Iterating Large Datasets

When processing large datasets, it is important to consider performance. In particular,

- The process must complete within a typical web transaction. If the process takes more than 30 seconds or so, it should be implemented as a Job Plugin and be processed by the Registry Job Shell.
- Large datasets should not be fully loaded into memory at once (eg: by using a `find('all')`), as a given server may not be able to handle the full dataset, or may suffer performance degradation. Use the `PaginatedSqlIterator` instead.

PaginatedSqlIterator

Available from Registry v3.3.0, the PaginatedSqlIterator is a COmanage class implementing the PHP Iterator interface. It paginates over a Cake model, internally loading a subset of the full results in order to optimize memory usage. It uses Keyset Pagination in order to guarantee a complete iteration over multiple queries. Since Keyset Pagination is based on the `id` field, if new records are added during pagination, they will become available when the iteration approaches the end of the results.

To use the PaginatedSqlIterator, instantiate it with the Model to query, along with *conditions*, *fields*, and *contains* parameters as appropriate. Then simply use the iterator in a `foreach()` loop.

The iterator can also obtain a record count for progress calculation purposes. For performance reasons, the record count is only updated when explicitly asked.

Registry v3/v4

```
App::uses("PaginatedSqlIterator", "Lib");

class myClass {
  public function doSomething() {
    $iterator = new PaginatedSqlIterator(
      // Model to query
      'CoPerson',
      // Condition
      array('CoPerson.co_id' => $coId),
      // Fields, or null to retrieve all fields
      array('CoPerson.id', 'CoPerson.status'),
      // Contains for associated models, or null
      array('EmailAddress')
    );

    // Initial record count
    $total = $iterator->count();
    $i = 0;

    foreach($iterator as $k => $v) {
      // Do something for this particular record

      ...

      // Every 1000 records, refresh the count just in case new records have been added.
      // Note this is for informational purposes for the calling code *only*, and is not
      // required. The iterator will correctly retrieve subsequently added records
      // regardless of whether or not count() is called.
      if($i % 1000 == 0) {
        $total = $iterator->count(true);
      }
    }
  }
}
```

Registry v5+

```
use App\Lib\Util\PaginatedSqlIterator;

class myClass {
  public function doSomething() {
    $iterator = new PaginatedSqlIterator(
      // Model to query
      $this->People->getTarget(),
      // Conditions
      ['co_id' => $coId]
    );

    // Initial record count
    $total = $iterator->count();
    $i = 0;

    foreach($iterator as $k => $v) {
      // Do something for this particular record

      ...

      // Every 1000 records, refresh the count just in case new records have been added.
      // Note this is for informational purposes for the calling code *only*, and is not
      // required. The iterator will correctly retrieve subsequently added records
      // regardless of whether or not count() is called.
      if($i % 1000 == 0) {
        $total = $iterator->count(true);
      }
    }
  }
}
```