**Build Apex Coding Skills | Salesforce Trailhead**

**Example 1: Apex Trigger & Classes**

Create a trigger that will prevent users from deleting the Accounts. This is because the client only wants the System Administrator to have the permissions to delete accounts.

---------------------------------------------------------------

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

// following code implements the business logic required to achieve the goal

        for(Account Acc:trigger.old)
        {
                acc.adderror('Unable to delete Accounts, please contact Admin..');
        }

}
```

*******************************************************

Test out your code by creating and deleting an account through the Anonymous window (Ctrl + e), (Click on 'Debug' in the top bar, then click on 'Open Execute Anonymous Window'):

Write the below code in the window, and click on Execute:

```
Account a = new Account(Name = 'Test Account');
Insert a;
Delete a;
```

You will get an error like this:

```
Line: 4, Column: 1
System.DmlException: Delete failed. First exception on row 0 with id 0010I00002SP1EcQAL; first error:
FIELD_CUSTOM_VALIDATION_EXCEPTION, Unable to delete Accounts, please contact Admin.: []
```

You can also try to delete any Account through UI.

**Example 2:**

Write a trigger so that when the leads are inserted into the database it would add 'Doctor' prefixed for all lead names before they are saved. This is applicable for both inserting and updating the lead records.

---------------------------------------------------------------

```
trigger addDrPrefixToLead on Lead (before insert) {

    List<Lead> leadList = Trigger.New;

    for(Lead leadRecord: leadList) {

        leadRecord.LastName = 'Dr. '+leadRecord.LastName;

        System.debug('Lead record: '+leadRecord);

    }

}
```

*******************************************************

You can test the above code in the same way as the first program, using the Anonymous Window (Ctrl + e) or through UI.

Try to create a Lead account through UI and you will see that Lead name gets prefixed with 'Dr.' automatically.

In real life scenarios you would never write business logic within your trigger code, rather always call a helper class, which would implement the logic required and then return the results. We'll see examples of it soon.

This is done because you might have multiple triggers on the same object, and there's no way of knowing which trigger Apex would fire first.

However, for the sake of learning and simplicity, we're considering this.

**Example 3:**

Create an Apex method that receives 2 input parameters, name and phone, and creates an account using these parameters. Please practice executing classes like these on your own from Anonymous Window (Ctrl + e).

-----------------------------------------------------------------

```
public class createAccUsingNameAndPhone {

    public void acceptNameAndPhone(String Name, String Phone){

        Account a = new Account();
        a.Name = Name;
        a.Phone = Phone;

        insert a;

        System.debug(a);

    }

}
```

*****************************************************

In order to execute the above class you need to create an instance of the class and call the method using that instance, and pass the parameters in the call. Here's how:

```
createAccUsingNameAndPhone acc = new createAccUsingNameAndPhone ();

acc.acceptNameAndPhone('Johnson and Johnsons','1234567890');
```

Make sure the 'Open Log' checkbox is checked and then click on Execute. And check out the result in the Log that gets opened automatically, by clicking on 'Debug only'.

In real life scenarios, classes like the above are called through different ways, such as trigger, or custom buttons, or visualforce pages, etc.

**Example 4:**

Create a class to search for an Account record using Phone number as a parameter.

-----------------------------------------------------------------

```
public class searchAccUsingPhNum {

    public void recievePhoneReturnAccount(String phone){

        Account a = [select name from Account where Phone =: phone];

        system.debug(a);

    }

}
```

*****************************************************

Execute the above class in the anonymous window (Ctrl + e):

```
searchAccUsingPhNum s = new searchAccUsingPhNum();

s.recievePhoneReturnAccount('1234567890');
```

Check out the result in the log by clicking on the ‘Debug only’ check box.

**Example 5:**

Create a trigger on Contact that will update contact’s description based on Trigger context (before context, after context).
Update the description to include the User’s name using the userInfo Keyword.
Example - (if update) ‘Last modified by Dhiraj’ or (if insert) ‘Last created by Dhiraj’

--------------------------------------------------------------

```
trigger updateContactOwner on Contact (before insert, before update) {

    if(trigger.isinsert) {

        for(Contact c: Trigger.new) {

           c.Description = 'Last Created By - ' + userInfo.getFirstName();

           }

    }

    if(trigger.isupdate) {

        for(Contact c: Trigger.new) {

           c.Description = 'Last Updated By - ' + userInfo.getFirstName();

           }

    }

}
```

*******************************************************

You can test this out easily in UI, by creating or updating a Contact record and then verifying the description of the record.
Keep in mind that this trigger replaces any description that you enter while creating or updating the record.

**Example 6:**

Create a trigger that will replace the word ‘and’ with the character ‘&’ (amperson) whenever an account is created or updated. Use a helper apex class.

--------------------------------------------------------------

**Class**:

```
public class replaceAndWithAmperson {

    public void replacing(List<Account> accs){

        for(Account a: accs){

            if((a.Name).contains('and'))
            {
                a.Name = (a.Name).replace('and','&');
            }

        }

    }

}
```

**Trigger**:

