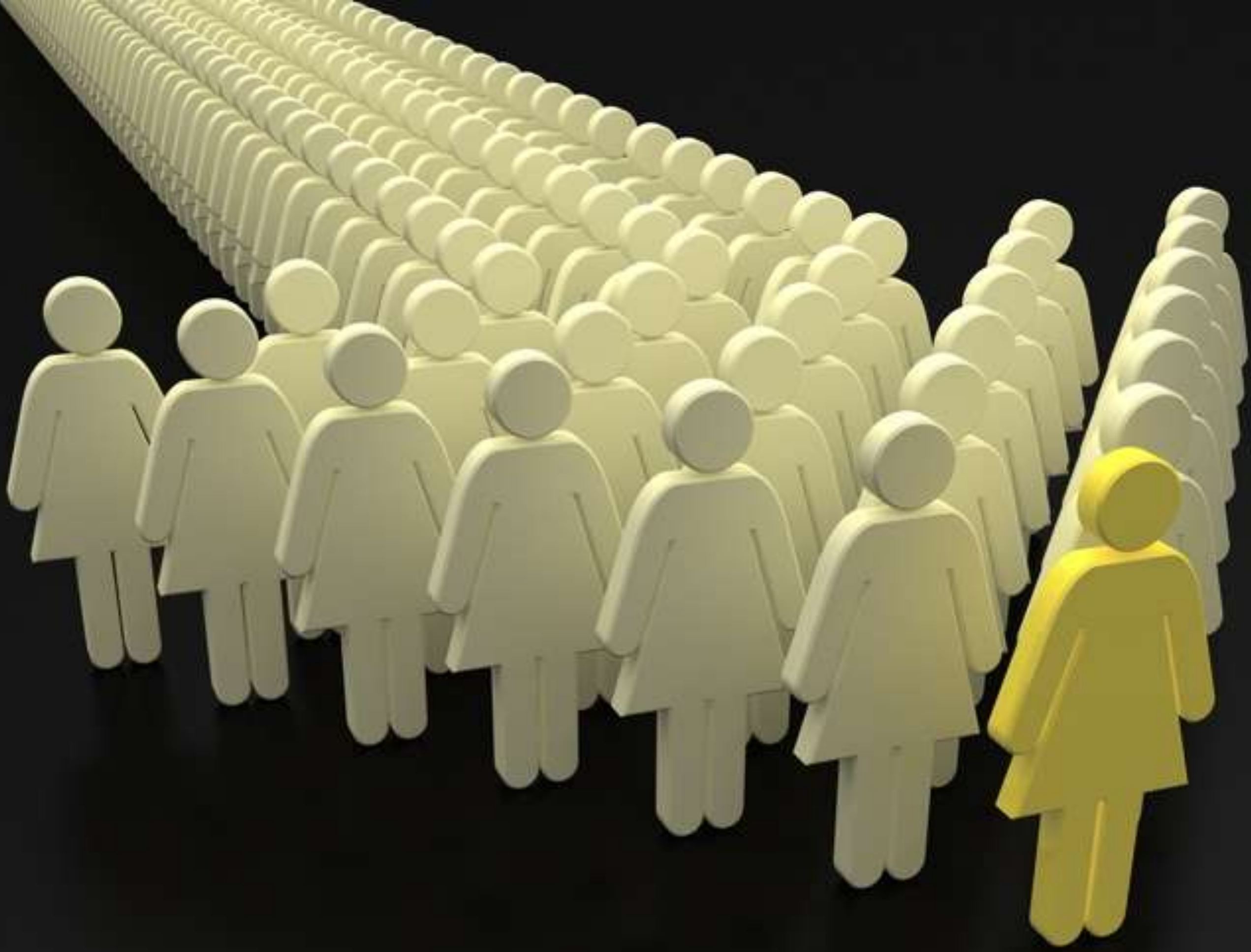


To use an ORM or not to use an ORM?

