APEX - TRIGGER DESIGN PATTERNS

Design patterns are used to make our code more efficient and to avoid hitting the governor limits. Often developers can write inefficient code that can cause repeated instantiation of objects. This can result in inefficient, poorly performing code, and potentially the breaching of governor limits. This most commonly occurs in triggers, as they can operate against a set of records.

We will be looking at some important design pattern strategies in this chapter.

Bulk Triggers Design Patterns

In real business case, it will be possible that you may need to process thousands of records in one go. If your trigger is not designed to handle such situations, then it may fail while processing the records. There are some best practices which you need to follow while implementing the triggers. All triggers are bulk triggers by default, and can process multiple records at a time. You should always plan to process more than one record at a time.

Consider a business case, where you would like to process large number of records and you have written the below trigger. This is the same example which we had taken for inserting the invoice record when Customer Status changes from Inactive to Active.

```
//Bad Trigger Example
trigger Customer_After_Insert on APEX_Customer__c (after update) {
 for (APEX_Customer__c objCustomer: Trigger.new) {
  if (objCustomer.APEX_Customer_Status__c == 'Active' &&
trigger.oldMap.get(objCustomer.id).APEX_Customer_Status__c == 'Inactive') {//condition to
check the old value and new value
   APEX_Invoice__c objInvoice = new APEX_Invoice__c();
   objInvoice.APEX_Status__c = 'Pending';
   insert objInvoice;//DML to insert the Invoice List in SFDC
  }
 }
}
```

If you have a closer look, then you could see that the DML Statement has been written in for loop block which will work when processing only few records but when you are processing some hundreds of records, it will reach the DML Statement limit per transaction which is governor limit. We will have a detailed look on Governor Limits in later chapter.

To avoid this, we have to make the trigger efficient for processing multiple records at a time.

Here is the best practice example for the same:

```
//Modified Trigger Code-Bulk Trigger
trigger Customer_After_Insert on APEX_Customer__c (after update) {
 List<apex_invoice__c> InvoiceList = new List<apex_invoice__c>();
 for (APEX_Customer__c objCustomer: Trigger.new) {
  if (objCustomer.APEX_Customer_Status__c == 'Active' &&
trigger.oldMap.get(objCustomer.id).APEX_Customer_Status__c == 'Inactive') {//condition to
check the old value and new value
   APEX_Invoice__c objInvoice = new APEX_Invoice__c();
   objInvoice.APEX_Status__c = 'Pending';
   InvoiceList.add(objInvoice);//Adding records to List
  }
 }
 insert InvoiceList;//DML to insert the Invoice List in SFDC, this list contains the all
records which need to be modified and will fire only one DML
}
```

This trigger will only fire 1 DML statement as it will operating over a List and List has all the records which needs to be modified.

By this way, you could avoid the DML statement governor limits.

Trigger Helper Class

Writing the whole code in trigger is also not a good practice. Hence you should call the Apex class and delegate the processing from Trigger to Apex class as shown below. Trigger Helper class is the class which does all the processing for trigger.

Let's take our invoice record creation example again.

```
//Below is the Trigger without Helper class
trigger Customer_After_Insert on APEX_Customer__c (after update) {
 List<apex_invoice__c> InvoiceList = new List<apex_invoice__c>();
 for (APEX_Customer__c objCustomer: Trigger.new) {
  if (objCustomer.APEX_Customer_Status__c == 'Active' &&
trigger.oldMap.get(objCustomer.id).APEX_Customer_Status__c == 'Inactive') {//condition to
check the old value and new value
   APEX_Invoice__c objInvoice = new APEX_Invoice__c();
   objInvoice.APEX_Status__c = 'Pending';
   InvoiceList.add(objInvoice);
  }
 }
 insert InvoiceList;//DML to insert the Invoice List in SFDC
}

//Below is the trigger with helper class
//Trigger with Helper Class
trigger Customer_After_Insert on APEX_Customer__c (after update) {
    CustomerTriggerHelper.createInvoiceRecords(Trigger.new, trigger.oldMap);//Trigger
calls the helper class and does not have any code in Trigger
}
```

Helper Class:

```
public class CustomerTriggerHelper {
    public static void createInvoiceRecords (List<apex_customer__c> customerList,
Map<id, apex_customer__c> oldMapCustomer) {
        List<apex_invoice__c> InvoiceList = new List<apex_invoice__c>();
        for (APEX_Customer__c objCustomer: customerList) {
            if (objCustomer.APEX_Customer_Status__c == 'Active' &&
oldMapCustomer.get(objCustomer.id).APEX_Customer_Status__c == 'Inactive') {//condition to
check the old value and new value
                APEX_Invoice__c objInvoice = new APEX_Invoice__c();
                //objInvoice.APEX_Status__c = 'Pending';
                InvoiceList.add(objInvoice);
            }
        }
        insert InvoiceList;//DML to insert the Invoice List in SFDC
    }
}
```

In this, all the processing has been delegated to the helper class and when we need a new functionality we can simply add the code to helper class without modifying the trigger.

Single Trigger on Each sObject

Always create a single trigger on each object. Multiple triggers on the same object could cause the conflict and errors if it reaches the governor limits.

You could use the context variable to call the different methods from helper class as per the requirement. Consider our previous example. Suppose that our createInvoice method should be called only when the record is updated and on multiple events. Then we could control the execution as below:

```
//Trigger with Context variable for controlling the calling flow
trigger Customer_After_Insert on APEX_Customer__c (after update, after insert) {
 if (trigger.isAfter && trigger.isUpdate) {//This condition will check for trigger
events using isAfter and isUpdate context variable
     CustomerTriggerHelper.createInvoiceRecords(Trigger.new);//Trigger calls the helper
class and does not have any code in Trigger and this will be called only when trigger
ids after update
 }
}

//Helper Class
public class CustomerTriggerHelper {
 //Method To Create Invoice Records
 public static void createInvoiceRecords (List<apex_customer__c> customerList) {
  for (APEX_Customer__c objCustomer: customerList) {
   if (objCustomer.APEX_Customer_Status__c == 'Active' &&
trigger.oldMap.get(objCustomer.id).APEX_Customer_Status__c == 'Inactive') {//condition to
check the old value and new value
    APEX_Invoice__c objInvoice = new APEX_Invoice__c();
    objInvoice.APEX_Status__c = 'Pending';
    InvoiceList.add(objInvoice);
   }
  }
  insert InvoiceList;//DML to insert the Invoice List in SFDC
 }
}
```