```
trigger replacingAndwithAmperson on Account (before insert, before update) {

    List<Account> accsList = Trigger.new;

    replaceAndWithAmperson r = new replaceAndWithAmperson();
    r.replacing(accsList);

}
```

*******************************************************

You can test this out easily in UI, by updating any account record that has ‘and’ in its name. Or you can also try creating a new account record with ‘and’ in its name.

**Example 7:**

Create a class that creates an account by receiving its name as a parameter and also creates a related Contact and an Opportunity.

-------------------------------------------------------------

```
public class createAccConOpp {
    public void createAcc(String accName){

        Account acc = new Account(Name = accName);

        insert acc;

        Contact c = new Contact(firstname = 'John', lastname = 'Doe', AccountId = acc.id);

        insert c;

        Date closeD = Date.newinstance(2020, 4, 25);

        Opportunity opp = new Opportunity(Name = '50 Laptops', CloseDate = closeD,
stagename = 'Prospecting', AccountId = acc.id);

        insert opp;

    }

}
```

******************************************************

You can test this easily with Anonymous Window, same as previous classes. And then see the results in UI.

```
createAccConOpp c = new createAccConOpp();
c.createAcc('John Doe Company');
```

**Example 8:**

Create a trigger that prevents a contact from being deleted if it is associated with an account.

---------------------------------------------------------------

```
trigger preventAccAssociatedContDeletion on Contact (before delete) {

    Set<ID> consID = Trigger.oldMap.keySet();

    List<Contact> cons = [SELECT Account.Name from Contact where ID IN: consID];
    for(Contact c: cons){

        if(c.AccountId != NULL)
        {
            Trigger.old[0].addError('Cannot delete this contact record since it is associated
with an Account.');
        }

    }

}
```

******************************************************

Test it out by simply trying to delete a Contact from UI that has an associated account.

**Example 9:**

Create a trigger that will send out an email to the email mentioned in the Contact record whenever the contact has been created or edited.

-------------------------------------------------------------

```
trigger sendEmailToContact on Contact (before insert, before update) {

    String subject;

    if(trigger.isinsert) {
        subject = 'Created';
    }
    if(trigger.isupdate) {
        subject = 'Updated';
    }

    /* You don't have to remember any of these methods below. Copy pasting every time
is fine. */

    for(Contact c: Trigger.new)
    {
        if(c.email != null)
        {
            Messaging.SingleEmailMessage mail = new Messaging.SingleEmailMessage();
                        String[] toAddresses = new String[] {c.email};
                        mail.setToAddresses(toAddresses);
                        mail.setSubject('Your Contact was '+subject);
                        mail.setPlainTextBody('Have a great day.');
                        Messaging.sendEmail(new Messaging.SingleEmailMessage[] { mail });
        }
    }

}
```

*****************************************************

Test it out by simply creating or editing a Contact record from UI and enter your email in the email field. And after clicking Save, simply check your email.

**Example 9:**

The following Trigger will fire when we try to create an account with same name i.e. Preventing the users to create Duplicate Accounts.

-------------------------------------------------------------------

```
trigger preventDuplicateAccount on Account (before insert) {

Set<String> accNames = new Set<String>();

    for(Account acc: Trigger.new)
    {
                accNames.add(acc.Name);
    }
    List<Account> listAcc = new List<Account>();

    listAcc = [select Name from Account where Name IN: accNames];

    if(listAcc.size()>0)
    {
        Trigger.new[0].addError(' Duplicate Account name already exists...');
    }

    system.debug(listAcc);
}
```

************************************************************

Test it in UI by creating an Account record of the same name as one that already exists..

**Example 10:**

Create a trigger that will add an Opportunity to an Account that doesn't have any opportunity already, whenever it is updated or created. If possible, try it once again with a helper class.

-----------------------------------------------------------------

```
trigger createOppOnAccount on Account (after insert, after update) {

    List<Account> accountIds = Trigger.new;
    List<Account> access List = new List<Account>();
    List<Opportunity> oppList = new List<Opportunity>();

    accsList = [Select Id, name, (select Id,name from opportunities) from account where
Id in:accountIds];
```

```
    for(Account a: Trigger.new){
        if (a.Opportunities.size() == 0) {

            Opportunity o = new Opportunity(
                                                Name = a.Name + ' - Free Demo Product',
                                                StageName = 'Prospecting',
                                                CloseDate = System.today().addMonths(1),
                                                AccountId = a.Id
                                               );
            oppList.add(o);
        }
    }

    if(oppList.size() > 0) {
        insert oppList;
    }
}
```

************************************************************

Test it by creating or editing an Account via UI.

**Example 11:**

Create a Custom field called “Number of Locations” on the Account Object (DataType = Number, Length = 1, Decimal = 0)

Write a trigger that creates the number of contacts, when an Account is created, that are equal to the number in the ‘Number of Locations’ field on the Account Object. If possible, try it once again with a helper class.

-----------------------------------------------------------------