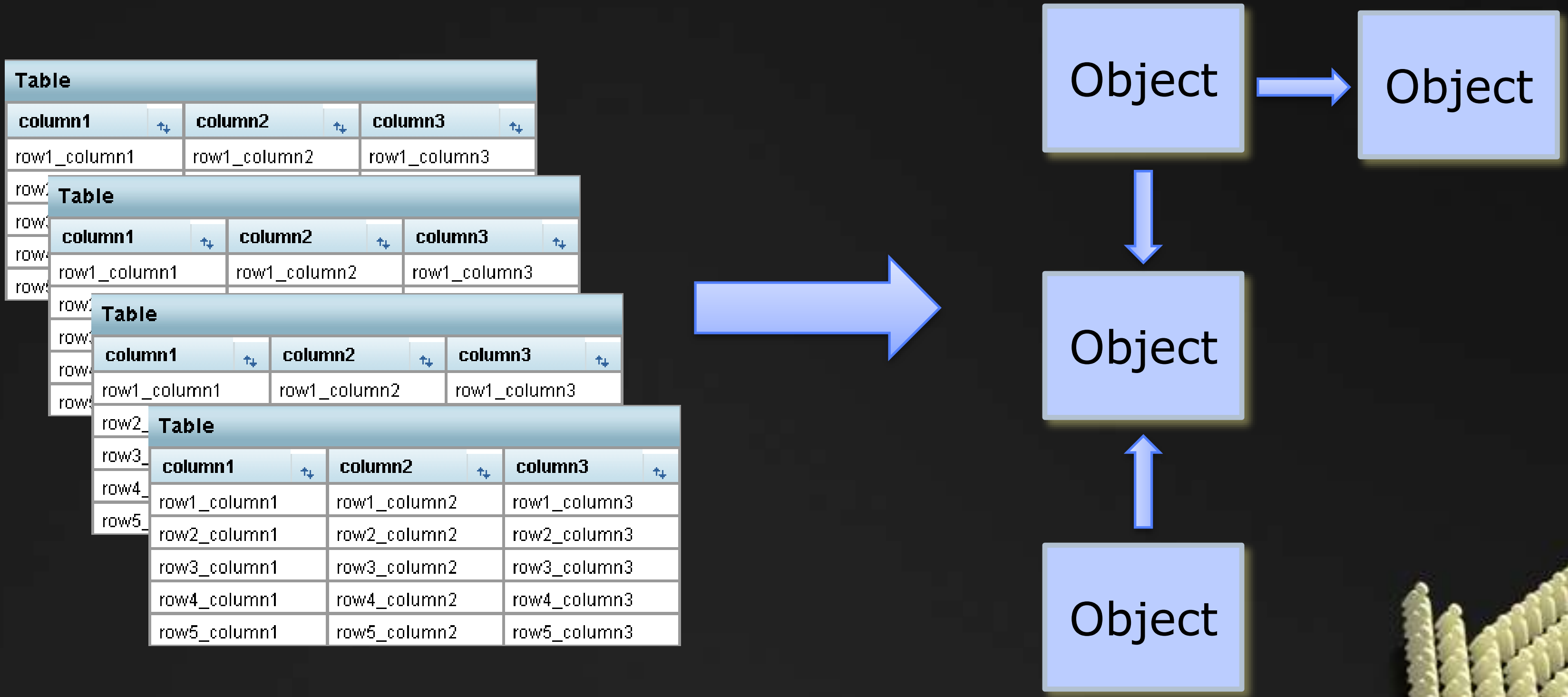
Harro Verton @ CICONF 2012, London, UK

Who is this guy?

- Digitally known as WanWizard
- Partner in Exite, an ICT company in the Netherlands
- Runs Exite’s Development and Hosting business
- In software development for over 30 years
- Loves writing code whenever time permits
  - DataMapper ORM
  - Modular CI
  - FuelPHP Core developer
- Hates being reminded of his age!

So, what is an ORM?

- Object Relational Mapping
- Programming technique to map scalar data into objects with full knowledge of their relationship with other objects

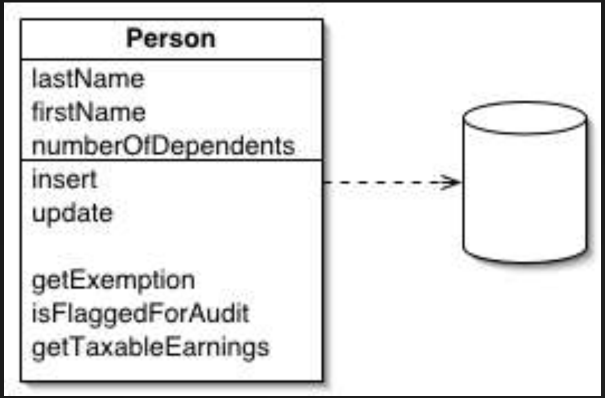


column1	column2	column3
row1_column1	row1_column2	row1_column3
row2_column1	row2_column2	row2_column3
row3_column1	row3_column2	row3_column3
row4_column1	row4_column2	row4_column3
row5_column1	row5_column2	row5_column3

