
# Apex

**(Implementing Triggers)**

Exercise Guide


salesforce

**Table of Contents**

**Exercise 5-1: Creating an Automated Chatter Subscription (Part 2)**

**Goal:**

Use a trigger to automate the following of an object with Chatter.

**Scenario:**

Hiring managers at Universal Containers want to follow positions for which they are responsible via Chatter. They would like this to happen automatically.

**Tasks:**

1. Write a trigger to create a subscription for a hiring manager when a `Position__c` sObject is created.
2. Test the trigger by creating a new position to see if the hiring manager for that position is subscribed.

**Time:**

15 minutes

---

**Instructions:**

1. Write a trigger to create a subscription for a hiring manager when a `Position__c` sObject is created.
   A. In the Force.com IDE, right-click the project folder in the Package Explorer pane and select **New | Apex Trigger**.
      i. **Name**: `PositionInsertTrigger`
      ii. **Object**: `Position__c`
      iii. **Apex Trigger Operations**: `after insert`
      iv. Click **Finish**.
   B. Write the code to call the `HiringManagerSubscribeNewPosition` method of the `SubscriptionsClass` class. Make sure to pass it the appropriate trigger value as an argument.
   C. Save it.
2. Test the trigger by creating a new position and see if the hiring manager for that position is subscribed.
   A. In Salesforce, create a new position with Ben Stuart as the hiring manager. Notice that when the record is saved, it shows Ben Stuart as a follower on the page.

B. Determine if the user is seeing the post:

   i. Go to **Setup | Administer | Manage Users | Users**, log in as the Ben Stuart, the hiring manager for the position for which the job application was created.

   ii. Notice there are no Chatter posts for Ben Stuart!

   NOTE: Ben Stuart does not currently have permission to view the newly created position, and hence cannot see the related Chatter post. This will be addressed in a later exercise.

---

**Review:**

1. If you were to write automated testing for this trigger and the associated `SubscriptionsClass` class, what cases would you want to check?

2. Why are the trigger and the associated class designed to handle a list of objects rather than a single object?

3. The current scenario does not deal with the possibility that a hiring manager for a position might change. What actions would you want to take in relation to Chatter objects if such a change was detected?

## Exercise 5-2: Managing Access to Position Objects via Sharing

### Goal:

Explore `Trigger` class functionality, sharing records, and SOQL `for` loops.

### Scenario:

1. Universal Containers has rules that need to be implemented concerning who can edit and view positions. If a position is open and approved, the hiring manager should have read/write access and the rest of the organization should have read-only access. Otherwise the hiring manager has read-only access and the rest of the organization cannot see the position at all.
2. The company wants to be sure that if the hiring manager is changed on a position, the access for the old hiring manager is removed and access for the new hiring manager is added.

### Tasks:

1. Set up the Apex sharing reasons for position access (if not present).
2. Set up the criteria based sharing rules to handle organization wide access.
3. Upload `PositionSharingClass`.
4. Upload and complete `PositionSharingTrigger`.
5. Test the trigger and class.

### Time:

50 minutes

---

### Instructions:

1. Setup the Apex sharing reasons for position access (if not present).
   A. In Salesforce, navigate to **Setup | Build | Create | Objects | Position**.
   B. In the Apex Sharing Reasons related list, if the following reason does not exist, create it by clicking **New**.
      i. **Label:** `Hiring Manager`
      ii. **Name:** `Hiring_Manager`
      iii. Click **Save**.
2. Set up the criteria based sharing rules to handle organization wide access.
   A. Navigate to **Setup | Administer | Security Controls | Sharing Settings**, and choose `Position` from the dropdown menu labeled **Manage sharing settings for:**.

