

Tasks:

1. Add logic to construct and use a Map to aggregate query results.
2. Test your new trigger logic.

Time:

20 minutes

Instructions:

1. Construct and use a Map to aggregate query results.
   A. In the Developer Console, click **File | Open**.
   B. Select **Classes** as the Entity Type, **CertificationAttemptTriggerHandler** as the Entity and click **Open**.
   C. Delete the default code and copy the contents of the CertificationAttemptTriggerHandler.txt file located in the `Exercises` folder into the content window.
   D. Complete the TODO sections to construct and use a map variable to aggregate query results and write the number of elements with a passed count to the debug log.
   E. Click **Save**. Leave Developer Console open.
2. Test your new trigger logic.
   A. In the Salesforce user interface, navigate to the **Certifications** tab and click **Go**.
   B. Click the link for the **AWCA Network** Certification.
   C. Click the link for the **AWCA Network Multiple Choice** Certification Element.
   D. Under the Certification Attempts related list, click the **New Certification Attempt** button.
   E. Click **Continue**.
   F. Enter `Arthur Franz` as the Certification Candidate and select **Complete/Pass** as the Status.
   G. Click **Save**.
   H. In Developer Console, switch to the **Logs** tab.
   I. Double-click the **most recent entry** in the list to load the log file.
   J. Select the **Debug Only** checkbox. You should see several entries appear. Look for an entry that indicates that the `createCertificationHeld` method was called, followed by an entry that shows what Certification Attempt record was retrieved, and finally one that shows the size of passCounts equals one. This indicates that the new trigger logic was fired successfully.
   K. Type `CTRL + ALT + /` to close all open windows.

14-5: Create Certification Held Records

Goal:

Complete the solution to create Certification Held records for qualified candidates.

Tasks:

1. Create Certification Held records for qualified candidates.
2. Test your solution.

Time:

15 minutes

Instructions:

1. Create Certification Held records for qualified candidates.
   A. In the Developer Console, click **File | Open**.
   B. Select **Classes** as the Entity Type, **CertificationAttemptTriggerHandler** as the Entity and click **Open**.
   C. Delete the default code and copy the contents of the CertificationAttemptTriggerHandler.txt file located in the `Exercises` folder into the content window.
   D. Complete the TODO sections to create a new Certification Held record for qualified candidates.
   E. Click **Save**. Leave Developer Console open.
2. Test your solution.
   A. In the Salesforce user interface, navigate to the **Certifications** tab and click **Go**.
   B. Click the link for the **AWCA Network** Certification.
   C. Click the link for the **AWCA Network Multiple Choice** Certification Element.
   D. Under the Certification Attempts related list, click the **New Certification Attempt** button.
   E. Click **Continue**.
   F. Enter `Clara Morales` as the Certification Candidate and select **Complete/Pass** as the Status.
   G. Click **Save**.
   H. In Developer Console, switch to the **Logs** tab.
   I. Double-click the **most recent entry** in the list to load the log file.
   J. Select the **Debug Only** checkbox. You should see several entries appear. Look for an entry that indicates that the `createCertificationHeld` method was called, followed by an entry that shows what Certification Attempt record was retrieved, one that shows the size of passCounts equals one and finally one that says a certification held record was added. This indicates that the new trigger logic was fired successfully.
   K. Type `CTRL + ALT + /` to close all open windows.
   L. Close the Developer Console.

14-6: Use a Workflow to Avoid Creation of Duplicate Records (Optional)

Goal:

Create a workflow rule that will prevent duplicate Certification Held records from being created.

Tasks:

1. Create a new hidden text field.
2. Create a workflow rule to populate the new field.
3. Execute the workflow rule for all existing records.
4. Test your workflow rule.

Time:

10 minutes

Instructions:

1. Create a new hidden text field.
   A. In the Salesforce UI in the top right-hand corner, select **Setup**.
   B. Type `objects` in the Quick Find textbox.
   C. Click **Objects** under Create.
   D. In Custom Objects, click the link for **Certification Held**.
   E. Under Custom Fields & Relationships, click **New**.
   F. Under Step 1: Choose the field type.
      i. Data Type: **Text**
      ii. Click **Next**.
   G. Under Step 2: Enter the details.
      i. Field Label: `Cert Held Dup Key`
      ii. Length: `37`
      iii. Field Name: `Cert_Held_Dup_Key`
      iv. Unique: **Select**
      v. Click **Next**.
   H. Under Step 3: Establish field-level security.
      i. Select checkbox next to **Read-Only** to select all checkboxes.
      ii. Click **Next**.
   I. Under Step 4: Add to Page Layouts.
      i. Click the checkbox next to **Add Field** to deselect all page layouts.
      ii. Click **Save**.

2. Create a workflow rule to populate the new field.
   A. In the Salesforce UI in the top right-hand corner, select **Setup**.
   B. Type `workflow` in the Quick Find textbox.