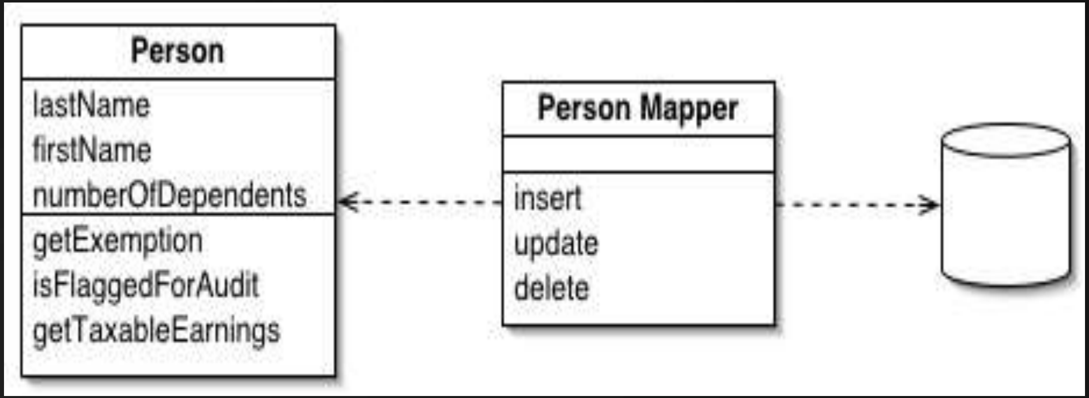
📖 *Martin Fowler, Patterns of Enterprise Application Architecture*

Architectural design patterns

Active Record pattern



Data Mapper pattern



- Other patterns exist, but are not widely used
  - Table Data Gateway
    - basic singleton model per table
  - Row Data Gateway
    - separates table and row singletons

Active Record pattern

- A design pattern where an object is represented as a record on a table in a relational database
- Very efficient due to the assumption "1 object = 1 record"
- Relationship from object ➡ database
- Configuration driven, not schema dependent
- Code and data in a single object
- Very fast if implemented properly
- Used by most ORM implementations

- CodeIgniter's Active Record is <u>**NOT**</u> an Active Record pattern, it's a Query Builder!

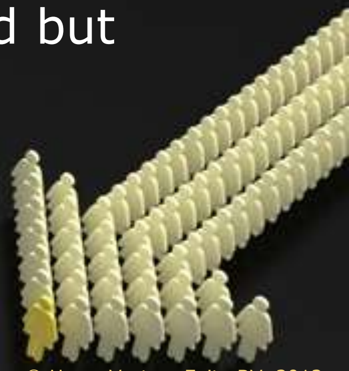
Data Mapper pattern

- A design pattern where an object represents a data collection
- More flexible, a collection can be anything
- Mapping code is separated from data
- Relationship from datastore ➡ object
- Complexity and flexibility comes at a cost
- Less efficient ➡ more overhead ➡ slower
- Not very widely used

Related patterns

- Unit of Work pattern
  - Maintains a list of objects affected by a business transaction and coordinates the writing out of changes and the resolution of concurrency problems.
- Identity Map pattern
  - Ensures that each object gets loaded only once by keeping every loaded object in a map. Looks up objects using the map when referring to them.
- Lazy Load pattern
  - An object that doesn't contain all of the data you need but knows how to get it.
- Most ORM implementations can't be bothered!

📖 *Matt Zandstra, PHP Objects, Patterns and Practice, 3rd edition*



Are you still with me? Good!

OK. Now why would you use an ORM?

- Object oriented access to your data
- Unified access to your data
- Abstracting the underlying storage mechanism
  - Both for the storage platform and the data access language
- Abstracting complexity of data manipulation
- Reduces the time to develop an application
- Easier and faster application maintenance

And why should you not?

- Every abstraction layer adds processing time
- Lots of objects ➡ lots of memory needed
- Generated queries can be sub-optimal
- An ORM can not generate all queries possible
- Some ORM’s duplicate the database layer
- Some ORM’s don’t save you much development time due to their complex implementation

