Camada de persistência

A camada de persistência do Miolo está baseada nos trabalhos de Ambler e Artyom Rudoy (v. Referências). A implementação atual tem as seguintes características:

- O mecanismo de persistência está encapsulado. A classe Business passa a herdar da classe de persistência, tornando os objetos de negócio virtualmente persistentes. Estão disponíveis métodos tais como **save**, **delete** e **retrieve** que tratam automaticamente o acesso ao banco de dados.
- Ações sobre múltiplos objetos. Mecanismos para recuperação e remoção de múltiplos objetos. Os mecanismos de recuperação permitem retornar objetos Query (com acesso ao ResultSet via camada DAO do Miolo) ou Cursores (um cursor está implementado como um array de objetos).
- Suporte a "lazy read" através do uso de proxies. Um objeto proxy permite recuperar apenas alguns atributos do objeto, evitando o overhead de recuperar todos os atributos. Isto é útil em situações tais como a exibição de listas de seleção, por exemplo.
- Suporte a associações. Quando um objeto é recuperado, removido ou atualizado, a mesma ação pode ser realizada nos objetos associados, se desejado. As associações do tipo ManyToMany podem ser tratadas automaticamente pela camada de persistência.
- Suporte a herança, tornando possível mapear uma árvore de herança para um esquema no banco de dados.
- Suporte a transações, geração automática de identificadores (OID), geração automática do comando SQL e acesso a diferentes bancos de dados, que são características implementadas pela camada DAO do Miolo.
- Suporte a conversão de valores de atributos, através de classes de conversão.

Neste tutorial o seguinte modelo de classes será usado (fig. 1), com o esquema do banco de dados correspondente (fig.2). Estão indicados os atributos auxiliares relativos à navegação das associações (geralmente não exibidos nos diagramas UML) para melhor compreensão do processo de mapeamento.

