ORM классы.

DomainObjectAssembler- класс, управляющий процессами по сохранению и извлечению данных.

PersistenceFactory- абстрактный класс-фабрика компонентов, необходимых в ходе работы системы…

DomainObjectFactory- инкапсулирует функции создания объектов. Абстрактный.

ObjectWather- автоматизирует процесс, посредством которого объекты сохраняются в базу данных, и гарантирует, что обновляются только измененные объекты, а вставляются- только вновь созданные. А также отсрочивает создание объектов и запросы к базе данных, пока они действительно не понадобятся.

IdentityObject- позволяет создавать критерии запросов. Абстрактный.

UpdateFactory- инкапсулирует логику построения SQL-запросов на обновление и вставку. Абстрактный.

SelectionFactory- инкапсулирует логику построения SQL-запросов на выборку.

Collection- реализация интерфейса Iterator, позволяющая работать с группой (коллекцией) объектов. Абстрактный.

Автор, статья, журнал- «сущности». Для каждой сущности требуется определить дочерние классы UpdateFactory, IdentityObject, DomainObjectFactory, Collection.

Подробнее

1. DomainObjectAssembler. Конструктору передается один аргумент- объект класса-фабрики PersistenceFactory. Т.к. для каждой сущности создается своя фабрика, то объект DomainObjectAssembler также создается для каждой сущности отдельно. Основная задача класса- выполнять запросы к БД.

*$factory= PersistenceFactory::getFactory('Author');*

*$finder= new DomainObjectAssembler($factory);*

*$p\_factory=PersistenceFactory::getFactory('Paper');*

*$p\_finder= new DomainObjectAssembler($p\_factory);*

Основные методы.

Find(), findOne()- позволяют создать коллекцию объектов или один объект соответственно из БД. В качестве аргумента- объект IdentityObject, с заданными параметрами запроса.

*$idobj=$factory->getIndentityObject()->field('id')->eq(1);*

*$a1=$finder->findOne($idobj);- так мы получим автора, найденного по id.*

Insert() – принимает в качестве аргумента объект-сущность и выполняет вставку объекта или его обновление. Однако непосредственно из выполняющего кода не вызывается. Причина в существовании Lazy Load. (поясняется далее)

2. PersistenceFactory. Конструктору передается имя сущности. Практически все методы вызываются из DomainObjectAssembler, кроме getIdentityObject(), когда нам требуется объект IdentityObject, чтобы передать его методам find()/ findOne().

*$factory= PersistenceFactory::getFactory('Author');*

*$idobj=$factory->getIndentityObject()->field('id')->eq(16);*

*$author= $finder->findOne($idobj);*

3. ObjectWather. В классе реализованы шаблоны Unit of Work и Lazy Load. Суть первого в том, что мы не создает один и тот же объект их БД дважды. А также позволяет сохранять и обновлять только те объекты, которые в этом нуждаются. Lazy Load позволяет избегать массовых обращений к БД.

От нас требуется лишь одно- не вызывать «искусственно» метод insert() класса DomainObjectFactory, а вызывать следующий метод: ????????

*ObjectWatcher::instance()->performOperations();*

Он и определяет какие объекты нуждаются в вставке в БД, а какие в обновлении. Смотрим тест lessons\ORM ver 0.1\data\test.ptp

4. IdentityObject. Объект класса хранит в себе все параметры, необходимые для составления запроса на SELECT. Пройдемся по методам, которые позволяют это сделать.

addWhat(array $fields)- метод позволяет задать поля БД, по которым мы хотим осуществить выборку. По умолчанию «\*».

*$idobj = new AuthorIdentityObject();*

*$idobj->addWhat(array('name','family'));*

Получим такой запрос: SELECT name, family FROM author WHERE…

addJoin($type='INNER',array $tables,array $raws) – метод позволяет задать операцию соединения таблиц.

$type- тип соединения (один из следующих- LEFT, RIGHT, INNER, OUTER). По умолчанию INNER.

$tables- массив имен соединяющихся таблиц.

$raws- массив полей этих таблиц, по которым идет соединение.

*$idobj->addJoin('INNER',array('author','paper\_authors'),array('id','author\_id'));*

В результате: SELECT \* FROM author INNER JOIN paper\_authors ON author.id=paper\_authors.author\_id

function addOrder($order)- задает сортировку результата выборки.

$order- ассоциативный массив вида array(field=>type,…), где field- имя поля, по которому сортируется результат, type- тип сортировки- DESC, ASC.

*$idobj->addOrder(array('name'=>'DESC'));*

Результат: SELECT \* FROM author …… OREDER BY name DESC

Чтобы построить условия для WHERE используется следующая группа методов.

field($fieldname)- задает имя поля БД, входящее в условие.

eq($value), lt($value), rgt($value)- задают простые операторы сравнения- «=», «<», «>» соответственно. $value- значение для сравнения.

*$idobj->field('id')->eq(1)*.

Результат: SELECT \* FROM author WHERE id=1.

like($value,$not=false)- использование предиката LIKE для сравнения строк. $not в значении true даст предикат NOT LIKE.

