**Batch Processing**

In the current DSpace design, the database transactions are in most of the cases relatively long: from Context creation to the moment the Context is completed. Especially when doing batch processing, that transaction can become very long. The new data access layer introduced in DSpace 6 which is based on Hibernate has built-in cache and auto-update mechanisms. But these mechanisms do not work well with long transactions and even have an exponentially adverse-effect on performance.

Therefore we added a new method enableBatchMode() to the DSpace Context class which tells our database connection that we are going to do some batch processing. The database connection (Hibernate in our case) can then optimize itself to deal with a large number of inserts, updates and deletes. Hibernate will then not postpone update statements anymore which is better in the case of batch processing. The method isBatchModeEnabled() lets you check if the current Context is in “batch mode”.

When dealing with a lot of records, it is also important to deal with the size of the (Hibernate) cache. A large cache can also lead to decreased performance and eventually to “out of memory” exceptions. To help developers to better manage the cache, a method getCacheSize() was added to the DSpace Context class that will give you the number of database records currently cached by the database connection. Another new method uncacheEntity(ReloadableEntity entity) will allow you to clear the cache (of a single object) and free up (heap) memory. The uncacheEntity() method may be used to immediately remove an object from heap memory once the batch processing is finished with it. Besides the uncacheEntity() method, the commit() method in the DSpace Context class will also clear the cache, flush all pending changes to the database and commit the current database transaction. The database changes will then be visible to other threads.

BUT uncacheEntity() and commit() come at a price. After calling this method all previously fetched entities (hibernate terminology for database record) are “detached” (pending changes are not tracked anymore) and cannot be combined with “attached” entities. If you change a value in a detached entity, Hibernate will not automatically push that change to the database. If you still want to change a value of a detached entity or if you want to use that entity in combination with attached entities (e.g. adding a bitstream to an item) after you have cleared the cache, you first have to reload that entity. Reloading means asking the database connection to re-add the entity from the database to the cache and get a new object reference to the required entity. From then on, it is important that you use that new object reference. To simplify the process of reloading detached entities, we've added a reloadEntity(ReloadableEntity entity) method to the DSpace Context class with a new interface ReloadableEntity. This method will give the user a new “attached” reference to the requested entity. All DSpace Objects and some extra classes implement the ReloadableEntity interface so that they can be easily reloaded.

Examples on how to use these new methods can be found in the IndexClient class. But to summarize, when batch processing it is important that:

1. You put the Context into batch processing mode using the method:

   ```java
   boolean originalMode = context.isBatchModeEnabled();
   context.enableBatchMode(true);
   ```

2. Perform necessary batch operations, being careful to call uncacheEntity() whenever you complete operations on each object. Alternatively, you can commit() the context once the object cache reaches a particular size (see getCacheSize()). Remember, once an object is "uncached", you will have to reload it (see reloadEntity()) before you can work with it again:

   ```java
   final Iterator<Item> itemIterator = itemService.findByCollection(context, collection);
   // Loop through all items
   while (itemIterator.hasNext()) {
       // Get access to next Item
       Item item = itemIterator.next();
       // do something with Item ...
       // To prevent memory issues, discard Item from the cache after processing
       context.uncacheEntity(item);
   }
   // Remember: calling commit() will decache all objects
   context.commit();
   // So, if you need to reuse your Collection *post* commit(), you'd have to reload it
   Collection collection = context.reloadEntity(collection);
   ```

3. When you're finished with processing the records, you put the context back into its original mode:

   ```java
   context.enableBatchMode(originalMode);
   ```