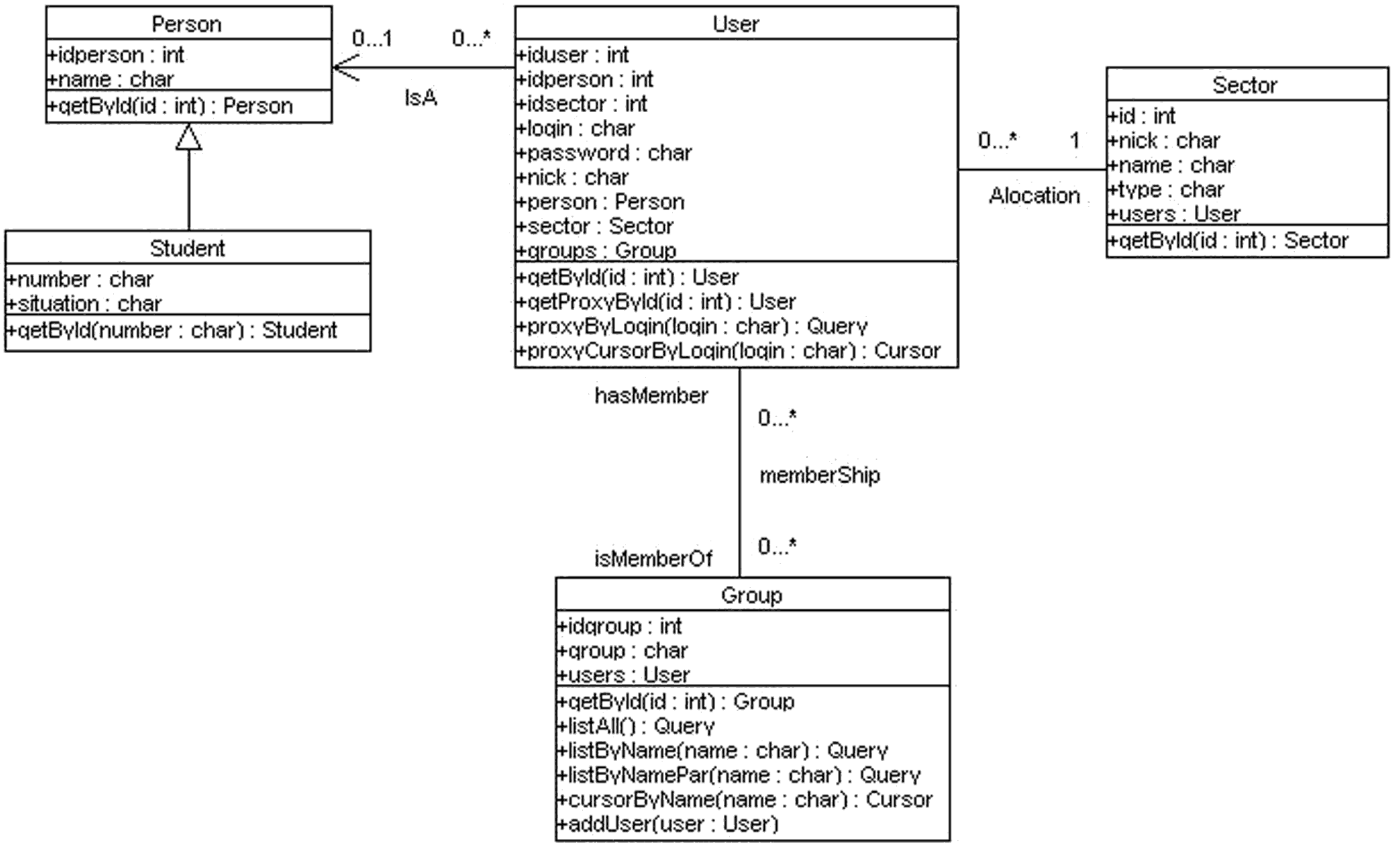

Figura 1 – Modelo de Classes

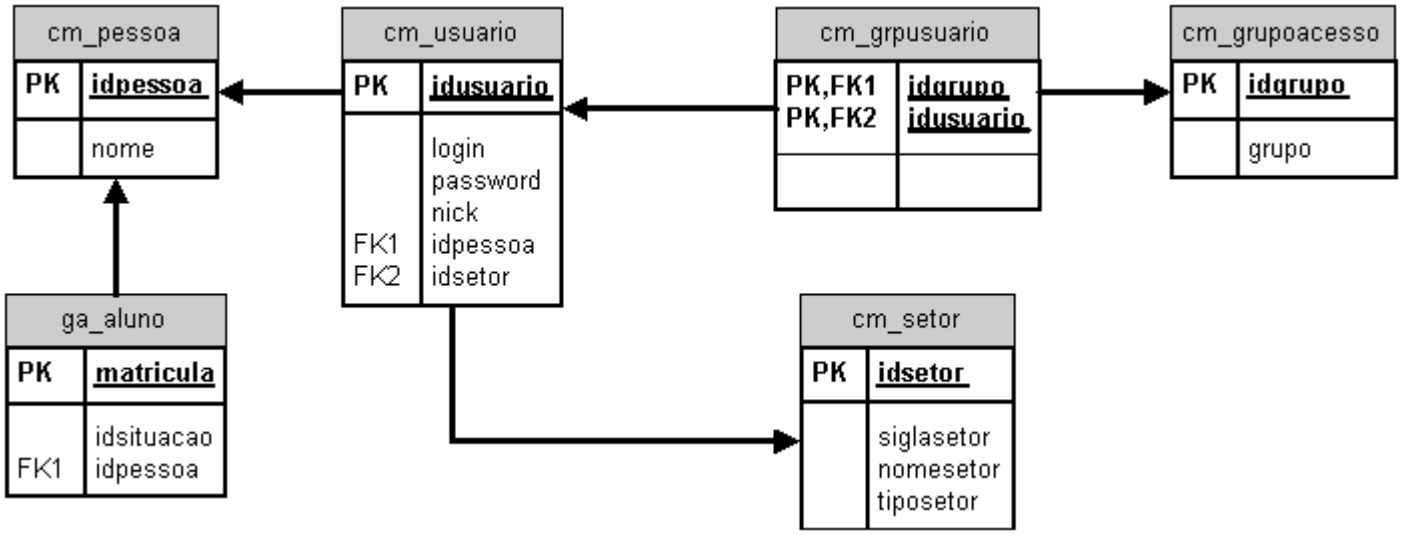

Figura 2 – Esquema do banco de dados relacional

Mapeamento

No contexto da camada de persistência, o mapeamento diz respeito a como são representados os objetos, seus atributos/propriedades e suas associações, em termos do modelo relacional.
Nesta implementação estamos usando arquivos XML para representar o mapeamento das classes persistentes e das classes associativas (classes que são usadas dentro da camada de persistência para representar as associações muitos-para-muitos - ManyToMany). São criados mapas para as classes, para os atributos, para as associações, para as colunas e para as tabelas, permitindo uma grande flexibilidade no processo de mapeamento.
Os mapas XML são criados com o nome <nome_da_classe>.xml e ficam localizados no subdiretório /map dentro do diretório <modulo>/classes.
Os exemplos seguintes mostram os mapas XML criados para cada uma das classes.

Classe USER

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<map>
   <moduleName>persistence</moduleName>
   <className>user</className>
   <tableName>cm_usuario</tableName>
   <databaseName>common</databaseName>
   <attribute>
      <attributeName>iduser</attributeName>
         <columnName>idusuario</columnName>
         <key>primary</key>
         <idgenerator>seq_cm_usuario</idgenerator>
   </attribute>
   <attribute>
      <attributeName>login</attributeName>
         <columnName>login</columnName>
         <proxy>true</proxy>
   </attribute>
   <attribute>
      <attributeName>password</attributeName>
         <columnName>password</columnName>
   </attribute>
   <attribute>
      <attributeName>nick</attributeName>
         <columnName>nick</columnName>
   </attribute>
   <attribute>
      <attributeName>idperson</attributeName>
         <columnName>idpessoa</columnName>
         <proxy>true</proxy>
   </attribute>
   <attribute>
      <attributeName>idsector</attributeName>
         <columnName>idsetor</columnName>
         <proxy>true</proxy>
   </attribute>
   <attribute>
```

```
      <attributeName>person</attributeName>
   </attribute>
   <attribute>
      <attributeName>groups</attributeName>
   </attribute>
   <attribute>
      <attributeName>sector</attributeName>
   </attribute>

   <association>
      <toClassModule>persistence</toClassModule>
      <toClassName>person</toClassName>
      <cardinality>oneToOne</cardinality>
      <target>person</target>
         <retrieveAutomatic>true</retrieveAutomatic>
         <saveAutomatic>true</saveAutomatic>
         <entry>
            <fromAttribute>idperson</fromAttribute>
            <toAttribute>idperson</toAttribute>
      </entry>
   </association>

   <association>
      <toClassModule>persistence</toClassModule>
      <toClassName>group</toClassName>
      <associativeClassModule>persistence</associativeClassModule>
      <associativeClassName>groupuser</associativeClassName>
      <cardinality>manyToMany</cardinality>
      <target>groups</target>
         <retrieveAutomatic>false</retrieveAutomatic>
         <saveAutomatic>false</saveAutomatic>
         <direction>
            <fromAttribute>users</fromAttribute>
            <toAttribute>groups</toAttribute>
      </direction>
   </association>