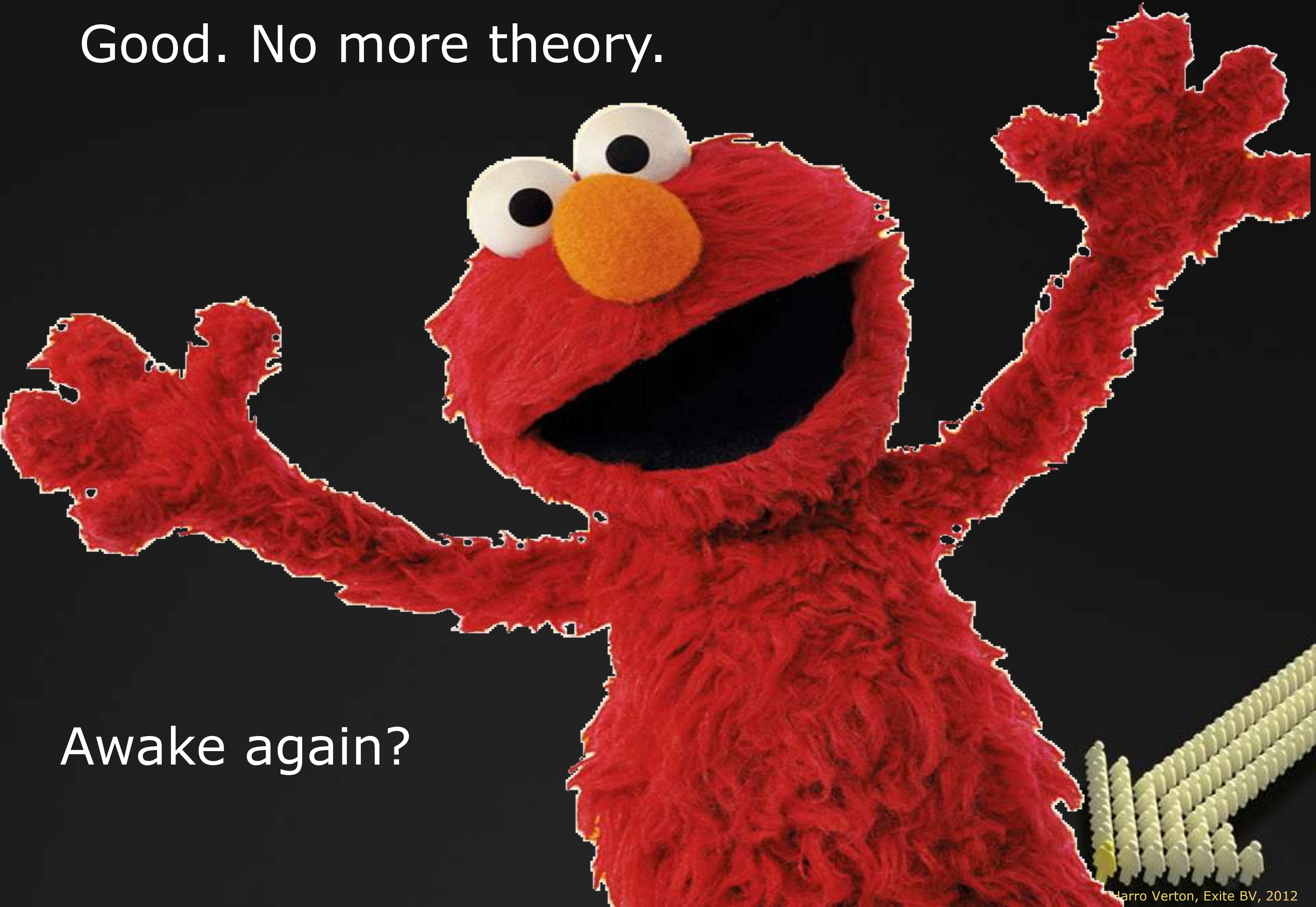
Work around the issues

- Lots of objects ➡ lots of memory needed
  - Think carefully about what you fetch
  - Use standard DB calls when manipulating large volumes
- Generated queries can be sub-optimal, or
- An ORM can not generate all queries possible
  - Determine if this really is a problem for your application
  - Some ORM's allow custom SQL
  - Fall back to standard DB calls as a last resort
- Some ORM's don't save you much development time due to their complex implementation.
  - Ask yourself: did I make the correct choice?

It also depends on your situation

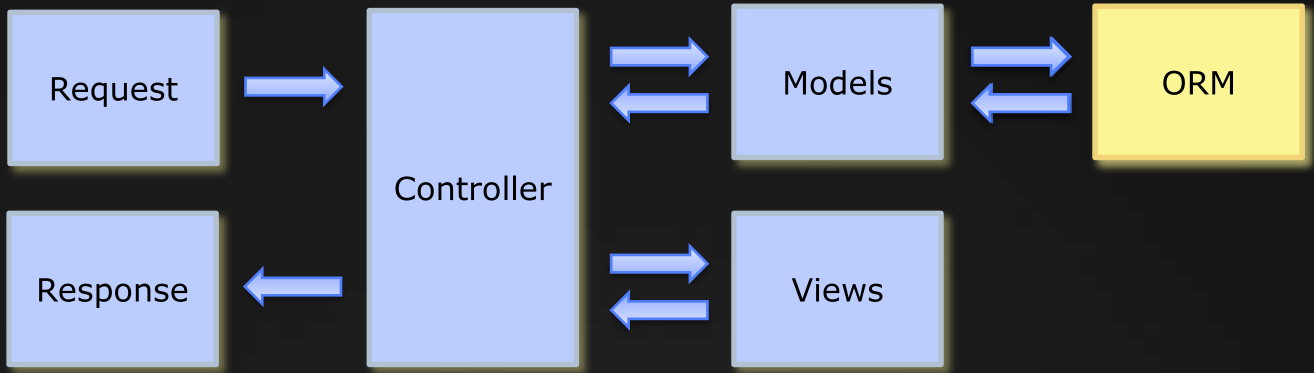
- Your time is free, and no money for servers?
  - Code low-level, every CPU cycle counts!
- Your code has to be maintained by others?
  - Using an ORM is probably a good idea
- You’re working in a team, and time == money?
  - ORM is the way to go, and use separate models too!
- The message here is:
  - CPU Power is cheap
  - Good developers are expensive
  - Spend money where it counts!