```
trigger creatingRelatedContacts on Account (after insert) {


    List<Contact> listOfListCon = new List<Contact>();


                    for(Account acc:Trigger.new) {


            List<Contact> listCon = new List<Contact>();
            for(Integer i = 0; i < acc.Number_of_Locations__c; i++) {

        Contact c = new Contact(Lastname = acc.Name+' - Contact '+i, AccountId = acc.Id);
        listCon.add(c);
            }
                    listOfListCon.addAll(listcon);
// Addall method appends all the elements of the listCon list, to the listOfListCon list.
   }
  if(listOfListCon.size() > 0) {
       insert listOfListCon;
  }
}
```

*************************************************************
Test it by creating an Account record via UI and don't forget to enter a value in the 'Number of Locations' field.

**Example 12: Test Classes**

Write a trigger so that when a new Account is created it will create a contact related to that account.

```
trigger AccountTriggerwithTest on Account (after insert) {
   // Check for the context variables which would tell us that
   // code should run only for “After Insert”
   if(Trigger.isAfter && Trigger.isInsert) {
       // Declare a List of Contacts.
       List<Contact> contacts = new List<Contact>();

       // Loop for each account which was inserted.
       for(Account account : Trigger.new) {
           // Add the contact which needs to be inserted in the list of Contacts.
           Contact newContact = new Contact(LastName = account.Name+'Con',
AccountId=account.Id);
           contacts.add(newContact);
       }
       // Now insert all the contacts together.
       insert contacts;
   }
}
```

**Test Class**: We will insert an account and then verify if the contact for that account has been created or not.

```
@isTest
private class AccountTriggerTest {
   @isTest
   static void accountTrigger_afterInsertTest() {
       // Create an instance of Account
       Account account = new Account(Name='TestAccount1', Type='Prospect');

       // Perform test
       Test.startTest();
           insert account;
       Test.stopTest();

       // Fetch the account that we have inserted above.
       Account insertedAccount = [Select Id, Name From Account LIMIT 1];

       // Fetch the Contact for the above account and assert that a contact should
be created
       // also we are verifying the Account Name
       List<Contact> insertedContacts = [Select Id, Account.Name From Contact];
       System.assertEquals(1, insertedContacts.size(), 'Only 1 Contact should
present');
       System.assertEquals(insertedAccount.Name, insertedContacts[0].Account.Name,
                           'Account Name does not match');
   }
}
```

## Example 13:

Write a trigger so that if there is any associated Contact to an Account, and User tries to delete the Account, the User should get the error that Account with associated Contact can not be deleted.

```
trigger AccountTriggerwithTest2 on Account (before delete) {
   // Condition to check that the code should run only before the account is being
deleted
   if(Trigger.isBefore && Trigger.isDelete) {
       // List of accounts with associated Contacts
       Map<Id, Account> mapOfAccounts = new Map<Id, Account> ([Select Id, (Select
Id From Contacts)
                                                              From Account
                                                              Where Id IN
:Trigger.oldMap.keySet()]);

       // Condition to check if any Contact is associated for each account
       // If yes, then throw the error
       for(Account acc : Trigger.old) {
           if(!mapOfAccounts.get(acc.Id).Contacts.isEmpty()) {
               acc.addError('Account with associated Contact(s) can not be
deleted.');
           }
       }
   }
}
```

```
@isTest
private class AccountTriggerTest2 {
    // Data method which insert two accounts
    // first account with 1 associated contact
    // second account with no associated contact
    @testSetup
    static void dataSetup() {
        Account acc1 = new Account(Name='TestAccount1');
        insert acc1;
        Contact con = new Contact(LastName='TestContact', AccountId=acc1.Id);
        insert con;

        Account acc2 = new Account(Name='TestAccount2');
        insert acc2;
    }

    // Test method to check if there is any associated contact with an account then
the account should not be deleted.
     @isTest
    static void beforeDeleteTest_accountWithContact() {
        List<Account> insertedAccount = [Select Id From Account Where
Name='TestAccount1'];
        System.assertEquals(1, insertedAccount.size(), 'List should have only 1
account.');

        Test.startTest();
            try {
                delete insertedAccount;
            }
            catch(DMLException e) {
                System.assert(e.getMessage().contains('Account with associated
Contact(s) can not be deleted'),
                                'Account with associated contact should not be
deleted.');
                System.assertEquals(1, [Select Id From Account Where
Name='TestAccount1'].size(),
                                       'Account with associated contact should not
be deleted.');
            }
        Test.stopTest();
    }

    // Test method to check if there is no associated contact with an account then
the account can be deleted.
    @isTest
    static void beforeDeleteTest_accountWithoutContact() {
        List<Account> insertedAccount2 = [Select Id From Account Where
Name='TestAccount2'];
        System.assertEquals(1, insertedAccount2.size(), 'List should have only 1
account.');

        Test.startTest();
            try {
                delete insertedAccount2;
            }
            catch(DMLException e) {
                System.assertEquals(0, [Select Id From Account Where
Name='TestAccount1'].size(),
                                       'Account without associated contact can be
deleted.');
            }
        Test.stopTest();
    }
}
```