   <association>
      <toClassModule>persistence</toClassModule>
      <toClassName>sector</toClassName>
      <cardinality>oneToOne</cardinality>
      <target>sector</target>
         <retrieveAutomatic>true</retrieveAutomatic>
         <saveAutomatic>false</saveAutomatic>
         <entry>
            <fromAttribute>idsector</fromAttribute>
            <toAttribute>id</toAttribute>
      </entry>
   </association>
</map>
```

Classe GROUP

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<map>
   <moduleName>persistence</moduleName>
   <className>group</className>
   <tableName>cm_grupoacesso</tableName>
   <databaseName>common</databaseName>
   <attribute>
      <attributeName>idgroup</attributeName>
         <columnName>idgrupo</columnName>
         <key>primary</key>
   </attribute>
   <attribute>
      <attributeName>group</attributeName>
         <columnName>grupo</columnName>
   </attribute>
   <attribute>
      <attributeName>users</attributeName>
   </attribute>

   <association>
      <toClassModule>persistence</toClassModule>
```

```
      <toClassName>user</toClassName>
      <associativeClassModule>persistence</associativeClassModule>
      <associativeClassName>groupuser</associativeClassName>
      <cardinality>manyToMany</cardinality>
      <target>users</target>
         <retrieveAutomatic>false</retrieveAutomatic>
         <saveAutomatic>true</saveAutomatic>
         <deleteAutomatic>true</deleteAutomatic>
         <direction>
            <fromAttribute>groups</fromAttribute>
            <toAttribute>users</toAttribute>
      </direction>
   </association>
</map>
```

Classe SECTOR

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<map>
   <moduleName>persistence</moduleName>
   <className>sector</className>
   <tableName>cm_setor</tableName>
   <databaseName>common</databaseName>
   <attribute>
      <attributeName>id</attributeName>
         <columnName>idsetor</columnName>
         <key>primary</key>
   </attribute>
   <attribute>
      <attributeName>nick</attributeName>
         <columnName>siglasetor</columnName>
   </attribute>
   <attribute>
      <attributeName>name</attributeName>
         <columnName>nomesetor</columnName>
   </attribute>
   <attribute>
      <attributeName>type</attributeName>
         <columnName>tiposetor</columnName>
   </attribute>
   <attribute>
      <attributeName>users</attributeName>
   </attribute>

   <association>
      <toClassModule>persistence</toClassModule>
      <toClassName>user</toClassName>
      <cardinality>oneToMany</cardinality>
      <target>users</target>
         <retrieveAutomatic>true</retrieveAutomatic>
         <saveAutomatic>false</saveAutomatic>
         <deleteAutomatic>true</deleteAutomatic>
         <inverse>true</inverse>
         <entry>
            <fromAttribute>idsector</fromAttribute>
            <toAttribute>id</toAttribute>
      </entry>
   </association>
</map>
```

Classe PERSON

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<map>
   <moduleName>persistence</moduleName>
   <className>person</className>
   <tableName>cm_pessoa</tableName>
   <databaseName>common</databaseName>
   <attribute>
      <attributeName>idperson</attributeName>
         <columnName>idpessoa</columnName>
         <key>primary</key>
   </attribute>
   <attribute>
      <attributeName>name</attributeName>
```

```
        <columnName>nome</columnName>
        <converter>
            <converterName>CaseConverter</converterName>
               <parameter>
                   <parameterName>case</parameterName>
                   <parameterValue>upper</parameterValue>
               </parameter>
            </converter>
   </attribute>
</map>
```

Classe STUDENT

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<map>
   <moduleName>persistence</moduleName>
   <className>student</className>
   <tableName>ga_aluno</tableName>
   <databaseName>common</databaseName>
   <extends>
       <moduleName>tutorial3</moduleName>
       <className>person</className>
   </extends>
   <attribute>
      <attributeName>number</attributeName>
        <columnName>matricula</columnName>
        <key>primary</key>
   </attribute>
   <attribute>
      <attributeName>situation</attributeName>
        <columnName>idsituacao</columnName>
   </attribute>
   <attribute>
      <attributeName>idperson</attributeName>
        <columnName>idpessoa</columnName>
        <reference>idperson</reference>
   </attribute>
</map>
```

Classe associativa GROUPUSER

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<map>
   <moduleName>persistence</moduleName>
   <className>groupuser</className>
   <tableName>cm_grpusuario</tableName>
   <databaseName>common</databaseName>
   <attribute>
      <attributeName>iduser</attributeName>
        <columnName>idusuario</columnName>
   </attribute>
   <attribute>
      <attributeName>idgroup</attributeName>
        <columnName>idgrupo</columnName>
   </attribute>

   <association>
      <toClassModule>persistence</toClassModule>
      <toClassName>user</toClassName>
      <cardinality>oneToOne</cardinality>
      <target>users</target>
        <retrieveAutomatic>true</retrieveAutomatic>
        <saveAutomatic>true</saveAutomatic>
        <entry>
           <fromAttribute>iduser</fromAttribute>
           <toAttribute>iduser</toAttribute>
      </entry>
   </association>

   <association>
      <toClassModule>persistence</toClassModule>
      <toClassName>group</toClassName>
      <cardinality>oneToOne</cardinality>
      <target>groups</target>
        <retrieveAutomatic>true</retrieveAutomatic>
```

```
        <saveAutomatic>true</saveAutomatic>
        <entry>
           <fromAttribute>idgroup</fromAttribute>
           <toAttribute>idgroup</toAttribute>
      </entry>
   </association>
</map>
```

O elemento principal do mapa XML é denominado <map>.

<moduleName>	nome do módulo onde a classe de negócios está definida
<className>	nome da classe de negócios, dentro do módulo <moduleName>
<tableName>	nome da tabela que mapeia a classe. Nesta versão uma classe pode ser mapeada para apenas uma tabela.
<databaseName>	nome da configuração do banco de dados no arquivo miolo.conf
<extends>	Usado no suporte à herança entre classes. <moduleName> e <className> indicam qual é a superclasse.

Atributos
Os atributos são definidos dentro da tag <attribute>.

<attributeName>	nome do atributo conforme definido na classe
<columnName>	nome da coluna que representa o atributo na tabela <tableName>. Atributos que são usados apenas para representar a associação entre duas classes geralmente não têm colunas associadas (p.ex. o atributo person na classe User).Opcional.
<key>	indica, quando este atributo é uma chave da tabela, qual o tipo de chave. Pode assumir dois valores: primary e foreign. Opcional.
<idgenerator>	quando o atributo é chave, esta tag pode indicar qual o nome do IdGenerator a ser passado para a camada DAO do Miolo, para a geração automática de OID. Opcional.
<proxy>	indica que este atributo será recuperado com objetos proxy. O valor default é true. Atributos chave são sempre recuperados com objetos proxy.
<reference>	Usado no suporte à herança entre classes. Indica o nome do atributo da superclasse que é referenciado por este atributo. É usado para a construção do join entre as tabelas da superclasse e da classe sendo mapeada.
<attributeIndex>	na criação de atributos indexados (arrays), pode-se informar qual será o índice associado à coluna descrita em <columnName>. Isto permite agrupar vários campos da tabela em um único atributo do objeto.