C. Click **Workflow Rules** under Workflow & Approvals.
D. Under All Workflow Rules, click **New Rule**.
E. Under Step 1: Select Object.
    i. Object: **Certification Held**
    ii. Click **Next**.
F. Under Step 2: Configure Workflow Rule.
    i. Rule Name: `PopulateCertHeldDupKey`
    ii. Evaluation Criteria: **created, and every time it's edited**
    iii. Run this rule if the following: **formula evaluates to true**
    iv. formula: **true**
    v. Click **Save & Next**.
G. Under Step 3: Specify Workflow Actions.
    i. Add Workflow Action: **New Field Update**
    ii. Click **Next**.
H. Under New Field Update.
    i. Name: `Cert Held Dup Key`
    ii. Unique Name: `Cert_Held_Dup_Key`
    iii. Field to Update: **Certification Held – Cert Held Dup Key**
    iv. Text Options: `Use a formula to set the new value`
    v. Click **Show Formula Editor**.
    vi. Click **Insert Field**.
    vii. Select **Certification Held> Certification**.
    viii. Click **Insert**.
    ix. Click **Insert Operator | (&) Concatenate**.
    x. Click **Insert Field**.
    xi. Select **Certification Held> Certified Professional**.
    xii. Click **Insert**.
    xiii. Click **Save**.
I. Click **Done**.
J. Click **Activate**.

3. Execute the workflow rule for all existing records.
    A. In the Developer Console, type `CTRL + E` to open the Enter Apex Code window.
    B. Remove any code that is currently in place, and execute the following code:

```
update [SELECT Id FROM Certification_Held__c];
```

4. Test your workflow rule.
    A. In the Salesforce user interface, navigate to the **Contacts** tab and click **Go**.
    B. Click the link for the **Morales, Clara** Contact.
    C. Verify that she has a Certification Held record for the AWCA Network Certification. This should have been added when completing the last exercise.
    D. Navigate to the **Certifications** tab and click **Go**.
    E. Click the link for the **AWCA Network** Certification.
    F. Click **New Certification Held** in the Certifications Held related list.

G. Enter `Clara Morales` as the Certified Professional and select the current date as the Date Achieved.
H. Click **Save**.
I. Verify that you see the error message, “Duplicate value on record.” This indicates that the new workflow rule was fired successfully.

15-1: Create a Simple Visualforce Page

Goal:

Create a simple Visualforce page that displays your name.

Tasks:

1. Create a Visualforce page using the inline editor.
2. Add static text to the page.
3. Add a reference to the global user variable to display your name.

Time:

10 minutes

Instructions:

1. Create a Visualforce page using the inline editor.
   A. Change the URL in the address bar of the browser to:

      `https://salesforceserver.salesforce.com/apex/HelloPage`

      Note: Be sure to replace `salesforceserver.salesforce.com` with the exact site name from your URL, which is typically something like `na9.salesforce.com`.

      A warning will be displayed that the page does not exist.
   B. Click the **Create Page HelloPage** link.
2. Add static text to the page.
   A. Once the page has been created and loaded, click **HelloPage** in the lower left corner of the screen.
   B. Delete all of the text between the `<apex:page>` components (lines 2-5) and enter:

      ```
      <b>Hello World</b>
      ```

   C. Click **Save**, then examine the changes in the page.
3. Add a reference to the global user variable to display your name.
   A. Modify the code for your page by replacing "`World`" with a reference to the global username variable. It will look like this:

      ```
      <b>Hello {!$User.firstName}</b>
      ```

   B. Click **Save**, then examine the changes in the page.

15-2: Display Data in a Visualforce Page

Goal:

Create a Visualforce Page that prints a simple Course Completion certificate.

Tasks:

1. Upload the pre-existing image file to be used as the certificate banner.
2. Create a new Visualforce page using Developer Console.
3. Create a custom button to launch the new Visualforce page.
4. Test the new page.

Time:

15 minutes

Instructions:

1. Upload the pre-existing image file to be used as the certificate banner.
   A. Click **Setup.**
   B. Type `static` in the Quick Find textbox.
   C. Click **Static Resources** under Develop**.**
   D. Click **New.**
      i. Name: `CourseCertificateImage`
      ii. Choose File: locate the `CourseCertificateImage.png` file in your Exercises folder.
      iii. Cache Control: **Public**
      iv. Click **Save**.
2. Create a new Visualforce page using the Developer Console.
   A. In the Salesforce UI in the top right-hand corner, click on **your name** and then click **Developer Console**.
   B. Click **File | New | Visualforce Page**.
      i. Name: `CourseCertificate`
      ii. Click **OK**.
   C. Delete the default contents and copy the contents of the `CourseCertificatePage.txt` file located in the Exercises folder into the content window and click **Save.**
   D. Complete the code, finishing the sections marked TODO and click **Save.**
3. Create a custom button to launch the new Visualforce page.
   A. Click **Setup**.
   B. Type `object` in the Quick Find textbox.
   C. Click **Objects** under Create.
   D. Click **Course**.
   E. In the Button, Links, and Actions section, click **New Button or Link**.

i. Label: `Course Certificate`
ii. Name: `Course_Certificate`
iii. Display Type: **Detail Page Button**
iv. Behavior: **Display in new window**
v. Content Source: **Visualforce Page**
vi. Content: **CourseCertificate**
vii. Click **Save**.
viii. Click **OK** to the warning about adding the button to the page layout.

F. Add the button to the Course layout.
   i. Click **Back to Custom Object: Course**.
   ii. In the Page Layouts related list, click **Edit** next to Course Layout.
   iii. Click the **Buttons** category.
   iv. Add the **Course Certificate** button to the **Custom Buttons** area.
   v. Click **Save**.

