Why should you bulkify your code?

First of all, let us start by checking what “bulkify your code” means. To put it into words, the term refers to the concept of making sure your code properly handles more than one record at a time. When working in Salesforce, it is always suggested that you do bulkify your code, so that it runs properly, without any possible error. Bulkifying your code becomes especially important when you import or insert more than one record so that your code properly handles all the records in different contexts. That is why we do advise you to bulkify all your Apex classes so as to improve your org performance, and not just the triggers, as suggested by Salesforce.

Apex code has Governor Limits (we will come to that later) and, for example, a Trigger can only process up to 200 records at once. Yet, it is rather common that with Data Loader you import much more than that. Thus, you have to be careful with the way that you build your Trigger so that you don’t face, for instance, the “Too many SOQL queries: 101” error.

What are Governor Limits?

Governor Limits are a set of rules that Salesforce has to improve performance in your system (by writing an efficient and scalable code) and to prevent your code from using all the shared resources. However, governor limits also present limitations, as you cannot improve your usable resources by upgrading your license or by any other way. Besides, you must be careful because Governor Limits are shared by all the triggers you have for one object, being either one or five triggers per object.

So, how can you overcome the Governor Limits drawbacks?

Overcoming Governor Limits with Maps

Maps are the solution to beat Governor Limits. So, what is a Map?

Basically, a Map is a searchable table composed of two fields. First, a unique value that you use to search, such as a record Id or a unique number or string. In the second one, you’ll find the information that you want like the record itself.

```
//instantiate the map
Map<Integer,String> fruitMap = new Map<Integer,String>();
//add fruit to the map with identifier number
//fruiMap.put(key,value);
fruitMap.put(1, 'apple');
fruitMap.put(2, 'banana');
fruitMap.put(3, 'pineapple');
//get the fruit that i want using the get() function
//with the key of the frui that i want
String fruitThatIWant = fruitMap.get(1);
/use System.assertEquals to confirm that i get the fruit that i want
System.assertEquals('apple', FruitThatIWant);
```

So, having this fruit Map is the same as having the following table and search for the first column.

1	apple
2	banana
3	pineapple

Now that you know what maps are, let’s see which are the most common errors and how to fix them!

Errors faced with not bulkified code and how to correct them

The next showing errors are faced when imported 200 records via Data Loader or Data Import Wizard.

Too many SOQL queries: 101

The most common error that it is faced with not bulkified code is “***Too many SOQL queries: 101***”.

To have this error is as simple as have a SOQL query inside a *for* loop, for example:

```
trigger ContactTrigger on Contact (before insert) {
    for (Contact newContact : Trigger.new) {
        //check if Contact Phone is null and fill it with related Account Phone
        if (newContact.Phone == null) {
            Account acc = [SELECT Phone FROM Account WHERE Id =: newContact.AccountId];
            newContact.Phone = acc.Phone;
        }//if
    }//for
}//trigger
```

In this case, if each imported contact has no phone number, the Trigger will try to do 200 SOQL queries to get the Phone field from the respective account and, as it reaches the 100 SOQL queries limit, you will get the mentioned error. So, you should never use a SOQL query inside a loop.

To improve the above code so that you won’t face the SOQL error you should use the Map class we mentioned earlier to save all the Account information and search for it inside the Trigger.new *for* cycle.

```
trigger ContactTrigger on Contact (before insert) {
    //create a set of Id to query Account object
    Set<Id> accountIdSet = new Set<Id>();
    for (Contact newContact : Trigger.new) {
        accountIdSet.add(newContact.AccountId);
    }//for

    //this map will have the Account Id has key and the account record has value
    Map<Id, Account> accountMap = new Map<Id, Account>([SELECT Id, Phone FROM Account WHERE Id IN: accountIdSet]);

    // auxiliary Account variable to use inside the loop
    Account auxAccount = new Account();
    for (Contact newContact : Trigger.new) {
        //the map comparation is to check if there are a value in the map with que newContact.AccountId key
        if (newContact.Phone == null && accountMap.get(newContact.AccountId) != null) {
            auxAccount = accountMap.get(newContact.AccountId);
            newContact.Phone = auxAccount.Phone;
        }//if
    }//for
}//trigger
```

This way you will not reach the SOQL limit because you only use one query in the whole trigger.