Associações
As associações representam os relacionamentos existentes entre as classes. São definidas a partir do ponto de vista da classe sendo mapeada. Suas características estão dentro da tag <association>.

<toClassModule>	nome do módulo da classe associada
<toClassName>	nome da classe associada, dentro do módulo <toClassModule>
<cardinality>	cardinalidade da associação. Refere-se a quantos objetos da classe associada estão relacionados com a classe sendo mapeada. Os valores possíveis são: oneToOne, oneToMany e manyToMany.
<target>	indica qual o atributo vai receber o objeto (ou conjunto de objetos) que for recuperado através da associação. É também o nome usado para identificar a associação. Por exemplo, a associação com <target> igual a 'person', na classe User, indica que um objeto da classe Person, associado com este objeto User, vai ser recuperado e armazenado no atributo 'person'.
<retrieveAutomatic> <saveAutomatic> <deleteAutomatic>	indica se as operações de recuperação, armazenamento e remoção (respectivamente) serão realizadas automaticamente na classe associada. Estas tags são opcionais e o valor default é false. Cuidado especial deve ser tomado nesta definição pois a recuperação (ou armazenamento ou remoção) de um simples objeto pode ocasionar a recuperação automática de dezenas de outros, dependendo do valor destas tags. De certa forma é usada para mapear (e sobrepor) as definições de integridade referencial declaradas no esquema relacional.
<orderAttribute>	Indica o(s) atributo(s) que serão usados para ordenar os objetos retornados da classe associada. <orderAttributeName> indica o nome do atributo e <orderAttributeDirecton> indica a direção ('ascend' ou 'descend').
<indexAttribute>	Indica, nas associações oneToMany e manyToMany, o atributo cujos valores serão usados como índices do array de objetos retornado pela recuperação na classe associada.

<entry>	descreve a associação entre os atributos da classe mapeada e os atributos da classe associada. Dentro da tag <entry>, <fromAttribute> indica o atributo da classe mapeada e <toAttribute> indica o atributo da classe associada. Podem existem várias tags <entry>.
<inverse>	indica a direção da associação. Se <inverse> é false (default), o atributo <fromAttribute> é acessado na classe mapeada e o atributo <toAttribute> é acessado na classe <toClassModule>:<toClassName>. Se <inverse> é true, o atributo <fromAttribute> é acessado na classe <toClassModule>:<toClassName> e o atributo <toAttribute> é acessado na classe mapeada. Esta configuração é necessária para o caso em que a direção da associação for inversa à direção da referência no banco de dados. Como exemplo do uso de <inverse> tomemos por exemplo a associação entre as classes USER e SECTOR: a) <inverse> = false Classe mapeada: USER Classe associada: SECTOR <toClassModule>:<toClassName> : persistence:sector <fromAttribute>: idsector (atributo na classe USER) <toAttribute>: id (atributo na classe SECTOR) A direção da associação (user->sector) coincide com a direção foreignKey -> primaryKey b) <inverse> = true Classe mapeada: SECTOR Classe associada: USER <toClassModule>:<toClassName> : persistence:user <fromAttribute>: idsector (atributo na classe USER) <toAttribute>: id (atributo na classe SECTOR) A direção da associação (sector->user) é inversa à direção foreignKey -> primaryKey
<associativeClassModule>	nome do módulo da classe associativa, usada no mapeamento de associações ManyToMany
<associativeClassName>	nome da classe associativa, dentro do módulo <toClassModule>, usada no mapeamento de associações ManyToMany. Esta classe não existe no modelo de classes de negócio. É criada internamente na camada de persistência para permitir o acesso à tabela de ligação no modelo relacional (uma vez que no modelo relacional os relacionamentos são sempre OneToOne ou OneToMany). Se no modelo de classes de negócio existir uma classe associativa (geralmente com atributos próprios) ela deve ser mapeada normalmente como uma classe persistente.
<direction>	descreve a direção de uma associação com cardinalidade ManyToMany. Dentro da tag <direction>, <fromAttribute> e <toAttribute> indicam os nomes da associações definidas no mapa da classe associativa que simulam o relacionamento com a tabela de ligação no modelo relacional. A ordem em que os atributos são colocados define a direção da associação. Por exemplo, no mapa da classe USER, a associação com <target> igual a 'groups' (que indica a associação entre USER e GROUP) tem cardinalidade ManyToMany (um usuário pode estar em vários grupos, um grupo pode ter vários usuários). É definida a classe associativa GROUPUSER (interna à camada de persistência), em cujo mapa encontramos duas associações (groups e users). A tag <direction> em USER indica que para materializar a associação com GROUP, deve ser percorrida a associação USER<-GROUPUSER e depois GROUPUSER->GROUP.

Classes de conversão de valores

A camada de persistência permite definir métodos de conversão de valores, que são podem ser usados antes e/ou depois de um determinado valore ser gravado/acessado no banco de dados.
Estes métodos de conversão devem ser definidos em classes próprias, que implementem a interface IConverter (definida em <miolo>/classes/persistence/converter). A camada já fornece duas classes pré-definidas: TrivialConverter e CaseConverter. TrivialConverter é a classe default, usada quando não for definida nenhuma outra classe, e na realidade não realiza nenhuma conversão. A classe CaseConverter é fornecida

como exemplo de implementação, e pode ser usada para converter os dados string para maiúsculas ou minúsculas.
Um exemplo da definição de conversão de atributos é dado no mapeamento da classe Person:

Classe PERSON

```
…
   <attribute>
      <attributeName>name</attributeName>
        <columnName>nome</columnName>
        <converter>
            <converterName>CaseConverter</converterName>
            <parameter>
                   <parameterName>case</parameterName>
                   <parameterValue>upper</parameterValue>
               </parameter>
        </converter>
   </attribute>
…
```

Os parâmetros para a conversão são definidos dentro da tag <converter>.

