<https://www.youtube.com/watch?v=W7PBphhw5WA>

Create Test Data for Apex Tests

**Learning Objectives**

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

* Create a test utility class.
* Use a test utility method to set up test data for various test cases.
* Execute all test methods in a class.

**Create Test Data for Apex Tests**

Use test utility classes to add reusable methods for test data setup.

**Prerequisites**

Complete the prerequisites in the previous unit, [Testing Apex Triggers](https://trailhead.salesforce.com/force_com_dev_beginner/apex_testing/apex_testing_triggers), if you haven’t done so already.

**Adding a Test Utility Class**

Let’s refactor the previous test method by replacing test data creation with a call to a utility class method. First, you need to create the test utility class.

The TestDataFactory class is a special type of class—it is a public class that is annotated with isTest and can be accessed only from a running test. Test utility classes contain methods that can be called by test methods to perform useful tasks, such as setting up test data. Test utility classes are excluded from the org’s code size limit.

To add the TestDataFactory class:

1. In the Developer Console, click **File** | **New** | **Apex Class**, and enter TestDataFactory for the class name, and then click **OK**.
2. Replace the default class body with the following.
3. @isTest
4. public class TestDataFactory {
5. public static List<Account> **createAccountsWithOpps**(Integer numAccts, Integer numOppsPerAcct) {
6. List<Account> accts = new List<Account>();
7. for(Integer i=0;i<numAccts;i++) {
8. Account a = new Account(Name='TestAccount' + i);
9. accts.**add**(a);
10. }
11. insert accts;
12. List<Opportunity> opps = new List<Opportunity>();
13. for (Integer j=0;j<numAccts;j++) {
14. Account acct = accts[j];
15. // For each account just inserted, add opportunities
16. for (Integer k=0;k<numOppsPerAcct;k++) {
17. opps.**add**(new Opportunity(Name=acct.Name + ' Opportunity ' + k,
18. StageName='Prospecting',
19. CloseDate=System.**today**().**addMonths**(1),
20. AccountId=acct.Id));
21. }
22. }
23. // Insert all opportunities for all accounts.
24. insert opps;
25. return accts;
26. }

}

Copy

This test utility class contains one static method, createAccountsWithOpps(), which accepts the number of accounts (held in the numAccts parameter) and the number of related opportunities to create for each account (held in the numOppsPerAcct parameter). The first loop in the method creates the specified number of accounts and stores them in the accts list variable. After the first loop, the insert() DML statement is called to create all accounts in the list in the database.

The second loop creates the opportunities. Because each group of opportunities are linked to one account, the outer loop iterates through accounts and contains a nested loop that creates related opportunities for the current account. The next time the nested loop is run, opportunities are added to the same list using the add() method. Opportunities are linked to their parent accounts using the AccountId field. The total number of all opportunities that are created is the product of the number of opportunities with the number of accounts (numOppsPerAcct\*numAccts). Next, the insert() DML statement is efficiently called outside the loop to create all opportunities in the collection for all accounts in one call only.

Finally, this method returns a list of the new accounts.

Note

**Note**

Even though this method doesn’t return the related opportunities, you can get those records by writing a SOQL query that makes use of the existing relationship between Account and Opportunity, such as the query used in the trigger in [Testing Apex Triggers](https://developer.salesforce.com/en/trailhead/force_com_dev_beginner/apex_testing/apex_testing_triggers#Tdxn4tBK-heading2).

**Calling Utility Methods for Test Data Creation**

Now that you’ve added the test utility class, modify the test class to take advantage of this class. In the TestAccountDeletion class, replace the block that starts with // Test data setup and ends with insert opp; with:

// Test data setup

// Create one account with one opportunity by calling a utility method

Account[] accts = TestDataFactory.**createAccountsWithOpps**(1,1);

Copy

The array returned by the TestDataFactory.createAccountsWithOpps(1,1) call contains one Account sObject.

Here’s the modified test method. A shorter version!