Good. No more theory.



Awake again?

Using an ORM in an MVC framework



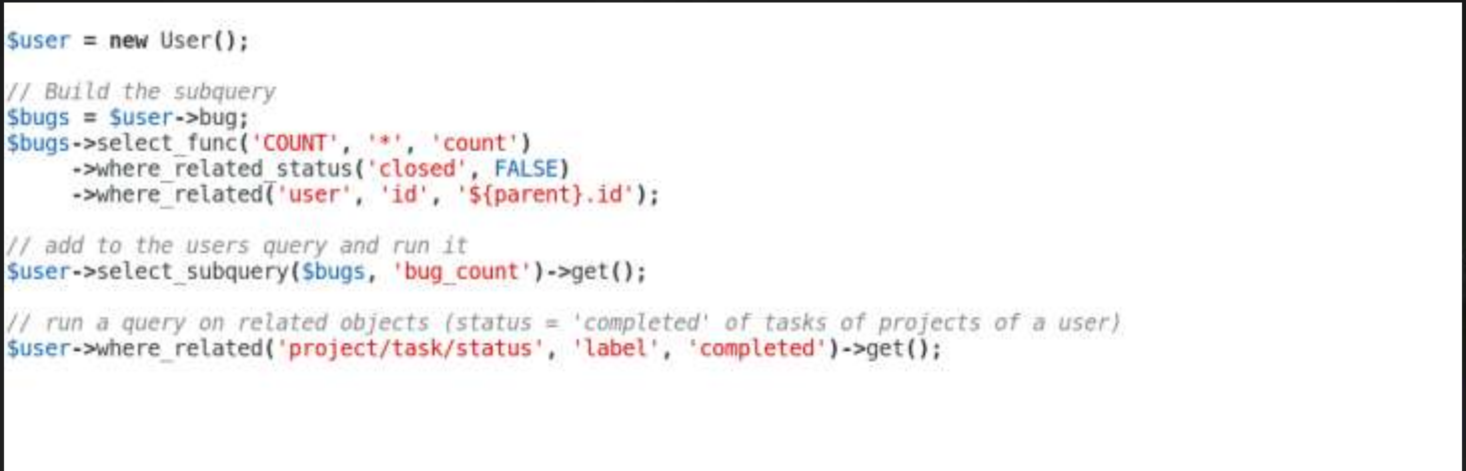
- The ORM is your data abstraction layer
- The Model contains your business logic
- What not to do:
  - have the business logic in the controller and use the ORM as your model
- Some ORM's allow you to merge the functionality of the Model and the ORM
  - but think carefully about the complexity...

Some ORM’s that can be used with CI

- Active Record based
  - Datamapper ORM (http://datamapper.wanwizard.eu)
  - Propel (http://www.propelorm.org)
  - NitroORM (http://nitro-orm.net)
  - PHP ActiveRecord (http://www.phpactiverecord.org)
  - GAS ORM (http://gasorm-doc.taufanaditya.com)
  - Ignited Record (http://www.assembla.com/spaces/IgnitedRecord/wiki)
- Data Mapper based
  - Doctrine (http://www.doctrine-project.org)
- There are more…