4. Test the new page.

   A. Return to the **Courses** tab and select a **Course**.
   B. On the Course detail page, click the **Course Certificate** button.
   C. Ensure that the certificate is rendered and that the course name listed on the certificate is correct**.**

16-1: Create a Simple Technician Status Page

Goal:

Create a simple technician status page to display technician name, account name, and courses attended. Launch the page with a custom button.

Tasks:

1. Provide a record to be used as context during development.
2. Create a technician status page.
3. Complete the TODO sections to display the necessary data.
4. Create a custom button on the Technician Contact Layout to launch the page.
5. Test your new page.

Time:

15 minutes

Instructions:

1. Provide a record to be used as context during development.
   A. Open a Contact record by clicking on the **Contacts** tab. Click the **Go!** button if necessary and open a Contact record.
   B. Copy the Salesforce record ID from the end of the URL:

      `https://salesforceserver.salesforce.com/RecordID`

   C. Paste the record ID in text editor such as Notepad.

2. Create a Visualforce page using the inline editor.
   A. Change the URL in the address bar of the browser to:

      `https://salesforceserver.salesforce.com/apex/TechnicianStatus`

      Note: Be sure to replace `salesforceserver.salesforce.com` with the exact site name from your URL, which is typically something like `na9.salesforce.com`.

      A warning will be displayed that the page does not exist.
   B. Click the **Create Page TechnicianStatus** link.
   C. Once the page has been created and loaded, click **TechnicianStatus** in the lower left corner of the screen.
   D. Add `?id=RecordID` to the end of the URL and press **ENTER**. Use the record ID copied in Step B as `RecordID`. Your new URL will look like the following:

      `https://salesforceserver.salesforce.com/apex/TechnicianStatus?id=RecordID`

3. Complete the TODO sections to display the necessary data.
   A. Delete all the contents of the page and paste the contents of the `SimpleTechnicianStatusPage.txt` file from the Exercises folder.

B. Complete the TODO sections to display data.
C. Click **Save**, then examine the changes in the page.

4. Create a custom button on the Technician Contact Layout to launch the page.

   A. Click **Setup**.
   B. Type `contact` in the Quick Find textbox.
   C. Click **Buttons, Links, and Actions** under Customize Contacts.
   D. Click the **New Button or Link** button.

      i. Label: `Technician Status`
      ii. Name: `Technician_Status`
      iii. Display Type: **Detail Page Button**
      iv. Behavior: **Display in new window**
      v. Content Source: **Visualforce Page**
      vi. Content: **TechnicianStatus**
      vii. Click **Save**.
      viii. Click **OK** to the warning about adding the button to the page layout.

   E. Add the button to the Technician Contact Layout.

      i. Click **Setup**.
      ii. Type `contact` in the Quick Find textbox.
      iii. Click **Page Layouts** under Customize Contacts.
      iv. Click **Edit** next to Technician Contact Layout.
      v. Click the **Buttons** category.
      vi. Add the **Technician Status** button to the Custom Buttons area.
      vii. Click **Save**.

5. Test your new page.

   A. Navigate to the **Contacts** tab.
   B. Select a Contact record that has been assigned the Technician Contact Record Type, e.g. Moreau, Leo or Li, Jian.
   C. Click the **Technician Status** button. Notice that the Visualforce page has been launched.
   D. Notice the Courses Attended section at the bottom of the page, which displays the additional information provided by the code.
   E. Close the Visualforce page.

16-2: Refine Your Page and Add Navigational Links

Goal:

Refine your Visualforce page to display only the necessary fields, then add navigational links so users can easily edit related records.

Tasks:

1. Overwrite the existing TechnicianStatus Visualforce page to only display the necessary fields.
2. Complete the TODO sections to refine your page and add navigational links.
3. Test the new page.

Time:

10 minutes

Instructions:

1. Overwrite the existing Visualforce page.
   A. Change the URL in the address bar of the browser to:

      `https://salesforceserver.salesforce.com/apex/TechnicianStatus?id=RecordID`

      and press **Enter**. Use the record ID of the Contact record, pasted into your text editor in the previous exercise, as `RecordID`.
      Note: Be sure to replace `salesforceserver.salesforce.com` with the exact site name from your URL, which is typically something like `na9.salesforce.com`.
   B. Delete all the contents of the page and paste the contents of the `NavigationalLinksPage.txt` file from the Exercises folder.
2. Complete the TODO sections to refine your page and add navigational links.
   A. Complete the TODO sections to refine your page.
   B. Click **Save**, then examine the changes in the page.
3. Test your new page.
   A. Navigate to the **Contacts** tab.
   B. Select a Contact record that has been assigned the Technician Contact Record Type, e.g., Moreau, Leo or Li, Jian.
   C. Click the **Technician Status** button.
   D. Click the **Edit Technician Record** button to confirm that you can edit the record.
   E. Click **Cancel** to return to the previous page.
   F. In the table, click the **Date** link to view the Course Attendee record of the Technician.
   G. Close the page.

17-1: Reference a Controller Extension in a Visualforce Page

Goal:

Use the provided controller extension to display all Certification Held records related to the current account in a Visualforce page embedded on the account page layout.