<converterName>	nome da classe de conversão. Deve implementar a interface IConverter.
<parameter>	Define os parâmetros que poderão ser usados como valores de propriedades na classe de conversão. Uma classe pode ter vários parâmetros associados. No caso do exemplo, o parâmetro 'case' define se a conversão será para maiúsculas ('upper') ou minúsculas ('lower').Opcional.
<parameterName> <parameterValue>	Define o nome e o valor do parâmetro a ser passado para a classe de conversão.

A listagem abaixo mostra a definição da classe CaseConverter.

```
class caseconverter implements IConverter
{
   private $case;

   function caseconverter()
   {
   }

   function init($properties)
   {
      $this->case = $properties['case'];
   }

   function convertFrom($object)
   {
       switch ($this->case)
       {
           case 'upper': $o = strtoupper((string)$object); break;
           case 'lower': $o = strtolower((string)$object); break;
       }
       return $o;
   }

   function convertTo($object)
   {
       return strtoupper((string)$object);
   }

   function convertColumn($object)
   {
       return $object;
   }
}
```

Os parâmetros passados no mapeamento são armazenados no array $properties, que é passado para o método init(). O método convertFrom() converte o valor que foi lido do banco para ser armazenado em um atributo. O método convertTo() converte o valor do atributo em um valor a ser armazenado no banco. O

método convertColumn é usado primariamente pelo Miolo para fazer conversões necessárias na construção do comando SQL (por exemplo, na formatação de campos do tipo Date).

Objetos persistentes

Uma vez que a classe Business passa a herdar da classe PersistentObject, todos os objetos de negócio são automaticamente persistentes, ou seja, podem usar os métodos de persistência. O acesso direto à camada DAO do Miolo continua válido, permitindo construir comandos SQL independente da camada de persistência. Os principais métodos de um objeto persistente são:

function isPersistent()

Retorna true/false dependendo se o objeto é persistente ou não. Um objeto é considerado persistente os valores de seus atributos foram obtidos do banco de dados através da camada de persistência. Os métodos de recuperação (retrieve) de objetos definem este valor como true. Isto é importante porque para persistir um objeto (com o método save()) é necessário saber se o objeto é "novo" ou se veio do banco de dados.

function isProxy()

- $value: true/false

Retorna true/false dependendo se o objeto é do tipo "proxy" ou não. Em um objeto do tipo proxy, apenas os atributos com proxy=true foram recuperados do banco de dados, ou seja, um objeto do tipo proxy não possui todos os seus atributos preenchidos.

function retrieve()

Preenche os atributos do objeto, acessando o banco de dados e recuperando o(s) registro(s) necessário (s) com base no atributo chave do objeto. O atributo chave do objeto deve ser preenchido antes da execução do método. As associações podem ser automaticamente recuperadas, dependendo da configuração feita no mapeamento.

function retrieveFromQuery($query)

Preenche os atributos do objeto, com os dados do primeiro registro do resultset da query $query. As associações são automaticamente recuperadas, dependendo da configuração feita no mapeamento.

function retrieveFromCriteria($criteria)

Preenche os atributos do objeto, com os dados do primeiro registro do resultset da query executada através de $criteria. As associações são automaticamente recuperadas, dependendo da configuração feita no mapeamento.

function retrieveAssociation($target)

- $target: string com o nome da associação a ser recuperada

Recupera os dados referentes a uma associação e preenche o atributo correspondente com o objeto (ou com um array de objetos). Geralmente usado quando a recuperação da associação não é automática.

function retrieveAsProxy()

Preenche os atributos do objeto, acessando o banco de dados e recuperando o(s) registro(s) necessário (s) com base no atributo chave do objeto. O atributo chave do objeto deve ser preenchido antes da execução do método. Somente são preenchidos os atributos que tenham a opção proxy=true na sua definição.

function getCriteria()

Retorna um objeto do tipo RetrieveCriteria (vide Queries), usado para executar consultas customizadas no banco de dados.

function save()

Armazena (persiste) um objeto no banco de dados. O armazenamento pode envolver uma ou mais inclusões ou atualizações, em uma ou mais tabelas do banco de dados. As associações podem ser automaticamente armazenadas, dependendo da configuração feita no mapeamento.

function saveAssociation($target)

- $target: string com o nome da associação a ser persistida

Persiste os dados referentes a uma associação, atualizando as tabelas e objetos envolvidos. Geralmente usado quando a persistência da associação não é automática.

function delete()

Remove um objeto no banco de dados. A remoção pode envolver uma ou mais exclusões ou atualizações, em uma ou mais tabelas do banco de dados. As associações podem ser automaticamente removidas, dependendo da configuração feita no mapeamento.

function deleteAssociation($target, $object)

- $target: string com o nome da associação a ser removida
- $object: objeto associado

Remove a associação do objeto com outro objeto. Isto implica na remoção da chave estrangeira nas relações 1:1 ou 1:N, ou do registro de ligação nas relações N:N.

Cursor

Um cursor é um array de objetos. Pode ser usado como resultado de uma consulta ao banco de dados. O meto disponível para percorrer o curso é getObject(), que obtém o objeto corrente e avança para o próximo, retornando NULL se o final da lista foi alcançado.

Exemplo

```
$cursor = $this->user->ProxyCursorByLogin('2146'); // retorna um cursor, depois de uma
                                                   // consulta ao banco
while ($obj = $cursor->getObject())
{
   $text .= "<br>Iduser: " . $obj->iduser;
   $text .= "<br>Login: " . $obj->login . "<br>";
}
echo $text;
```

Queries

As consultas ao banco de dados (queries) são realizadas através de um mecanismo chamado “query by criteria”. Através deste mecanismo, definimos um “criteria” (um critério para a consulta), instanciando um objeto do tipo RetrieveCriteria. Um objeto RetrieveCriteria está associado a uma classe persistente que serve de base para a consulta (as consultas sempre são feitas sob a perspectiva de uma determinada classe persistente), sendo obtido através do método getCriteria().

