Managing the heap using SOQL For Loops (with a little code seperation)

Posted on August 25, 2015 by Tony Scott

I'm a big fan of code separation and code reuse. When I'm coding I like to keep my SOQL out of the business logic and in a gateway class of its own. That way I can reuse common queries in my code and keep a seperate database layer of sorts. If I need to modify my queries due to new custom fields I know straight where to go.

When working with large data sets it's important to manage the heap which is one of the benefits you get from using a ***SOQL For Loop***. However that presents a challenge in separating business logic and database access since the business logic must be contained within the loop.

Consider a basic query that loads opportunities for a given account. I would put this in a gateway class like this:

```
1  public with sharing class OpportunityGateway
2  {
3      public static List<Opportunity> getOpportunitiesByAccountId(Id accoun
4      {
5          return [Select Id, Name, StageName
6                  From Opportunity
7                  Where AccountId = :accountId];
8      }
9  }
```

I might have a service class that sets all Opportunities to Closed Won for a given account:

```
1  public with sharing class OpportunityService
2  {
3      public static Integer closeOpportunities(Id accountId)
4      {
5          // Load the Opportunities for the Account
6          List<Opportunity> opps = OpportunityGateway.getOpportunitiesByAc
7
```

```
        // Iterate the Opportunities and set the Stage to Closed Won
        for(Opportunity opp : opps)
        {
            opp.StageName = 'Closed Won';
        }

        // Update the Opportunities
        update opps;

        System.debug('Heap Size: ' + Limits.getHeapSize());

        return opps.size();
    }
}
```

So the problem with this is that if I have a lot of Opportunities associated with my Account I'm going to consume a lot of heap. This is because Salesforce will load the entire result set into the List.

To demonstrate, I have the following test class that creates 400 opportunities and then uses the above service to update them:

```
@isTest
private class OpportunitySericeTest
{
    @isTest
    static void testCloseOpportunities()
    {
        Account acc = [Select Id From Account Where Name = 'Test Account

        Integer updateCount = OpportunityService.closeOpportunities1(acc

        System.assertEquals(400, updateCount);
    }

    @testSetup
    static void createTestData()
    {
        Account acc = new Account(Name = 'Test Account');
        insert acc;

        List<Opportunity> opps = new List<Opportunity>();

        for(Integer i=0; i<400; i++)
        {
            opps.add(
                new Opportunity(
```

```
                    AccountId = acc.Id,
                    Name = 'Test Opportunity ' + i+1,
                    StageName = 'Prospecting',
                    CloseDate = System.today()
                )
            );
        }

        insert opps;
    }
}
```

Running the test and looking at the debug log I can see how much heap this has consumed:

DEBUG|Heap Size: 36273

The solution to this is to use a ***SOQL For Loop*** to chunk the large data set and therefore reduce the heap by working on a subset of records at a time.

But wait, I still want to retain my code seperation and keep my SOQL in my gateway class.

To do this I introduce a simple interface:

```
public interface IQueryHandler
{
    void execute(List<SObject> scope);
}
```

I then modify my gateway class to make use of a SOQL For Loop and delegate the contents to an implementation of my interface rather than return the result set:

```
public with sharing class OpportunityGateway
{
    public static void getOpportunitiesByAccountId(Id accountId, IQueryH
    {
        for (List<Opportunity> opps : [Select Id, Name, StageName
                                       From Opportunity
                                       Where AccountId = :accountId])
        {
            handler.execute(opps);
        }
    }
}
```

My service class looks a little different too as I need to pass an implementation of the ***IQueryHandler*** interface to the gateway method. To do this I use a private inner class within the service class:

```
public with sharing class OpportunityService
{
    public static Integer closeOpportunities(Id accountId)
    {
        // Create the Query Handler
        CloseOpportunitiesQueryHandler queryHandler = new CloseOpportuni

        // Process the Opportunities for the Account using the Query Har
        OpportunityGateway.getOpportunitiesByAccountId(accountId, queryH

        System.debug('Heap Size: ' + Limits.getHeapSize());
        System.debug('Max Heap Size: ' + queryHandler.getMaxHeap());

        return queryHandler.getRowCount();
    }

    private class CloseOpportunitiesQueryHandler
        implements IQueryHandler
    {
        private Integer m_rowCount;
        private Integer m_maxHeap;

        // Class Constructor
        public CloseOpportunitiesQueryHandler()
        {
            m_rowCount = 0;
            m_maxHeap = 0;
        }

        // Interface method takes a List of SOBjects
        public void execute(List<SObject> scope)
        {
            // Iterate the Opportunities and set the Stage to Closed Won
            for(Opportunity opp : (List<Opportunity>)scope)
            {
                opp.StageName = 'Closed Won';
            }

            // Update the Opportunities
            update scope;

            m_rowCount += scope.size();

            Integer heapSize = Limits.getHeapSize();
```

```
45              m_maxHeap = heapSize > m_maxHeap ? heapSize : m_maxHeap;
46          }
47
48          public Integer getRowCount()
49          {
50              return m_rowCount;
51          }
52
53          public Integer getMaxHeap()
54          {
55              return m_maxHeap;
56          }
57      }
58  }
```

Just to make sure this acheives the benefit I'm looking for I can run the unit test again and check the results:

DEBUG|Heap Size: 1322
DEBUG|Max Heap Size: 18881

As you can see the the heap consumed at the end of the operation is significantly less using the ***SOQL For Loop***. Even during the processing of the query it peaks around half of the original implentation. Considering the ***SOQL For Loop*** chunks the records into batches of 200 and we are processing 400 records this is what we would expect.

A couple of things to take note of:

- Be aware of governor limits, in particular the number of query rows and DML operations. This pattern does a single update within a ***SOQL For Loop***. The query will chunk at 200 records per iteration. This would, in theory, allow 30000 records to be updated using the maximum of 150 DML statements. However, the maximum number of DML rows that can be processed is 10000 so in reality if you have more than 10000 records, you'll be in trouble way before DML statments.
- If you are going to be processing a large number of records you may wish to look at using batch apex instead.