Tasks:

1. Upload the pre-existing controller extension into the org.
2. Create a page to display all Certification Held records.
3. Create a section on the Account Page Layout to display the new page.
4. Test your new page.

Time:

15 minutes

Instructions:

1. Upload the pre-existing controller extension into the org.
   A. In the Salesforce user interface, in the top right-hand corner, click on **your name** and then click **Developer Console**.
   B. In the Developer Console, click **File | New | Apex Class**.
      i. Name: AccountDisplayCertsHeld_CX
      ii. Click **OK**.
   C. Delete the default code and copy the contents of the AccountDisplayCertsHeld_CX.txt file located in the Exercises folder into the content window and type `CTRL + S` to save the class.
2. Create a page to display the Certification Held records.
   A. In the Developer Console, click **File | New | Visualforce Page.**
      i. Name: AccountDisplayCertsHeld
      ii. Click **OK**.
   B. Delete the default page and copy the contents from the AccountDisplayCertsHeldPage.txt file located in the Exercises folder into the content window.
   C. Complete the TODO sections to add page attributes and a pageBlockTable component.
   D. Type `CTRL + S` to save the page.
   E. Type `CTRL + ALT + /` to close all open windows.
   F. Close the Developer Console.
3. Create a section on the Account page layout to display the new page.
   A. Click **Setup**.
   B. Type account in the Quick Find textbox.
   C. Click **Page Layouts** under Customize Accounts.
   D. In the Page Layouts related list, click the **Edit** link next to the Service Vendor Account

Layout page layout.

E. Click the **Visualforce Pages** category.
F. Drag a **Section** element and place below the Account Information section.
    i. Section Name: Certifications Held
    ii. Display Section Header On: **Only Detail Page** should be selected
    iii. Layout: 1-Column
    iv. Click **OK**.
G. Drag the **AccountDisplayCertsHeld** page into the new section.
H. Click the **Tool** icon in the top right area of the page to access the Visualforce Page Properties.
    i. Width (in pixels or %): 100%
    i. Height (in pixels): 100
    ii. Show Scrollbars: **Select**
    iii. Click **OK**.
I. Click **Save**.

4. Test your new page.
    A. Enter Alveswood in the Search textbox and select **Alveswood Technologies** as the Account.
    B. Verify that the new section appears and that you can scroll through all the records.

17-2: Create a Simple Read-Only Property

Goal:

Create a simple controller extension that uses a property with a get block to allow you to display a table showing all currently in-progress certification attempts associated with an account on the account page layout.

Tasks:

1. Write the code necessary for the controller extension.
2. Create a page to display in-progress Certification Attempt records.
3. Create a section on the Account Page Layout to display the new page.
4. Test your new page.

Time:

15 minutes

Instructions:

1. Write the code necessary for the controller extension.
   A. In the Salesforce user interface, in the top right-hand corner, click on **your name** and then click **Developer Console**.
   B. Click **File | New | Apex Class**.
      i. Name: AccountDisplayCertAttempts_CX
      ii. Click **OK**.
   C. Delete the default code and copy the contents of the AccountDisplayCertAttempts_CX.txt file located in the Exercises folder into the content window.
   D. Complete the TODO sections to create the controller extension constructor and write the read-only property.
   E. Click **Save**.
2. Create a page to display in-progress Certification Attempt records.
   A. In the Developer Console, click **File | New | Visualforce Page**.
      i. Name: AccountDisplayCertAttempts
      ii. Click **OK**.
   B. Delete the default page and copy the contents from the AccountDisplayCertAttemptsPage.txt file located in the Exercises folder into the content window.
   C. Complete the TODO sections to display the data from the controller extension.
   D. Click **Save**.
   E. Type CTRL + ALT + / to close all open windows.
   F. Close the Developer Console.

3. Create a section on the Account page layout to display the new page.
   A. Click **Setup**.
   B. Type account in the Quick Find textbox.
   C. Click **Page Layouts** under Customize Accounts.
   D. In the Page Layouts related list, click the **Edit** link next to the Service Vendor Account Layout page layout.
   E. Click the **Visualforce Pages** category.
   F. Drag a **Section** element and place below the Certifications Held section.
      i. Section Name: Certification Attempts
      ii. Display Section Header On: **Only Detail Page** should be selected
      iii. Layout: 1-Column
      iv. Click **OK**.
   G. Drag the **AccountDisplayCertAttempts** page into the new section.
   H. Click the **Tool** icon in the top right area of the section to access the Visualforce Page Properties.
      i. Width (in pixels or %): 100%
      i. Height (in pixels): 100
      ii. Show Scrollbars: **Select**
      iii. Click **OK**.
   I. Click **Save**.
4. Test your new page.
   A. Enter Alveswood in the Search textbox and select **Alveswood Technologies** as the Account.
   B. Verify that the new section appears and displays one or more records.

17-3: Creating a Read/Write Property in a Custom Controller

Goal:

Create a Visualforce page and custom controller to display a list of courses with corresponding checkboxes. Begin writing the action method that will trigger the search.