Métodos de recuperação

Os seguinte métodos podem ser executados em um objeto RetrieveCriteria, para recuperação dos dados do banco de dados:

function retrieveAsQuery($parameters=null)

Retorna um objeto Query como resultado da consulta.

function retrieveAsCursor($parameters=null)

Retorna um objeto Cursor (um array de objetos) como resultado da consulta.

function retrieveAsProxyQuery($parameters=null)

Retorna um objeto Query como resultado da consulta. Somente são retornados os campos definidos com proxy=true no mapeamento da classe.

function retrieveAsProxyCursor($parameters=null)

Retorna um objeto Cursor como resultado da consulta. Somente são retornados os campos definidos com proxy=true no mapeamento da classe.

Métodos para definição do critério

Para melhor compreensão do processo de definição de um critério, os seguintes conceitos devem ser compreendidos:

- classe base: a classe usada como base para a consulta
- attributo: é um atributo/propriedade de uma classe persistente. Pode ser definido simplesmente através do nome do atributo, ou através do caminho (path) do atributo através das associações da classe
- operando: é um valor usado em um critério. Pode ser um atributo, uma constante, uma função ou um subcriteria (um objeto RetrieveCriteria)
- operador: define qual a operação usada no critério. As seguintes operações estão definidas: =, <>, >, <, >=, <=, LIKE, IN
- associação: é o nome de uma associação definida para a classe base do critério. O nome da associação é definido no atributo <target> do mapeamento da associação.
- Alias: nome usado como sinônimo de uma tabela do esquema relacional.

Os seguinte métodos podem ser executados em um objeto RetrieveCriteria, para definição dos critérios a serem usados na consulta ao banco de dados:

function addColumnAttribute($attribute, $alias='')

Acrescenta um campo a ser recuperado na consulta. Se não for definido nenhum campo desta forma, todos os atributos da classe base serão retornados. O parâmetro $alias permite definir um alias para a coluna, no comando SQL a ser gerado.

function addCriteria($op1, $operator, $op2)

Acrescenta uma operação a ser executada no critério. $op1 e $op2 são operandos, e $operator define a operação. Se outras operações já tiverem sido definidas, é usado o operador AND.

function addGroupAttribute($attribute)

Define o atributo a ser usado na cláusula GROUP BY.

function addHavingCriteria($op1, $operator, $op2)

Define a operação a ser usada na cláusula HAVING. Se outras operações já tiverem sido definidas, é usado o operador AND.

function addJoinCriteria($criteria)

Define uma operação de JOIN entre duas classes. A operação de join é realizada definindo-se dois critérios e associando um com o outro através deste método (vide Exemplos).

function addOrCriteria($op1, $operator, $op2)

Acrescenta uma operação a ser executada no critério. $op1 e $op2 são operandos, e $operator define a operação. Se outras operações já tiverem sido definidas, é usado o operador OR.

function addOrderAttribute($attribute, $ascend=true)

Define o atributo a ser usado na cláusula ORDER BY, e a direção da ordenação (ascendente ou descendente).

function addOrHavingCriteria($op1, $operator, $op2)

Define a operação a ser usada na cláusula HAVING. Se outras operações já tiverem sido definidas, é usado o operador OR.

function getCriteria($op1, $operator, $op2)

Retorna um objeto BaseCriteria, que pode ser anexado a um objeto CriteriaCondition, para criação de filtros compostos (vide Exemplos).

function setAlias($alias, $classMap=NULL)

Define um alias para a classe definida por $classMap. Se $classMap=NULL, o alias é definido para a classe base.

function setAssociationAlias($associationName,$alias)

Define um alias para a associação $associationName. Este alias pode ser usado para referenciar atributos de outras classes.

function setAssociationType($associationName,$joinType)

Define o tipo de join a ser feito em relação à associação. $joinType pode ser 'inner','left','right'. O default, quando este método não é usado, é fazer um INNER JOIN com a associação.

function setAutoAssociationAlias($alias0, $alias1)

Define os aliases a serem usados quando é necessário fazer um auto-relacionamento da classe.

function setDistinct($distinct=false)

Define a flag DISTINCT da query.

function setReferenceAlias($alias)

Define o alias a ser usado para que atributos em subqueries possam referenciar uma classe usada na query externa (vide Exemplos).

Exemplos

Para efeito dos exemplos, $this->group, $this->user, $this->sector, $this->person, $this->student são objetos das respectivas classes.

Consulta simples

Código:

```
$criteria = $this->group->getCriteria();
$query = $criteria->retrieveAsQuery();
```

SQL:
```
SELECT cm_grupoacesso.idgrupo,cm_grupoacesso.grupo FROM cm_grupoacesso
```

Consulta com filtro
Código:
```
        $criteria = $this->group->getCriteria();
        $criteria->addCriteria('group','LIKE',"'SIGA%'");
        $query = $criteria->retrieveAsQuery();
```
SQL:
```
SELECT cm_grupoacesso.idgrupo,cm_grupoacesso.grupo FROM cm_grupoacesso WHERE (cm_grupoacesso.grupo
LIKE 'SIGA%')
```

Consulta com filtro composto
A cláusula WHERE é criada a partir de um objeto da classe CriteriaCondition, que normalmente é encapsulado pela camada de persistência. No caso de criação de filtros compostos, pode ser necessário instanciar um objeto CriteriaCondition separado e anexa-lo ao critério da consulta.
Código:
```
        $criteria = $this->group->getCriteria();
        $cc = new CriteriaCondition;
        $cc->addCriteria($criteria->getCriteria('group','LIKE',"'%A%'"));
        $cc->addOrCriteria($criteria->getCriteria('group','LIKE',"'%E%'"));
        $criteria->addCriteria('group','LIKE',"'C%'");
        $criteria->addCriteria($cc);
        $query = $criteria->retrieveAsQuery();
```
SQL:
```
SELECT cm_grupoacesso.idgrupo,cm_grupoacesso.grupo FROM cm_grupoacesso WHERE ((cm_grupoacesso.grupo
LIKE 'C%') AND (((cm_grupoacesso.grupo LIKE '%A%') OR (cm_grupoacesso.grupo LIKE '%E%')) ))
```