Datamapper ORM

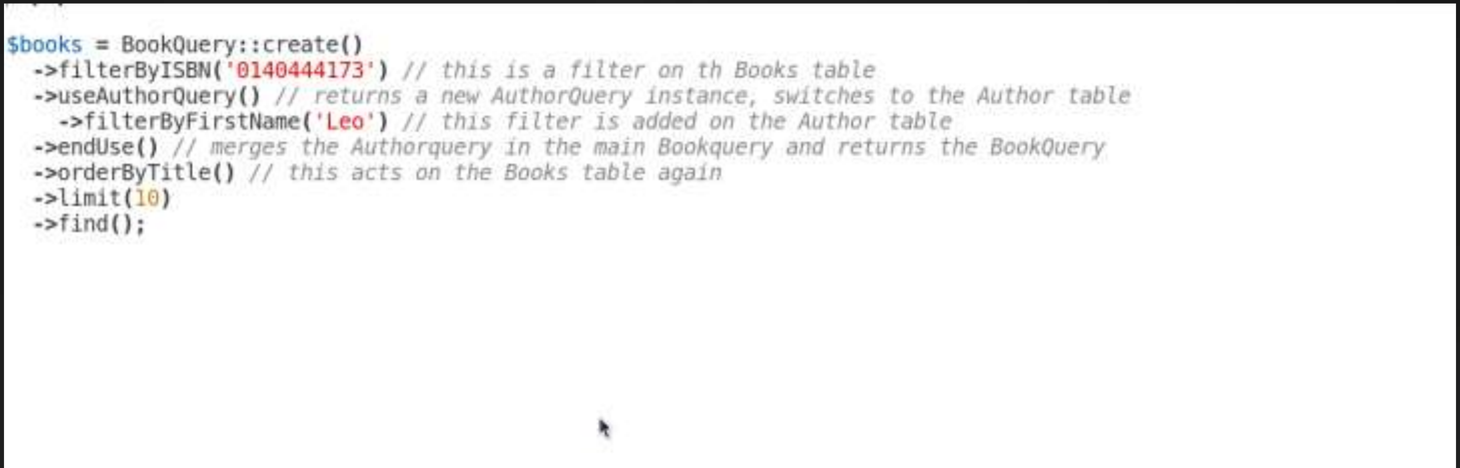
- Pro's:
  - Around for a very long time (< 2008)
  - Very stable and full-featured API
  - Fully integrated into CI
- Con's:
  - Code base has aged due to CI's PHP4 support
  - Implemented as a Model base class



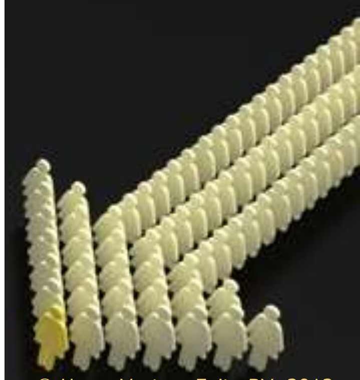
```php
$user = new User();

// Build the subquery
$bugs = $user->bug;
$bugs->select_func('COUNT', '*', 'count')
     ->where_related_status('closed', FALSE)
     ->where_related('user', 'id', '${parent}.id');

// add to the users query and run it
$user->select_subquery($bugs, 'bug_count')->get();

// run a query on related objects (status = 'completed' of tasks of projects of a user)
$user->where_related('project/task/status', 'label', 'completed')->get();
```

Propel

- Pro’s:
  - Around even longer (since 2005)
  - Very rich feature set
- Con’s:
  - Has it’s own database layer
  - Uses XML schema’s to generate the model code
  - Requires command-line access



```
$books = BookQuery::create()
  ->filterByISBN('0140444173') // this is a filter on th Books table
  ->useAuthorQuery() // returns a new AuthorQuery instance, switches to the Author table
    ->filterByFirstName('Leo') // this filter is added on the Author table
  ->endUse() // merges the Authorquery in the main Bookquery and returns the BookQuery
  ->orderByTitle() // this acts on the Books table again
  ->limit(10)
  ->find();
```