Tasks:

1. Write the code necessary for the custom controller.
2. Create a page to display all courses.
3. Test your new page.

Time:

25 minutes

Instructions:

1. Write the code necessary for the custom controller.
   A. In the Salesforce user interface, in the top right-hand corner, click on **your name** and then click **Developer Console**.
   B. Click **File | New | Apex Class**.
      i. Name: SearchCourses_CC
      ii. Click **OK**.
   C. Delete the default code and copy the contents of the SearchCourses_CC.txt file located in the Exercises folder into the content window.
   D. Complete the TODO sections to create the controller extension constructor and write the read-only property.
   E. Click **Save**.
2. Create a page to display all courses.
   A. In the Developer Console, click **File | New | Visualforce Page**.
      i. Name: SearchCourses
      ii. Click **OK**.
   B. Delete the default page and copy the contents from the SearchCoursesPage.txt file located in the Exercises folder into the content window.
   C. Complete the TODO sections to display the data from the custom controller.
   D. Click **Save**.
3. Test your new page.
   A. In the Developer Console, with the tab selected for the **SearchCourses** page, click **Preview** to open your browser and load the new page. Do NOT close Developer Console.
   B. In the Salesforce user interface with the page displaying the new SearchCourses page, select one or more of the **checkboxes** next to the courses listed (Hint: include [101 AWCA Server]) and click **See Upcoming Course Deliveries**.

C. In the Developer Console, select the **Perspective manager** from the Debug menu. Set the default perspective to **Log Only (Predefined)**, if it is not already the default.
D. Select the **Logs** tab and double-click the latest entry to view the contents.
E. Select the **Debug Only** checkbox and ensure that the names for the course(s) you selected appear in the results.
F. Type CTRL + ALT + / to close all open windows.
G. Close the Developer Console.

17-4: Implement the Search Button

Goal:

Create action method to create a map of just the selected courses. Your method should fire when the user clicks the Search button.

Tasks:

1. Write the code necessary to create map of selected courses.
2. Test the code changes.

Time:

10 minutes

Instructions:

1. Write the code necessary to create map of selected courses.
   A. In the Salesforce user interface, in the top right-hand corner, click on **your name** and then click **Developer Console**.
   B. Click **File | Open**.
   C. Select **Classes** as the Entity Type, **SearchCourses_CC** as the Entity, and click **Open**.
   D. Delete the contents of the file and copy the contents from the SearchCourses_CC.txt file located in the Exercises folder into the content window.
   E. Complete the TODO sections to create a map of selected courses.
   F. Click **Save**.
2. Test the code changes.
   A. In the Developer Console, click **File | Open**.
   B. Select **Pages** as the Entity Type, **SearchCourses** as the Entity, and click **Open**.
   C. In the Developer Console, with the tab selected for the **SearchCourses** page, click **Preview** to open your browser and load the new page. Do NOT close Developer Console.
   D. In your browser displaying the new SearchCourses page, select one or more of the **checkboxes** next to the courses listed and click **See Upcoming Course Deliveries**.
   E. In Developer Console, select the **Perspective manager** from the Debug menu. Set the default perspective to **Log Only (Predefined)**, if it is not already the default.
   F. Select the **Logs** tab and double-click the latest entry to view the contents.
   G. Select the **Debug Only** checkbox and ensure that the names for the course(s) you selected appear in the results. You should also see an entry for the number of selected courses.
   H. Type CTRL + ALT + / to close all open windows.
   I. Close the Developer Console.

17-5: Redirecting to a Results Page

Goal:

You must redirect the user to show the results of the search they have set up.

Tasks:

1. Create a new results page.
2. Write the code necessary to redirect the user to a new page.
3. Add code to the new page to display the course deliveries.
4. Test your new page.

Time:

25 minutes

Instructions:

1. Create a new results page.
   A. In the Salesforce user interface, in the top right-hand corner, click on **your name** and then click **Developer Console**.
   B. Click **File | New | Visualforce Page**.
      i. Name: ListCourseDeliveries
      ii. Click **OK**.

   Note: We will return to this page in a later step to complete the code for it.
2. Write the code necessary to redirect the user to a new page.
   A. In the Developer Console, click **File | Open**.
   B. Select **Classes** as the Entity Type, **SearchCourses_CC** as the Entity, and click **Open**.
   C. Delete the contents of the file and copy the contents from the SearchCourses_CC.txt file located in the Exercises folder into the content window.
   D. Complete the TODO sections to redirect the user to a new page, return course delivery records for the selected courses and redirect the user back to the original page.
   E. Click **Save**.
3. Add code to the new page to display the course deliveries.
   A. In the Developer Console, return to the tab for your new page named ListCourseDeliveries.
   B. Delete the contents of the file and copy the contents from the ListCourseDeliveriesPage.txt file located in the Exercises folder into the content window.
   C. Complete the TODO sections to display results and redirect back to original page.
   D. Click **Save**.
4. Test your new page.
   A. In the Salesforce user interface, change the URL in the address bar of the browser to: https://salesforceserver.salesforce.com/apex/SearchCourses
   B. Select one or more of the **checkboxes** next to the courses listed and click **See**

Upcoming Course Deliveries.

C. Verify that the Upcoming Course Deliveries page displays course deliveries related to the course(s) you selected. Also verify that you can click the **New Search** button and return to the original search page.

17-6: Handle Basic Save Errors in Your Method

Goal:

Add simple error handling to make sure the user has selected courses to search for before clicking Search.