Consulta com parâmetro
Código:
```
        $criteria = $this->group->getCriteria();
        $criteria->addCriteria('group','LIKE',"?");
        $criteria->addOrCriteria('group','LIKE',"'C%'");
        $query = $criteria->retrieveAsQuery("A%");
```
SQL:
```
SELECT cm_grupoacesso.idgrupo,cm_grupoacesso.grupo FROM cm_grupoacesso WHERE ((cm_grupoacesso.grupo
LIKE 'A%') OR (cm_grupoacesso.grupo LIKE 'C%'))
```

Consulta com ordenação
Código:
```
        $criteria = $this->group->getCriteria();
        $criteria->addOrderAttribute('group');
        $query = $criteria->retrieveAsQuery();
```
SQL:
```
SELECT cm_grupoacesso.idgrupo,cm_grupoacesso.grupo FROM cm_grupoacesso ORDER BY
cm_grupoacesso.grupo ASC
```

Consulta com join, via associação oneToOne, sem definição de colunas
Código:
```
        $criteria = $this->user->getCriteria();
        $criteria->addCriteria('sector.nick','LIKE',"'PROR%'");
        $query = $criteria->retrieveAsProxyQuery();
```
SQL:
```
SELECT cm_usuario.idusuario,cm_usuario.login,cm_usuario.idpessoa,cm_usuario.idsetor FROM
cm_usuario,cm_setor WHERE (cm_setor.siglasetor LIKE 'PROR%') and
(cm_usuario.idsetor=cm_setor.idsetor)
```

Consulta com join, via associação oneToOne, com definição de colunas
Código:
```
        $criteria = $this->user->getCriteria();
        $criteria->addCriteria('sector.nick','LIKE',"'PROR%'");
        $criteria->addColumnAttribute('login');
        $criteria->addColumnAttribute('sector.nick');
        $query = $criteria->retrieveAsProxyQuery();
```
SQL:
```
SELECT cm_usuario.login,cm_setor.siglasetor FROM cm_usuario,cm_setor WHERE (cm_setor.siglasetor
LIKE 'PROR%') and (cm_usuario.idsetor=cm_setor.idsetor)
```

Consulta com join, via associação manyToMany, sem definição de colunas
Código:
```
        $criteria = $this->user->getCriteria();
        $criteria->addCriteria('groups.group','LIKE',"'CDARA%'");
```

```
        $query = $criteria->retrieveAsProxyQuery();
```

SQL:

```
SELECT cm_usuario.idusuario,cm_usuario.login,cm_usuario.idpessoa,cm_usuario.idsetor FROM
cm_grpusuario,cm_usuario,cm_grupoacesso WHERE (cm_grupoacesso.grupo LIKE 'CDARA%') and
(cm_grpusuario.idusuario=cm_usuario.idusuario) and (cm_grpusuario.idgrupo=cm_grupoacesso.idgrupo)
```

Consulta com join, via associação manyToMany, com definição de colunas e agrupamento

Código:

```
        $criteria = $this->group->getCriteria();
        $criteria->addColumnAttribute('group');
        $criteria->addColumnAttribute('count(users.iduser)');
        $criteria->addGroupAttribute('group');
        $query = $criteria->retrieveAsQuery();
```

SQL:

```
SELECT cm_grupoacesso.grupo,count(cm_usuario.idusuario) FROM
cm_grpusuario,cm_grupoacesso,cm_usuario WHERE (cm_grpusuario.idgrupo=cm_grupoacesso.idgrupo) and
(cm_grpusuario.idusuario=cm_usuario.idusuario) GROUP BY cm_grupoacesso.grupo
```

Consulta com join, via associação manyToMany, com definição de colunas, agrupamento e cláusula having

Código:

```
        $criteria = $this->group->getCriteria();
        $criteria->addColumnAttribute('group');
        $criteria->addColumnAttribute('count(users.iduser)');
        $criteria->addGroupAttribute('group');
        $criteria->addHavingCriteria('count(users.iduser)','>','50');
        $query = $criteria->retrieveAsQuery();
```

SQL:

```
SELECT cm_grupoacesso.grupo,count(cm_usuario.idusuario) FROM
cm_grpusuario,cm_grupoacesso,cm_usuario WHERE (cm_grpusuario.idgrupo=cm_grupoacesso.idgrupo) and
(cm_grpusuario.idusuario=cm_usuario.idusuario) GROUP BY cm_grupoacesso.grupo HAVING (count
(cm_usuario.idusuario) > 50)
```

Consulta com uso do operador de conjuntos IN

Código:

```
        $criteria = $this->group->getCriteria();
        $values = array('CDARA','CURRICULO','EXCECAO');
        $criteria->addCriteria('group','IN', $values);
        $query = $criteria->retrieveAsQuery();
```

SQL:

```
SELECT cm_grupoacesso.idgrupo,cm_grupoacesso.grupo FROM cm_grupoacesso WHERE (cm_grupoacesso.grupo
IN ('CDARA', 'CURRICULO', 'EXCECAO'))
```

Consulta com uso de alias

Código:

```
        $criteria = $this->group->getCriteria();
        $criteria->setAlias('G');
        $criteria->addColumnAttribute('G.idgroup');
        $criteria->addColumnAttribute('G.group');
        $criteria->addCriteria('group','LIKE', "'A%'");
        $query = $criteria->retrieveAsQuery();
```

SQL:

```
SELECT G.idgrupo,G.grupo FROM cm_grupoacesso G WHERE (G.grupo LIKE 'A%')
```