@isTest

private class TestAccountDeletion {

@isTest static void TestDeleteAccountWithOneOpportunity() {

// Test data setup

// Create one account with one opportunity by calling a utility method

Account[] accts = TestDataFactory.**createAccountsWithOpps**(1,1);

// Perform test

Test.**startTest**();

Database.DeleteResult result = Database.**delete**(accts[0], false);

Test.**stopTest**();

// Verify that the deletion should have been stopped by the trigger,

// so check 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

**Testing for Different Conditions**

One test method is not enough to test all the possible inputs for the trigger. We need to test some other conditions, such as when an account without opportunities is deleted. We also need to test the same scenarios with a bulk number of records instead of just a single record. Here is an updated version of the test class that contains the three additional test methods. Save this updated version of the class.

@isTest

private class TestAccountDeletion {

@isTest static void TestDeleteAccountWithOneOpportunity() {

// Test data setup

// Create one account with one opportunity by calling a utility method

Account[] accts = TestDataFactory.**createAccountsWithOpps**(1,1);

// Perform test

Test.**startTest**();

Database.DeleteResult result = Database.**delete**(accts[0], false);

Test.**stopTest**();

// Verify that the deletion should have been stopped by the trigger,

// so check 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**());

}

@isTest static void TestDeleteAccountWithNoOpportunities() {

// Test data setup

// Create one account with no opportunities by calling a utility method

Account[] accts = TestDataFactory.**createAccountsWithOpps**(1,0);

// Perform test

Test.**startTest**();

Database.DeleteResult result = Database.**delete**(accts[0], false);

Test.**stopTest**();

// Verify that the deletion was successful

System.assert(result.**isSuccess**());

}

@isTest static void TestDeleteBulkAccountsWithOneOpportunity() {

// Test data setup

// Create accounts with one opportunity each by calling a utility method

Account[] accts = TestDataFactory.**createAccountsWithOpps**(200,1);

// Perform test

Test.**startTest**();

Database.DeleteResult[] results = Database.**delete**(accts, false);

Test.**stopTest**();

// Verify for each record.

// In this case the deletion should have been stopped by the trigger,

// so check that we got back an error.

for(Database.DeleteResult dr : results) {

System.assert(!dr.**isSuccess**());

System.assert(dr.**getErrors**().**size**() > 0);

System.**assertEquals**('Cannot delete account with related opportunities.',

dr.**getErrors**()[0].**getMessage**());

}

}

@isTest static void TestDeleteBulkAccountsWithNoOpportunities() {

// Test data setup

// Create accounts with no opportunities by calling a utility method

Account[] accts = TestDataFactory.**createAccountsWithOpps**(200,0);

// Perform test

Test.**startTest**();

Database.DeleteResult[] results = Database.**delete**(accts, false);

Test.**stopTest**();

// For each record, verify that the deletion was successful

for(Database.DeleteResult dr : results) {

System.assert(dr.**isSuccess**());

}

}

}

Copy

**Running All Test Methods**

The final step is to run the test methods in our test class, now that the class contains more comprehensive tests and has been refactored to use a test data factory. Since you’ve already run the tests in the TestAccountDeletion class, you can just rerun this test class to run all its test methods.

1. To execute the same test run, click the **Tests** tab, select your test run, and then click **Test** | **Rerun**.
2. Check the results in the Tests tab by expanding the latest test run. The test run should report that all four tests passed!

**Resources**

**Documentation**

[*Apex Developer Guide*: Common Test Utility Classes for Test Data Creation](https://developer.salesforce.com/docs/atlas.en-us.224.0.apexcode.meta/apexcode/apex_testing_utility_classes.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 Contact Test Factory**

Create an Apex class that returns a list of contacts based on two incoming parameters: the number of contacts to generate and the last name. Do not insert the generated contact records into the database.  
  
NOTE: For the purposes of verifying this hands-on challenge, don't specify the @isTest annotation for either the class or the method, even though it's usually required.

* Create an Apex class in the public scope
  + Name: RandomContactFactory (without the @isTest annotation)
* Use a Public Static Method to consistently generate contacts with unique first names based on the iterated number in the format Test 1, Test 2 and so on.
  + Method Name: generateRandomContacts (without the @isTest annotation)
  + Parameter 1: An integer that controls the number of contacts being generated with unique first names
  + Parameter 2: A string containing the last name of the contacts
  + Return Type: List < Contact >

**swamy2**

Last used on 6/27/2022

[Launch](https://trailhead.salesforce.com/launch_org/9460915)

Check Challenge to Earn 500 Points

Bottom of Form