Tasks:

1. Add conditional logic to check for selected course.
2. Add the <apex:pageMessages/> tag to your Visualforce page.
3. Test the code changes.

Time:

10 minutes

Instructions:

1. Add conditional logic to check for selected course.
   A. In the Salesforce user interface, in the top right-hand corner, click on **your name** and then click **Developer Console**
   B. In the Developer Console, click **File | Open**.
   C. Select **Classes** as the Entity Type, **SearchCourses_CC** as the Entity, and click **Open**.
   D. Delete the contents of the file and copy the contents from the SearchCourses_CC.txt file located in the Exercises folder into the content window.
   E. Complete the TODO sections to add conditional logic to check for a selected course.
   F. Click **Save**.
2. Add conditional logic to check for selected course.
   A. In the Developer Console, click **File | Open**.
   B. Select **Pages** as the Entity Type, **SearchCourses** as the Entity, and click **Open**.
   C. Delete the contents of the file and copy the contents from the SearchCoursesPage.txt file located in the Exercises folder into the content window.
   D. Complete the TODO sections to add a pageMessages component.
   E. Click **Save**.
   F. Type CTRL + ALT + / to close all open windows.
   G. Close the Developer Console.
3. Test the code changes.
   A. In the Salesforce user interface, change the URL in the address bar of the browser to: https://salesforceserver.salesforce.com/apex/SearchCourses
   B. Select NO checkboxes next to the courses listed and click **See Upcoming Course Deliveries**.
   C. Verify that a message appears telling you to “Please select at least one Course.”

18-1: Create a Page to Display a List of Records

Goal:

Create a Visualforce page that displays all Account records and uses pagination and filtered list views.

Tasks:

1. Create a page to display all Account records.
2. Add pagination and filtered list views to the page.
3. Test your new page.

Time:

15 minutes

Instructions:

1. Create a page to display all Account records.
   A. In the Salesforce user interface, in the top right-hand corner, click on **your name** and then click **Developer Console**.
   B. In the Developer Console, click **File | New | Visualforce Page**.
      i. Name: `AccountDisplay`
      ii. Click **OK**.
   C. Delete the default code and copy the contents of the AccountDisplayPage.txt file located in the Exercises folder into the content window.

2. Add pagination and filtered list views to the page.
   A. Complete the TODO sections to add commandButtons to handle pagination and a selectList component to handle filtering**.**
   B. Click **Save**.

3. Test your new page.
   A. In the Developer Console, click **Preview** to open your browser and load the new page.
   B. Type `CTRL + ALT + /` to close all open windows.
   C. Close the Developer Console.
   D. In the Salesforce user interface with the page displaying the new `AccountDisplay` page, click the **Next** button to move to the next page and notice that the text displaying the number of records has changed. You might also notice that the word “Working” appears briefly as the results advance.
   E. Continue paging through the results until you get to the final page, or use the **Last** button to click to the last page. Notice that when you get to the end of the results, the **Next** and **Last** buttons are disabled.
   F. Select a different filter from the Filter by selectList and notice that the results displayed have changed.

18-2: Integrate SOSL Search in a Visualforce Page

Goal:

Search for text across multiple sObjects.

Tasks:

1. Construct a simple SOSL search.
2. Create a code block to cycle through the results.

Time:

15 minutes

Instructions:

1. Construct a simple SOSL search.
   A. Open the Developer Console.
   B. In the Developer Console, navigate to the **Query Editor** tab.
   C. Delete the existing contents in the Query Editor tab.
   D. Write a SOSL search that searches for a person named `Frank`. This search should be made in the Contact and User. If the search string is found, the person's first name and last name should be returned.
   Note: The search string must be delimited by curly braces without any quotes in the Query Editor tab. However, single quotes are required in an Anonymous block.
   E. Click **Execute** to test your code. Note that there is a separate tab for each sObject type.

2. Create a code block to cycle through the results.
   A. In the Developer Console, type `CTRL + E` to open the Enter Apex Code window.
   B. Remove any code that is currently in place, and insert the code from the Exercise file ParseSOSLSearch.txt.
   C. Complete the TODOs as required.
   D. Click **Execute** to test your code. Review the resulting debug log.
   E. In the Developer Console, type `CTRL + ALT + /` to close all open windows.
   F. Close the Developer Console.

18-3: Create a Simple Search Page

Goal:

Create a Visualforce page and custom controller to display a list of contacts based on the results of a SOSL search.

Tasks:

1. Write the code necessary for the custom controller.
2. Create a page to search for contacts.
3. Test your new page.

Time:

15 minutes

Instructions:

1. Write the code necessary for the controller extension.
   A. In the Salesforce user interface, in the top right-hand corner, click on **your name** and then click **Developer Console**.
   B. In the Developer Console, click **File | New | Apex Class**.
      i. Name: `DisplayContacts_CC`
      ii. Click **OK**.
   C. Delete the default code and copy the contents of the DisplayContacts_CC.txt file located in the Exercises folder into the content window.
   D. Complete the TODO sections to create the searchText property and add code to the Search action method.
   E. Click **Save**.

