APEX - GOVERNER LIMITS

Governor execution limits ensure the efficient use of resources on the Force.com multitenant platform. It is the limit specified by the Salesforce.com on code execution for efficient processing.

What are Governor Limits?

As we know, Apex runs in multi-tenant environment; i.e. a single resource is shared by all the customers and organizations. So, it is necessary to make sure that no one monopolizes the resources and hence Salesforce.com has created the set of limits which governs and limits the code execution. Whenever any of the governor limits are crossed, it will throw error and will halt the execution of program.

From a Developer's perspective, it is of utmost important to ensure that our code should be scalable and should not hit the limits.

All these limits are applied on per transaction basis. A single trigger execution is one transaction.

As we have seen, the trigger design pattern is the one way to avoid the limit error. Let's have look at some important limits.

Avoiding SOQL Query Limit

You could issue only 100 queries per transaction, that is when your code will issue more than 100 SOQL queries then it will throw error.

Example:

Below is the example which shows how SOQL query limit could be reached:

The following trigger iterates over a list of customers and updates the child record's *Invoice* description with string 'Ok to Pay'.

```
//Heper class:Below code needs o be checked.
public class CustomerTriggerHelper {

 public static void isAfterUpdateCall(Trigger.new) {
  createInvoiceRecords(trigger.new);//Method call
  updateCustomerDescription(trigger.new);
 }

 //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
 }

 //Method to update the invoice records
 public static updateCustomerDescription (List<apex_customer__c> customerList) {
  for (APEX_Customer__c objCust: customerList) {
   List<apex_invoice__c> invList = [SELECT Id, Name, APEX_Description__c FROM
APEX_Invoice__c WHERE APEX_Customer__c = :objCust.id];//This query will fire for the
number of records customer list has and will hit the governor limit when records are
more than 100
   for (APEX_Invoice__c objInv: invList) {
    objInv.APEX_Description__c = 'OK To Pay';
```

```
    update objInv;//Update invoice, this will also hit the governor limit for DML if
large number(150) of records are there
   }
  }
 }
}
```

When method 'updateCustomerDescription' is called and number of customer records are more than 100 then it will hit the SOQL limit.

To avoid this, never write the SOQL query in For Loop. In this case, SOQL query has been written in for loop.

Below is the example with which we could avoid the DML as well as SOQL limit. We have used the nested relationship query to fetch the invoice records and used the context variable trigger.newMap to get the map of id and Customer records.

```
//SOQL-Good Way to Write Query and avoid limit exception
//Helper Class
public class CustomerTriggerHelper {

 public static void isAfterUpdateCall(Trigger.new) {
  createInvoiceRecords(trigger.new);//Method call
  updateCustomerDescription(trigger.new, trigger.newMap);
 }

 //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
 }

 //Method to update the invoice records
 public static updateCustomerDescription (List<apex_customer__c> customerList, Map<id,
apex_customer__c> newMapVariable) {
  List<apex_customer__c> customerListWithInvoice = [SELECT id, Name,(SELECT Id, Name,
APEX_Description__c FROM APEX_Invoice__r) FROM APEX_Customer__c WHERE Id IN
:newMapVariable.keySet()];//Query will be for only one time and fetches all the records
  List<apex_invoice__c> invoiceToUpdate = new List<apex_invoice__c>();
  for (APEX_Customer__c objCust: customerList) {
   for (APEX_Invoice__c objInv: invList) {
    objInv.APEX_Description__c = 'OK To Pay';
    invoiceToUpdate.add(objInv);//Add the modified records to List
   }
  }
  update invoiceToUpdate;
 }
}
```

DML Bulk Calls

As we have seen in many examples, we have been adding the modified records in a List and then we are performing the DML operation on that List rather than performing the DML for single records. Below is the example for the same. This is the example of Bulk trigger along with the trigger helper class pattern. You must save the helper class first and then save the trigger.

Note: Paste the below code in 'CustomerTriggerHelper' class which we have created earlier.

```
//Helper Class
```

```
public class CustomerTriggerHelper {

    public static void isAfterUpdateCall(List<apex_customer__c> customerList, Map<id,
apex_customer__c> mapIdToCustomers, Map<id, apex_customer__c> mapOldItToCustomers) {
        createInvoiceRecords(customerList, mapOldItToCustomers);//Method call
        updateCustomerDescription(customerList,mapIdToCustomers, mapOldItToCustomers);
    }

    //Method To Create Invoice Records
    public static void createInvoiceRecords (List<apex_customer__c> customerList,
Map<id, apex_customer__c> mapOldItToCustomers) {
        List<apex_invoice__c> InvoiceList = new List<apex_invoice__c>();
        List<apex_customer__c> customerToInvoice = [SELECT id, Name FROM
APEX_Customer__c LIMIT 1];
        for (APEX_Customer__c objCustomer: customerList) {
            if (objCustomer.APEX_Customer_Status__c == 'Active' &&
mapOldItToCustomers.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';
                objInvoice.APEX_Customer__c = objCustomer.id;
                InvoiceList.add(objInvoice);
            }
        }
        system.debug('InvoiceList&&&'+InvoiceList);
        insert InvoiceList;//DML to insert the Invoice List in SFDC. This also follows
the Bulk pattern
    }

    //Method to update the invoice records
    public static void updateCustomerDescription (List<apex_customer__c> customerList,
Map<id, apex_customer__c> newMapVariable, Map<id, apex_customer__c> oldCustomerMap) {
        List<apex_customer__c> customerListWithInvoice = [SELECT id, Name,(SELECT Id,
Name, APEX_Description__c FROM Invoices__r) FROM APEX_Customer__c WHERE Id IN
:newMapVariable.keySet()];//Query will be for only one time and fetches all the records
        List<apex_invoice__c> invoiceToUpdate = new List<apex_invoice__c>();
  List<apex_invoice__c> invoiceFetched = new List<apex_invoice__c>();
  invoiceFetched = customerListWithInvoice[0].Invoices__r;
  system.debug('invoiceFetched'+invoiceFetched);
        system.debug('customerListWithInvoice****'+customerListWithInvoice);
  for (APEX_Customer__c objCust: customerList) {
   system.debug('objCust.Invoices__r'+objCust.Invoices__r);
            if (objCust.APEX_Active__c == true &&
oldCustomerMap.get(objCust.id).APEX_Active__c == false) {
    for (APEX_Invoice__c objInv: invoiceFetched) {
     system.debug('I am in For Loop'+objInv);
     objInv.APEX_Description__c = 'OK To Pay';
     invoiceToUpdate.add(objInv);//Add the modified records to List
    }
   }
        }
  system.debug('Value of List ***'+invoiceToUpdate);
        update invoiceToUpdate;//This statement is Bulk DML which performs the DML on
List and avoids the DML Governor limit
    }
}

//Trigger Code for this class: Paste this code in 'Customer_After_Insert' trigger on
Customer Object
trigger Customer_After_Insert on APEX_Customer__c (after update) {
    CustomerTriggerHelper.isAfterUpdateCall(Trigger.new, trigger.newMap,
trigger.oldMap);//Trigger calls the helper class and does not have any code in Trigger
}
```

Other Salesforce Governor Limits

Below are some important governor limits which we need to remember. You could check other governor limits as well using Salesdorce.com Apex Developer guide.

Description	Limit
Total heap size	6 MB/12 MB
Total number of DML statements issued	150
Total number of records retrieved by a single SOSL query	2000
Total number of SOSL queries issued	20
Total number of records retrieved by Database.getQueryLocator	10000
Total number of records retrieved by SOQL queries	50000