B. Under Position Sharing Rules click **New**.

i. **Rule Label:** `Position Sharing Rule`

ii. **Rule Type:** `Based on criteria`

iii. **Rule entry 1:**
   a. **Field:** `Status`
   b. **Operator:** `equals`
   c. **Value:** `Open`

iv. **Rule entry 2:**
   a. **Field:** `Sub-Status`
   b. **Operator:** `equals`
   c. **Value:** `Approved`

v. **Share with:** `Public Groups All Internal Users`

vi. **Access Level:** `Read Only`

vii. Click **Save**.

viii. In the dialog box, click **OK**.

3. Upload `PositionSharingClass`.

A. In the Force.com IDE, right-click the project folder and select **New | Apex Class**.

i. **Name:** `PositionSharingClass`

ii. Click **Finish**.

B. Copy the code from the `5-2.PositionSharingClass.txt` file in the `Exercises` folder.

C. Save the class.

4. Upload and complete `PositionSharingTrigger`.

A. In the Force.com IDE, right-click the project folder in the Package Explorer pane and select **New | Apex Trigger**.

i. **Name**: `PositionSharingTrigger`

ii. **Object**: `Position__c`

iii. **Apex Trigger Operations**: `after insert, after update`

iv. Click **Finish**.

B. Copy the code from the `5-2.PositionSharingTrigger.txt` file in the `Exercises` folder, paste it into Eclipse (replacing any default text), and complete the `TODO:` sections.

Note: The code you are copying has the same header as the code Eclipse generated for you, indicating the sObject for which the trigger is designed and the operations it should perform. Once a trigger is created, you cannot modify which sObject the trigger is associated with, but you can modify the events which activate the trigger.

C. Save the trigger.

5. Test the trigger and class.
   A. Log in to your org and create and save a new position.
   B. Click **Sharing** and view **User and Group Sharing** to ensure that the hiring manager was automatically added with the appropriate permissions.
   C. Modify the position, setting the **Hiring Manager** to another user in the org and changing the status to `open` and sub-status to `approved`.
   D. View **User and Group Sharing** to ensure that the hiring manager was automatically added and given the appropriate permissions, and that org-wide read privileges were set for all users.
   E. Modify the position, setting the status to `Closed`.
   F. View **User and Group Sharing** to ensure that the hiring manager now has only read permission, and org-wide privileges for all users have been removed.

---

**Review:**

1. Why did we have you create the class before creating the trigger?

2. Sharing sObjects can be considered junctions combining two different types of sObjects. What field in a `sharing` sObject refers to the sObject to which the sharing is being applied? What field refers to the other side of the junction, and what are the types of sObjects to which it can refer?

3. Universal Containers wants hiring managers to be aware of when job applications come in for their positions, as well as keeping them apprised of changes to the status of those applications. The company would also like the managers to be able to view the job applications. How different would this implementation be from the position triggers for setting up subscriptions and sharing rules? Would you need one trigger for this, or do you need two?

4. Sharing rules are first set up using the Force.com declarative tools, but as seen in this example, sometimes these need to be augmented by a programmatic solution. How would your company document that the rules are defined in multiple places?

5. In Exercise 5-1, Ben Stuart, the hiring manager, could not see his Chatter subscription because he did not have permission to see the position created. If you logged onto the system as Ben Stuart now, would you be able to see the subscription? Why or why not?

## Exercise 5-3: Applying Best Practices for Bulkification and Limits in Triggers

### Goal:

1. Learn to identify inefficient and incorrect code related to triggers and limits.
2. Apply best practices regarding bulkification and limits.

### Scenario:

1. A Universal Containers developer new to the Force.com platform was given the task of creating a trigger that posts a Chatter notification to the entire organization when a position moves to the open/approved state, so that everyone in the company has the opportunity to recommend people and earn the hiring bonus.
2. You have been asked to review the code and help him debug it and improve it so that it uses best practices.

### Tasks:

1. Upload and analyze `PositionAnnouncementTrigger`.
2. Make modifications to `PositionAnnouncementTrigger` based on your analysis in the first task.
3. Test the trigger.