2. Create a page to display all courses.
   A. In the Developer Console, click **File | New | Visualforce Page**.
      i. Name: `DisplayContacts`
      ii. Click **OK**.
   B. Delete the default page and copy the contents from the DisplayContactsPage.txt file located in the Exercises folder into the content window.
   C. Complete the TODO sections to add an inputText component and display the search results in a pageBlockTable component.
   D. Click **Save**.

3. Test your new page.
   A. In the Developer Console, with the tab selected for the `DisplayContacts` page, click **Preview** to open your browser and load the new page.

B. Type `CTRL + ALT + /` to close all open windows.
C. Close the Developer Console.
D. In the Salesforce user interface with the page displaying the new `DisplayContacts` page, enter a value in the Search Text field that will be used to search across all fields in Contacts and click **Search**.
E. Verify that results are returned in a table and that the pagination buttons for **Next** and **Last** and enabled. Notice that the wildcard character * has been appended to the end of the search text.
F. Erase the value in the Search Text input field and click **Search**. Verify that a message appears that informs you that “Error: Please enter search text at least two characters long.” SOSL searches require this.

19-1: Determine Whether a Declarative Solution Exists

Goal:
Determine which scenarios are best solved using Visualforce.

Task:
Review each scenario and determine if Visualforce would be the best solution.

Time:
5 minutes

Scenarios:

1. The certification team would like to include a Chatter feed on each Certification record.

2. The certification team would like the Start Date field to display only when the Status is set to Scheduled on a Course Delivery.

3. The certification team would like the Course Deliveries related list to be the only related list displayed on the Course record.

4. The Contact object has a lot of fields and requires scrolling to see all the information. The certification team would like every section and related list to display as an individual tab that can be viewed when clicked on.

5. The certification team would like the Course list view to match the look and feel of their corporate website.

19-2: Defend Against SOQL Injection

Goal:

Modify the controller of a Visualforce page to defend against SOQL Injection.

Tasks:

1. Create a custom Visualforce Controller.
2. Create a Visualforce page.
3. Search for an existing record.
4. Sanitize the code to defend against SOQL Injection attacks.

Time:

10 minutes

Instructions:

1. Create a Controller Extension and save it.
   A. In the Salesforce UI in the top right-hand corner, select ***your name*** **| Developer Console**.
   B. In the Developer Console, select **File | New | Apex Class**.
   C. In the New Class dialog box, type `SOQLController` for the name of your new class.
   D. Click **OK**.
   E. Copy and paste the contents of SOQLControllerClass.txt from the Exercises folder into the window, overwriting all existing text. Do NOT complete the TODO at this time.
   F. Type `CTRL + S` to save the class.
2. Create a Visualforce page.
   A. In the Developer Console, select **File | New | Visualforce Page**.
   B. In the New Apex Page dialog box, type `CertificationHeldQuickInfo` for the name of your new page.
   C. Click **OK**.
   D. Copy and paste the contents of CertificationHeldQuickInfoPage.txt from the Exercises folder into the window, overwriting all existing text.
   E. Type `CTRL + S` to save the page.
   F. Close the Developer Console.
3. Search for an existing record.
   A. Change the URL in the address bar of the browser to:

      ```
      https://salesforceserver.salesforce.com/apex/CertificationHeldQuickInfo
      ```

      Note: Be sure to replace `salesforceserver.salesforce.com` with the exact site name from your URL, which is typically something like `na9.salesforce.com`.
   B. Type `CERTIFICATION-00001` in the textbox.
   C. Click **Query**.

D. Verify that the correct information is displayed.

4. Sanitize the code to defend against SOQL Injection attacks.

   A. Type `CERTIFICATION` and click **Query**. Try `CERTIFICATION-0000*`, `CERTIFICATION-0000?`, `CERTIFICATION-0000_` or any other text to confirm that the query is not displaying unintended information.
   B. Simulate a SOQL Injection attack by typing:

      ```
      test' OR name like '%
      ```

      This has the effect of displaying all Certification Held records.
   C. In the Development Mode Footer, click the SOQLController tab and complete the TODO to ensure that all records are never displayed.
   D. Click the Disc icon to save the class and refresh the page.
   E. Repeat the query to simulate a SOQL injection attack by typing `test' OR name like'%`. This time there should be no results displayed.

19-3: Create a Custom Button that Uses JavaScript (Optional)

Goal:

Modify the TechnicianStatus Visualforce page to include a custom button.

Tasks:

1. Upload the JavaScript as a static resource.
2. Open the TechnicianStatus Visualforce page and complete the TODOs.
3. Test the button.

Time:

10 minutes

Instructions:

1. Upload the JavaScript as a static resource.
   A. Click **Setup**.
   B. Type `static` in the Quick Find textbox.
   C. Click **Static Resources** under Develop.
   D. Click **New**.

Name:	`customCancelButton`
Description:	`A custom cancel button that creates a dialog box to ask users if they are sure they want to cancel.`
File:	Click **Choose File**. Navigate to the Exercises folder and select **customCancelButton.txt**. Click **Open**.

2. Open the TechnicianStatus Visualforce page.
   A. Change the URL in the address bar of the browser to:

   `https://salesforceserver.salesforce.com/apex/TechnicianStatus?id=RecordID`

   and press **Enter**. Use the record ID of a Contact record for `RecordID`.
   B. Copy and paste the contents of `TechnicianStatusPage.txt` from the Exercises folder into the inline editor window, overwriting all existing text.
   C. In the inline editor, complete the TODOs to create a Cancel button which calls the custom JavaScript.
3. Test the button.
   A. Navigate to the **Contacts** tab.
   B. Select a Contact record that has been assigned the Technician Contact Record Type,

e.g., **Li, Jian**.