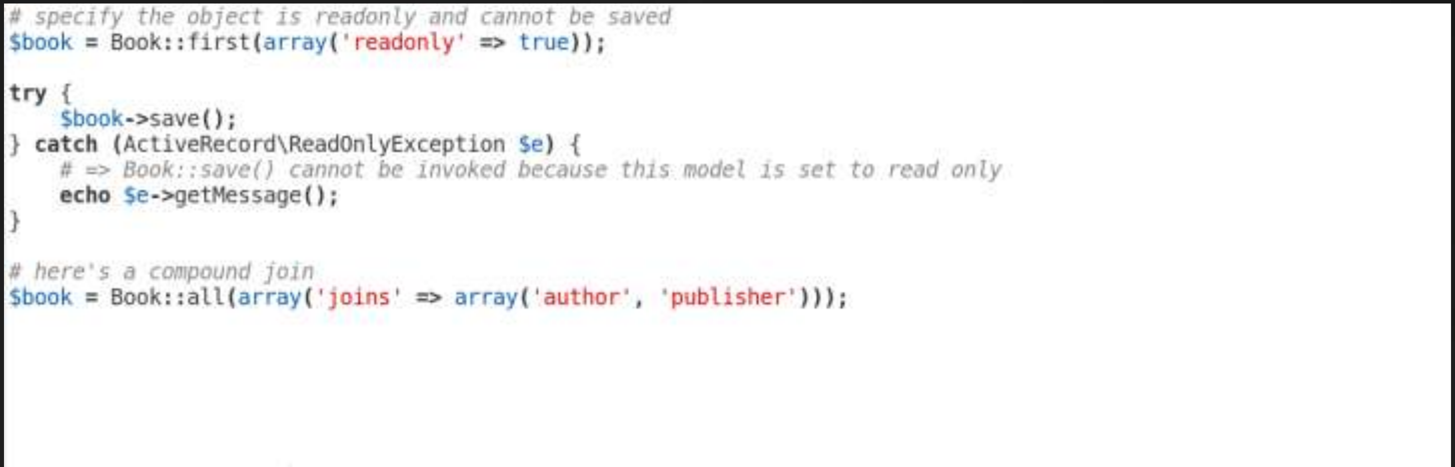


NitroORM

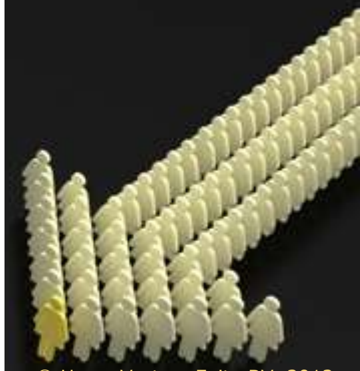
- Pro’s:
  - Very promising feature roadmap
- Con’s:
  - Very new (7 months), not mature
  - Lots of features still missing
  - No visible activity after the last release (11/2011)
  - Requires PHP 5.3+ (note that CI only requires 5.1.6+)
    - Your host might not support it!
  - Documentation is very sparse

PHP ActiveRecord

- Pro's:
  - Complete feature set, including validation
  - CI integration available (how-to or using a spark)
- Con's:
  - No visible activity:
    - Last stable release almost 2 years old
    - Last nightly build 5 months old

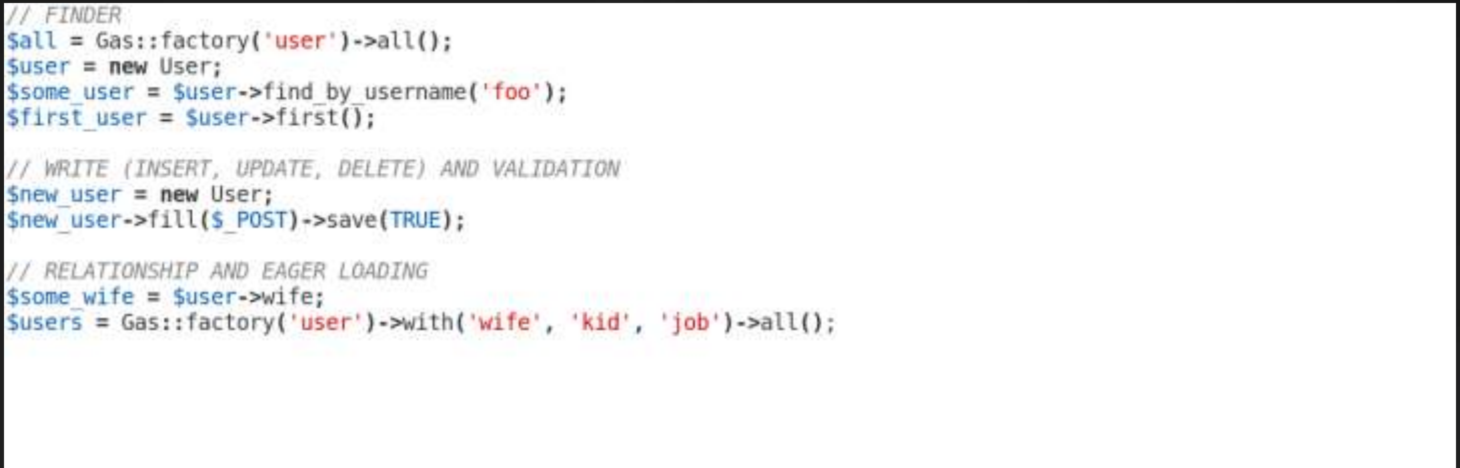


```
# specify the object is readonly and cannot be saved
$book = Book::first(array('readonly' => true));

try {
    $book->save();
} catch (ActiveRecord\ReadOnlyException $e) {
    # => Book::save() cannot be invoked because this model is set to read only
    echo $e->getMessage();
}

# here's a compound join
$book = Book::all(array('joins' => array('author', 'publisher')));
```



GAS ORM

- Pro’s:
  - Fully integrated into CI
  - Built-in query caching
- Con’s:
  - Only a few months old, no roadmap known
  - Not a lot of documentation available