Too many DML statements: 151

Another error you sometimes may face is "***Too many DML statements: 151***". In this error, the total number of DML issued is 150 and it could appear in the following case:

```
trigger AccountTrigger on Account (after insert) {
    for (Account newAccount : Trigger.new) {
        //create a new Contact for each new Account
        Contact contact = new Contact();
        contact.LastName = newAccount.Name;
        contact.AccountId = newAccount.Id;
        insert contact;
    }//for
}//trigger
```

In this case, the error you face happens because of the *insert* inside the *for* cycle. As the Trigger.new list has 200 records, and it reach the limit, so, the solution is to create a list, add the new contacts to that list and then insert it.

```
trigger AccountTrigger on Account (after insert) {
    //create a list where you will add all the contacts to insert
    List<Contact> contactsToInsertList = new List<Contact>();

    Contact contact = new Contact();
    for (Account newAccount : Trigger.new) {
        contact = new Contact();
        contact.LastName = newAccount.Name;
        contact.AccountId = newAccount.Id;
        //add the new contact to the list
        contactsToInsertList.add(contact);
    }//for

    //insert the list
    insert contactsToInsertList;
}//trigger
```

By creating a list of contacts to insert, only one insert is needed.

Apex CPU time limit exceeded

You can also face the "***Apex CPU time limit exceeded***" error. This error means what is explained in the message: you reach the CPU usage limit that your org can use, as is explained in the governor limits above.

```
trigger AccountTrigger on Account (after insert) {
    //get all contacts
    List<Contact> contactsList = [SELECT Id, Phone FROM Contact];
    for (Account newAccount : Trigger.new) {
        for(Contact cnt: contactsList){
            //for each insert Account, run through all existing Contacts, check
            //if there is a Contact whith the same Phone as the Account and add
            //PRIMARY CONTACT - to the Contact's Title field
            if(newAccount.Id == cnt.AccountId && newAccount.Phone == cnt.Phone){
                cnt.Title = 'PRIMARY CONTACT - ' + cnt.Title;
            }//if
        }//for
    }//for
    //update Contacts
    update contactsList;
}//trigger
```

This error occurs in case you have a lot of contacts because in every new account inserted, the inner *for* cycle will run across all contacts in the list.

```
trigger AccountTrigger on Account (after update) {
    //create a set of Id to query Account object
    Set<Id> accountIdSet = new Set<Id>();
    for (Account newAccount : Trigger.new) {
        accountIdSet.add(newAccount.Id);
    }//for

    //create a list with contacts that have the Account Id of the inserted Accounts
    List<Contact> contactsList = [SELECT Id, Phone, Title FROM Contact WHERE AccountId IN: accountIdSet];

    //create a map that has the Contact Phone as a key and Contact as value
    Map<String,Contact> contactPhoneIdMap = new Map<String, Contact>();
    for(Contact cnt: contactsList){
        contactPhoneIdMap.put(cnt.Phone, cnt);
    }//for

    contactsList = new List<Contact>();
    Contact contact = new Contact();
    for (Account newAccount : Trigger.new) {
        //check if in the contactPhoneIdMap, the Phone field is equals to the newAccount Phone
        if (newAccount.Phone == contactPhoneIdMap.get(newAccount.Phone).Phone) {
            contact = contactPhoneIdMap.get(newAccount.Phone);
            contact.Title = 'PRIMARY CONTACT - ' + contact.Title;
            //add contact to Contacts to update
            contactsList.add(contact);
        }//if
    }//for
    //update Contacts
    update contactsList;
}//trigger
```

To upgrade your code, you should create a Set of the inserted Account IDs and use it to query only the contacts that belong to those Accounts. Hereafter, you can create a Contact map in a way to search by Phone inside the Trigger.new *for* cycle.