C. Click the **Technician Status** button.
D. Click the **Cancel** button. Confirm that the JavaScript executes.

20-1: Write the Test Method for the Constructor

Goal:

Write a test method to verify that the Technician Status page's controller extension constructor is invoked successfully.

Tasks:

4. Upload controller code for a new extended version of Technician Status.
5. Upload markup code for a new extended version of Technician Status.
6. Create a unit test method to test the extension constructor.
7. Test your new unit test logic.

Time:

20 minutes

Instructions:

1. Upload controller code for a new extended version of Technician Status.
   A. In the Salesforce user interface, in the top right-hand corner, click on **your name** and then click **Developer Console**.
   B. Click **File | New | ApexClass**.
      i. Name: `TechnicianStatus_CX`
      ii. Click **OK**.
   C. Delete the default code and copy the contents of the TechnicianStatus_CX.txt file located in the Exercises folder into the content window.
   D. Click **Save**.

2. Upload markup code for new extended version of Technician Status.
   A. Click **File | Open**.
   B. Select **Pages** as the Entity Type and **TechnicianStatus** as the Entity and click **Open**.
   C. Delete the default code and copy the contents of the TechnicianStatusPage.txt file located in the Exercises folder into the content window.
   D. Click **Save**.

3. Create a unit test method to test the extension constructor.
   A. Click **File | New | ApexClass**.
      i. Name: `TechnicianStatus_Test`
      ii. Click **OK**.
   B. Delete the default code and copy the contents of the TechnicianStatus_Test.txt file located in the Exercises folder into the content window.
   C. Complete the TODO sections to create testing logic.
   D. Click **Save**.

4. Test your new unit test logic.
   A. Click **Run Test**.
   B. Switch to the **Tests** tab and wait for the test to complete. You should see it pass with a green checkmark, no failures, and 1 total test. If it does not, then look at the results, correct the problem, and re-run the test until it completes successfully.
   C. Type `CTRL + ALT + /` to close all open windows.
   D. Close the Developer Console.

20-2: Write Unit Tests for Action Methods

Goal:

Write the necessary unit tests for the two custom action methods in the controller extension.

Tasks:

1. Examine the controller extension code for the Technician Status Page.
2. Create unit test methods to test the two custom action methods.
3. Test your new unit test logic.

Time:

20 minutes

Instructions:

1. Examine the controller extension code for the Technician Status Page.
   A. In the Salesforce user interface, in the top right-hand corner, click on **your name** and then click **Developer Console**.
   B. Click **File | Open**.
   C. Select **Classes** as the Entity Type and **TechnicianStatus_CX** as the Entity and click **Open**.
   D. Examine the code for the two action methods.

2. Create unit test methods to test the two custom action methods.
   A. Click **File | Open**.
   B. Select **Classes** as the Entity Type and **TechnicianStatus_Test** as the Entity and click **Open**.
   C. Delete the default code and copy the contents of the TechnicianStatus_Test.txt file located in the Exercises folder into the content window.
   D. Complete the TODO sections to create testing logic for two action methods.
   E. Click **Save**.

3. Test your new unit test logic.
   A. Click **Run Test**.
   B. Switch to the **Tests** tab and wait for the test to complete. You should see it pass with a green checkmark, no failures, and 3 total tests. If it does not, then look at the results, correct the problem, and re-run the test until it completes successfully.
   C. Type `CTRL + ALT + /` to close all open windows.
   D. Close the Developer Console.

20-3: Write Unit Tests for Getters and Setters

Goal:

Test whether the getter method in the controller extension returns results including the string "Attendees: ".

Tasks:

1. Examine the controller extension code for the Technician Status Page.
2. Create unit test methods to test the getAttendeeList getter method.
3. Test your new unit test logic.

Time:

20 minutes

Instructions:

1. Examine the controller extension code for the Technician Status Page.
   A. In the Salesforce user interface, in the top right-hand corner, click on **your name** and then click **Developer Console**.
   B. Click **File | Open**.
   C. Select **Classes** as the Entity Type and **TechnicianStatus_CX** as the Entity and click **Open**.
   D. Examine the code for the getAttendeeList getter method.

2. Create unit test methods to test the two custom action methods.
   A. Click **File | Open**.
   B. Select **Classes** as the Entity Type and **TechnicianStatus_Test** as the Entity and click **Open**.
   C. Delete the default code and copy the contents of the TechnicianStatus_Test.txt file located in the Exercises folder into the content window.
   D. Complete the TODO sections to create testing logic for testing the getAttendeeList getter method.
   E. Click **Save**.

3. Test your new unit test logic.
   A. Click **Run Test**.
   B. Switch to the **Tests** tab and wait for the test to complete. You should see it pass with a green checkmark, no failures, and 4 total tests. If it does not, then look at the results, correct the problem, and re-run the test until it completes successfully.
   C. Type `CTRL + ALT + /` to close all open windows.
   D. Close the Developer Console.

Certification App Object Design