est Apex Triggers

**Learning Objectives**

After completing this unit, you'll be able to:

* Write a test for a trigger that fires on a single record operation.
* Execute all test methods in a class.

**Test Apex Triggers**

Before deploying a trigger, write unit tests to perform the actions that fire the trigger and verify expected results.

Let’s test a trigger that we worked with earlier in the Writing Apex Triggers unit. If an account record has related opportunities, the AccountDeletion trigger prevents the record’s deletion.

**Prerequisites**

1. If you haven’t yet added the AccountDeletion trigger, follow these steps.
   1. In the Developer Console, click **File** | **New** | **Apex Trigger**.
   2. Enter AccountDeletion for the trigger name, and then select **Account** for the sObject. Click **Submit**.
   3. Replace the default code with the following.
   4. trigger AccountDeletion on Account (before delete) {
   5. // Prevent the deletion of accounts if they have related opportunities.
   6. for (Account a : [SELECT Id FROM Account
   7. WHERE Id IN (SELECT AccountId FROM Opportunity) AND
   8. Id IN :Trigger.old]) {
   9. Trigger.oldMap.**get**(a.Id).**addError**(
   10. 'Cannot delete account with related opportunities.');
   11. }

}

Copy

1. If you added the AccountDeletion trigger in a previous unit but disabled it so that the system could check your challenge, re-enable it.
   1. From Setup, search for Apex Triggers.
   2. On the Apex Triggers page, click **Edit** next to the AccountDeletion trigger.
   3. Select **Is Active**.
   4. Click **Save**.
2. If your org contains triggers from a previous unit called AddRelatedRecord, CalloutTrigger, or HelloWorldTrigger, disable them. For example, to disable the AddRelatedRecord trigger:
   1. From Setup, search for Apex Triggers.
   2. On the Apex Triggers page, click **Edit** next to the AddRelatedRecord trigger.
   3. Deselect **Is Active**.
   4. Click **Save**.
3. To disable the HelloWorldTrigger and CalloutTrigger triggers, repeat the previous steps.

**Adding and Running a Unit Test**

First, let’s start by adding a test method. This test method verifies what the trigger is designed to do (the positive case): preventing an account from being deleted if it has related opportunities.

1. In the Developer Console, click **File** | **New** | **Apex Class**.
2. Enter TestAccountDeletion for the class name, and then click **OK**.
3. Replace the default class body with the following.
4. @isTest
5. private class TestAccountDeletion {
6. @isTest static void TestDeleteAccountWithOneOpportunity() {
7. // Test data setup
8. // Create an account with an opportunity, and then try to delete it
9. Account acct = new Account(Name='Test Account');
10. insert acct;
11. Opportunity opp = new Opportunity(Name=acct.Name + ' Opportunity',
12. StageName='Prospecting',
13. CloseDate=System.**today**().**addMonths**(1),
14. AccountId=acct.Id);
15. insert opp;
16. // Perform test
17. Test.**startTest**();
18. Database.DeleteResult result = Database.**delete**(acct, false);
19. Test.**stopTest**();
20. // Verify
21. // In this case the deletion should have been stopped by the trigger,
22. // so verify that we got back an error.
23. System.assert(!result.**isSuccess**());
24. System.assert(result.**getErrors**().**size**() > 0);
25. System.**assertEquals**('Cannot delete account with related opportunities.',
26. result.**getErrors**()[0].**getMessage**());
27. }

}

Copy

The test method first sets up a test account with an opportunity. Next, it deletes the test account, which fires the AccountDeletion trigger. The test method verifies that the trigger prevented the deletion of the test account by checking the return value of the Database.delete() call. The return value is a Database.DeleteResult object that contains information about the delete operation. The test method verifies that the deletion was not successful and verifies the error message obtained.

1. To run this test, click **Test** | **New Run**.
2. Under Test Classes, click **TestAccountDeletion**.
3. To add all the methods in the TestAccountDeletion class to the test run, click **Add Selected**.
4. Click **Run**.Find the test result in the Tests tab under the latest run.

The TestAccountDeletion test class contains only one test method, which tests for a single account record. Also, this test is for the positive case. Always test for more scenarios to ensure that the trigger works in all cases, including deleting an account without opportunities and bulk account deletions.

Test data is set up inside the test method, which can be time-consuming as you add more test methods. If you have many test methods, put test-data creation in a test utility class and call the utility class from multiple test methods. The next unit shows you how to take advantage of a test utility class and add more test methods.

**Tell Me More**

The test method contains the Test.startTest() and Test.stopTest() method pair, which delimits a block of code that gets a fresh set of governor limits. In this test, test-data setup uses two DML statements before the test is performed. To test that Apex code runs within governor limits, isolate data setup’s limit usage from your test’s. To isolate the data setup process’s limit usage, enclose the test call within the Test.startTest() and Test.stopTest() block. Also use this test block when testing asynchronous Apex. For more information, see [Using Limits, startTest, and stopTest](https://developer.salesforce.com/docs/atlas.en-us.224.0.apexcode.meta/apexcode/apex_testing_tools_start_stop_test.htm).

Note

**Note**

A known issue with the Developer Console prevents it from updating code coverage correctly when running a subset of tests. To update your code coverage results, use **Test** | **Run All** rather than **Test** | **New Run**.

**Resources**

**Documentation**

Check out the following in the Apex Developer Guide.

* [Understanding Testing in Apex](https://developer.salesforce.com/docs/atlas.en-us.224.0.apexcode.meta/apexcode/apex_testing_intro.htm)
* [Triggers](https://developer.salesforce.com/docs/atlas.en-us.224.0.apexcode.meta/apexcode/apex_triggers.htm)

Top of Form

**Hands-on Challenge**

**+500 points**

**GET READY**

You’ll be completing this unit in your own hands-on org. Click **Launch** to get started, or click the name of your org to choose a different one.

If you use Trailhead in a language other than English, make sure that your hands-on org is set to the same language as the challenge instructions. Otherwise you may run into issues passing this unit. Want to find out more about using hands-on orgs on Trailhead? Check out [Trailhead Playground Management](https://trailhead.salesforce.com/en/content/learn/modules/trailhead_playground_management).

**YOUR CHALLENGE**

**Create a Unit Test for a Simple Apex Trigger**

Create and install a simple Apex trigger which blocks inserts and updates to any contact with a last name of 'INVALIDNAME'. You'll copy the code for the class from GitHub. Then write unit tests that achieve 100% code coverage.

* Create an Apex trigger on the Contact object
  + Name: RestrictContactByName
  + Code: [**Copy from GitHub**](https://github.com/developerforce/trailhead-code-samples/blob/master/RestrictContactByName.cls)
* Place the unit tests in a separate test class
  + Name: TestRestrictContactByName
  + Goal: 100% test coverage
* Run your test class at least once

**swamy2**

Last used on 6/27/2022

Bottom of Form