*$idobj->field('family')->like(‘%ов’)*.

Результат: SELECT \* FROM author WHERE family LIKE ‘%ов’

По умолчанию несколько условий объединяются с помощью AND.

*$idobj->field('name')->eq('Андрей')->field("family")->like("%ов",true)*

*Получим SELECT \* FROM author WHERE name=’Андрей’ AND family NOT LIKE ‘%ов’.*

Чтобы использовался OR, необходимо воспользоваться методом changeLink():

*$idobj->field('name')->eq('Андрей')->field("family")->like("%ов",true)->changeLink();*

*Получим SELECT \* FROM author WHERE name=’Андрей’ OR family NOT LIKE ‘%ов’*

Примечание: в IdentityObject не хранится формулировка запроса, а только его составляющие. Сам запрос составляется в SelectionFactory, а выполняется в DomainObjectAssembler.

Все примеры имеются в тесте \lessons\ORM ver 0.1\ AuthorSelectionFactoryTest.php

Что необходимо реализовать. (в качестве примера рассмотрим сущность статья)

1. Collection

Метод targetClass().

*class PaperCollection extends Collection{*

*function targetClass(){*

*return "Paper";*

*}*

*}*

От него требуется только чтобы он возвращал имя сущности.

1. IdentityObject

Определить конструктор:

*class PaperIdentityObject extends IdentityObject{*

*function \_\_construct($field=null){*

*parent::\_\_construct($field, array('id','title','content','journal\_id'));*

*}*

*}*

Необходимо перечислить допустимые поля базы данных, по которым будет производиться выборка. (подчеркнуто красным)

1. DomainObjectFactory

Метод doCreateObject()- отвечает за создание объекта на основе выборки из БД.

*function doCreateObject(array $array){*

*$obj= new Paper();*

*$obj->setTitle($array['title']);*

*$obj->setContent($array['content']);*

*$obj->setId($array['id']);*

*$author\_collection=$this->createCollection($obj->getId());*

*$obj->setAuthors($author\_collection);*

*return $obj;*

*}*

На вход подается массив- результат запроса к БД. Создается требуемый объект и задаются все его параметры. Если объект «сложный», т.е. содержит коллекцию (как статья- коллекцию авторов), необходимо обратиться к методу createCollection(), которому передается id только что созданного объекта.

Метод createCollection()- создает коллекцию объектов.

*function createCollection($id){*

*$factory= PersistenceFactory::getFactory('Author');*

*$finder= new DomainObjectAssembler($factory);*

*$idobj=$factory->getIndentityObject();*

*$idobj->addJoin('INNER',array('author','paper\_authors'),array('id','author\_id'));*

*$idobj->field('paper\_id')->eq($id);*

*return $author\_collection=$finder->find($idobj);*

*}*

В сущности тут создается запрос к БД, который затем выполняется и на выходе имеем коллекцию объектов.

Метод targetClass()- аналогичен в Collection.

1. UpdateFactory

Основной метод- newUpdate(). Аргумент- объект-сущность.

*function newUpdate(DomainObject $obj){*

*$id= $obj->getId();*

*$cond=null;*

*$values['title']=$obj->getTitle();*

*$values['content']=$obj->getContent();*

*if ($id >-1){*

*$cond['id']=$id;*

*return $this->buildStatement('paper',$values,$cond);*

*}*

*return $this->buildStatement('paper',$values,$cond,true);*

*}*

Тут мы задаем в массив $values значения, которые будем вставлять в БД. Проверка на наличие id позволяет определить какой объект перед нами- созданный из БД или совершенно новый (т.е. id еще не имеет).

В конечном итоге необходимо передать методу buildStatement следующие параметры: имя таблицы которую будем обновлять, массив значений; содержимое массива $cond позволит методу понять обновлять объект или вставлять новый объект в БД. Последний параметр true передается в том случае, если вставляется сложный объект и необходима также вставка в таблицу связей.

В общем случае на выходе мы имеем готовый запрос, который еще предстоит выполнить и массив значений, которые мы и будем вставлять в БД.

Метод InsertLink()- осуществляет вставку в таблицу связей. Аргумент- объект-сущность.

*function InsertLink(DomainObject $obj){*

*$authors=$obj->getAuthors();*

*$links= array('paper\_id','author\_id'); //поля таблицы связей*

*$query=$this->buildLinks('paper\_authors',$links); // имя таблицы связей + массив полей*

*foreach ($authors as $author){ // задаем массив для вставки в БД*

*$values[]=array($obj->getId(),$author->getId());*

*}*

*return array($query,$values);*

*}*

Вытаскиваем из объекта коллекцию (для статьи- коллекция авторов). Задаем массив из полей таблицы связей. Вызываем метод buildLinks(), который составит нам sql-запрос. Далее нужно только составить массив значений, которые будут вставляться в таблицу связей. В данном примере это пары значений- id статьи и id каждого автора.

А дальше что?

Смотрим как это все работат на тестах в lessons\ORM ver 0.1\data\ mainAuthorTest.php, mainPaperTest.php, mainJournalTest.php.