1. Apex Testing
2. Test Apex Triggers

Test 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.
   a. In the Developer Console, click **File** | **New** | **Apex Trigger**.
   b. Enter `AccountDeletion` for the trigger name, and then select **Account** for the sObject. Click **Submit**.
   c. Replace the default code with the following.

```
trigger AccountDeletion on Account (before delete) {

    // Prevent the deletion of accounts if they have related contacts.
    for (Account a : [SELECT Id FROM Account
                     WHERE Id IN (SELECT AccountId FROM Opportunity) AND
```

```
                             Id IN :Trigger.old]) {
            Trigger.oldMap.get(a.Id).addError(
                'Cannot delete account with related opportunities.');
    }

}
```

Copy

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

```
@isTest
private class TestAccountDeletion {
    @isTest static void TestDeleteAccountWithOneOpportunity() {
        // Test data setup
        // Create an account with an opportunity, and then try to delete it
        Account acct = new Account(Name='Test Account');
        insert acct;
        Opportunity opp = new Opportunity(Name=acct.Name + ' Opportunity',
                                       StageName='Prospecting',
                                       CloseDate=System.today().addMonths(1),
                                       AccountId=acct.Id);
        insert opp;

        // Perform test
        Test.startTest();
        Database.DeleteResult result = Database.delete(acct, false);
        Test.stopTest();
        // Verify
        // In this case the deletion should have been stopped by the trigger,
        // so verify that we got back an error.
        System.assert(!result.isSuccess());
        System.assert(result.getErrors().size() > 0);
        System.assertEquals('Cannot delete account with related opportunities.',
                             result.getErrors()[0].getMessage());
    }
    
}
```

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.

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
- Triggers

Get Ready

You'll be completing this challenge in your own personal Salesforce environment. Choose from the dropdown menu, then click **Launch** to get started. If you use Trailhead in a language other than English, set the language of your Trailhead Playground to English before you attempt this challenge. Want to find out more about using hands-on orgs for Trailhead learning? Check out the Trailhead Playground Management module.

Your Challenge

Create a unit test for a simple Apex trigger.
Install a simple Apex trigger, write unit tests that achieves 100% code coverage for the trigger, and run your Apex tests.

- The Apex trigger to test is called 'RestrictContactByName', and the code is available here. Copy and paste this trigger into your Developer Edition via the Developer Console.
- 'RestrictContactByName' is a trigger which blocks inserts and updates to any contact with a last name of 'INVALIDNAME'.
- The unit tests must be in a separate Apex class called 'TestRestrictContactByName'.
- The unit tests must cover scenarios for all lines of code included in the Apex trigger, resulting in 100% code coverage.
- Run your test class at least once (via 'Run All' tests the Developer Console) before attempting to verify this challenge.

My Trailhead Playground 3

Launch

Check challenge to earn 500 points