### Time:

50 minutes

---

### Instructions:

1. Upload and analyze `PositionAnnouncementTrigger`.
   A. In the Force.com IDE, right-click the project folder in the Package Explorer pane and select **New | Apex Trigger**.
      i. **Name**: `PositionAnnouncementTrigger`
      ii. **Trigger Object**: `Position__c`
      iii. **Trigger Operations**: `after insert, after update`
   B. Click **Finish**.
   C. Copy the code from the `5-3.PositionAnnouncementTrigger.txt`file in the `Exercises`folder and paste it into Eclipse (replacing any default text).
   D. Save the trigger.
   E. Answer the following questions regarding the trigger you just uploaded:
      i. Engineering has received authorization for 40 new positions. An HR employee is going to upload the approved positions from a spreadsheet rather than entering them

individually. The code in `PositionAnnouncementTrigger` will not work correctly in this case. What basic rule of triggers is this code violating?

ii. When the code is fixed, you will potentially have multiple `FeedItem`objects that need to be inserted into the database. This course describes two potential ways of doing this. What is the worst way to do this? And why is it problematic?

iii. What is the best pattern?

iv. What code have you already worked on that you can use as a model for fixing this code?

2. Make modifications to `PositionAnnouncementTrigger`based on your analysis in the first task.
   A. Modify the trigger to handle multiple incoming `Position__c` sObjects.
   B. Make sure that the trigger uses only one `insert`statement when saving `FeedItem` objects.
3. Test the trigger.
   A. Log in to your Salesforce org and create a new position, with **Status** of `Open` and a **Substatus** of `Approved`.
   B. Go to **Setup | Administer | Manage Users | Users** and click **login** next to a user other than yourself.
   C. Verify that the user has received an announcement about the position in a Chatter post.
   D. In the Force.com IDE, in the **Source to Execute** text area of the Execute Anonymous tab**:**
      i. Remove any code that is currently in place.
      ii. Copy the code from the `5-3.TestPositionAnnouncement.txt`file in the `Exercises`folder and paste it into the Execute Anonymous tab in Eclipse (replacing any existing text).
      iii. Click **Execute Anonymous**.

**Review:**

1. Beyond DML limits, about what other limits do you need to be concerned when writing `insert`or `update` statements?

2. The code you executed to test your changes creates 20 positions. Do these positions now exist in your org? If so, what user created them?

## Exercise 5-4: Determining to Use Declarative or Programmatic Solutions

**Goal:**

Given a scenario, determine whether to use a declarative or programmatic solution, and provide a high-level overview of that solution.

**Scenario:**

Universal Containers has a few customizations it would like to develop and are trying to determine the best ways to implement them.

**Tasks:**

Analyze the given scenario and answer the questions regarding implementing a solution.

**Time:**

20 minutes

---

**Instructions:**

1. Analyze the given scenario and answer the questions regarding implementing a solution.

### Scenario A: Preventing Duplicate Candidates

Universal Containers is having problems with duplicate records for candidates entering the database. The company has determined that the combination of a candidate's last name and email address is enough to ensure uniqueness within the system, and it wants to prevent records being saved that would match in this last name and email address combination.

For this scenario, would you use a declarative or programmatic solution? What are the steps of the high-level solution?

### Scenario B: Restricting the Position-Salary Relationship to One-to-One

Universal Containers wants to ensure that a) each position has, at most, one related salary record; and b) records that violate this rule should be prevented from being saved.

For this scenario, would you use a declarative or programmatic solution? What are the steps of the high-level solution?

### Scenario C: Updating the Job Application Status When Creating a Job Offer

Universal Containers wants to notate on the job application once a job has been offered to a candidate. If a new offer record is created, then the job application should be set to **Stage** = `Offer Extended`and **Status** = `Hold`.

For this scenario, would you use a declarative or programmatic solution? What are the steps of the high-level solution?