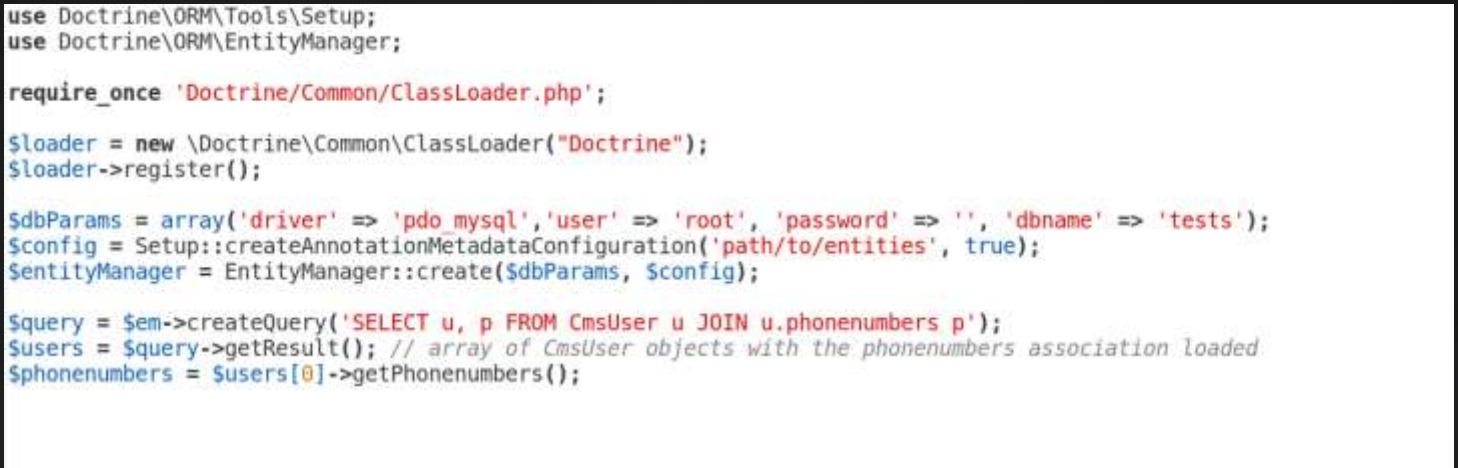
```
// FINDER
$all = Gas::factory('user')->all();
$user = new User;
$some_user = $user->find_by_username('foo');
$first_user = $user->first();

// WRITE (INSERT, UPDATE, DELETE) AND VALIDATION
$new_user = new User;
$new_user->fill($_POST)->save(TRUE);

// RELATIONSHIP AND EAGER LOADING
$some_wife = $user->wife;
$users = Gas::factory('user')->with('wife', 'kid', 'job')->all();
```

Ignited Record

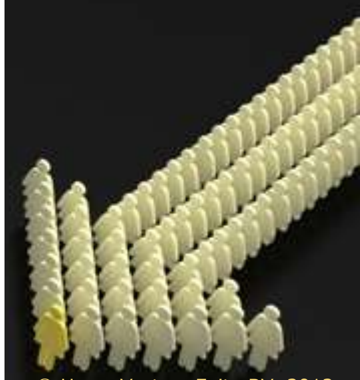
- Pro's:
  - Eh...
- Con's:
  - No visible activity since 2008
  - Compatible with CI 2.x?
- Consider this one dead and buried…

Doctrine

- Pro’s:
  - Full featured, implements the full “Fowler stack”
- Con’s:
  - Uses a command line interface
  - Very complex to work with
  - No query methods, the DQL must be used



```
use Doctrine\ORM\Tools\Setup;
use Doctrine\ORM\EntityManager;

require_once 'Doctrine/Common/ClassLoader.php';

$loader = new \Doctrine\Common\ClassLoader("Doctrine");
$loader->register();

$dbParams = array('driver' => 'pdo_mysql','user' => 'root', 'password' => '', 'dbname' => 'tests');
$config = Setup::createAnnotationMetadataConfiguration('path/to/entities', true);
$entityManager = EntityManager::create($dbParams, $config);

$query = $em->createQuery('SELECT u, p FROM CmsUser u JOIN u.phonenumbers p');
$users = $query->getResult(); // array of CmsUser objects with the phonenumbers association loaded
$phonenumbers = $users[0]->getPhonenumbers();
```



Considerations when making a choice

- Does my project benefit from using an ORM?
- Which pattern am I going to use?
- Does it have all functionality my project requires?
- How about support and documentation?
- Other considerations:
  - How easy is the integration into CodeIgniter?
  - Do they provide CodeIgniter support?
  - How large is the user base?
  - Is it still actively maintained?
  - Is their code stable enough to invest time in?
  - Check the CI forums for possible issues!

Question Time !

Thank you for attending!

- Tomorrow:
  - Masterclass on Datamapper ORM
  - Download the files from the programme page on the website

- More questions?
  - Find me here today or tomorrow
  - Use the CodeIgniter forums
  - Twitter: @WanWizard
  - Email: codeigniter@wanwizard.eu