Consulta com uso de alias para associação

Código:

```
        $criteria = $this->group->getCriteria();
        $criteria->setAlias('G');
        $criteria->setAssociationAlias('users','U');
        $criteria->addColumnAttribute('G.idgroup');
        $criteria->addColumnAttribute('G.group');
        $criteria->addColumnAttribute('U.iduser');
        $criteria->addCriteria('group','=', "'ADMIN'");
        $query = $criteria->retrieveAsQuery();
```

SQL:

```
SELECT G.idgrupo,G.grupo,U.idusuario FROM cm_grpusuario,cm_grupoacesso G,cm_usuario U WHERE
(G.grupo = 'ADMIN') and (cm_grpusuario.idgrupo=G.idgrupo) and (cm_grpusuario.idusuario=U.idusuario)
```

Consulta com auto- relacionamento

Código:

```
        $criteria = $this->sector->getCriteria();
        $criteria->setAutoAssociationAlias('S1','S2');
        $criteria->addColumnAttribute('S1.nick');
        $criteria->addColumnAttribute('S2.nick');
        $criteria->addCriteria('S1.nick', '=', 'S2.parent');
        $criteria->addCriteria('S1.parent','=', "'PROGRAD'");
        $query = $criteria->retrieveAsQuery();
```

SQL:

```
SELECT S1.siglasetor,S2.siglasetor FROM cm_setor S1,cm_setor S2 WHERE ((S1.siglasetor =
S2.paisetor) AND (S1.paisetor = 'PROGRAD'))
```

Consulta com subquery

Código:

```
        $subCriteria = $this->user->getCriteria();
        $subCriteria->addCriteria('sector.nick','LIKE',"'PROR%'");
        $subCriteria->addColumnAttribute('iduser');
        $criteria = $this->user->getCriteria();
        $criteria->addCriteria('iduser','IN', $subCriteria);
        $query = $criteria->retrieveAsQuery();
```

SQL:

```
SELECT
cm_usuario.idusuario,cm_usuario.login,cm_usuario.password,cm_usuario.nick,cm_usuario.idpessoa,
cm_usuario.idsetor FROM cm_usuario WHERE (cm_usuario.idusuario IN (SELECT cm_usuario.idusuario FROM
cm_usuario,cm_setor WHERE (cm_setor.siglasetor LIKE 'PROR%') and
(cm_usuario.idsetor=cm_setor.idsetor)))
```

Consulta com subquery referenciando a query externa

Código:

```
        $subCriteria = $this->user->getCriteria();
        $subCriteria->setReferenceAlias('S');
        $subCriteria->addColumnAttribute('count(iduser)');
        $subCriteria->addCriteria('idsector','=',"S.idsector");
        $criteria = $this->sector->getCriteria();
        $criteria->setAlias('S');
        $criteria->addCriteria($subCriteria, '>', '150');
        $query = $criteria->retrieveAsQuery();
```

SQL:

```
      SELECT S.idsetor,S.siglasetor,S.nomesetor,S.paisetor,S.tiposetor FROM cm_setor S WHERE
((SELECT count(cm_usuario.idusuario) FROM cm_usuario WHERE (cm_usuario.idsetor = S.idsetor)) > 150)
```

Consulta utilizando “path join”

Código:

```
        $criteria = $this->group->getCriteria();
        $criteria->addColumnAttribute('group');
        $criteria->addColumnAttribute('users.login');
        $criteria->addColumnAttribute('users.sector.nick');
        $criteria->addCriteria('users.sector.nick','=',"'REITORIA'");
        $query = $criteria->retrieveAsQuery();
```

SQL:

```
SELECT cm_grupoacesso.grupo,cm_usuario.login,cm_setor.siglasetor FROM
cm_grpusuario,cm_grupoacesso,cm_usuario,cm_setor WHERE (cm_setor.siglasetor = 'REITORIA') and
(cm_grpusuario.idgrupo=cm_grupoacesso.idgrupo) and (cm_grpusuario.idusuario=cm_usuario.idusuario)
and (cm_usuario.idsetor=cm_setor.idsetor)
```

Consulta com outer join

Código:

```
        $criteria = $this->sector->getCriteria();
        $criteria->addColumnAttribute('nick');
        $criteria->addColumnAttribute('count(users.iduser)');
        $criteria->addCriteria('nick','LIKE',"'PRO%'");
        $criteria->addGroupAttribute('nick');
        $criteria->addHavingCriteria('count(users.iduser)','=','0');
        $criteria->setAssociationType('users','right');
        $query = $criteria->retrieveAsQuery();
```

SQL:

```
SELECT cm_setor.siglasetor,count(cm_usuario.idusuario) FROM cm_setor,cm_usuario WHERE
(cm_setor.siglasetor LIKE 'PRO%') and (cm_setor.idsetor = cm_usuario.idsetor(+)) GROUP BY
cm_setor.siglasetor HAVING (count(cm_usuario.idusuario) = 0)
```

Consulta com herança entre classes

Código:

```
        $criteria = $this->student->getCriteria();
        $criteria->addColumnAttribute('number');
        $criteria->addColumnAttribute('name');
        $criteria->addCriteria('name','LIKE',"'FELIPE A%'");
        $query = $criteria->retrieveAsQuery();
```

SQL:

```
SELECT ga_aluno.matricula,cm_pessoa.nome FROM ga_aluno,cm_pessoa WHERE (cm_pessoa.nome LIKE 'FELIPE A%') and (ga_aluno.idpessoa=cm_pessoa.idpessoa) and (ga_aluno.idpessoa=cm_pessoa.idpessoa)
```

Referências

- Scott W. Ambler, The fundamental of Mapping Objects to Relational Databases. Disponível em: http://www.agiledata.org/essays/mappingObjects.html
- Scott W. Ambler, The design of a Robust Persistence Layer for Relational Databases. Disponível em: http://www.ambysoft.com/persistenceLayer.pdf
- Artyom Rudoy, Persistence Layer. Disponível em: http://artyomr.narod.ru

ChangeLog

2.0a:

- Alteração na descrição de function retrieveAssociation($target)
- Inclusão da descrição de <indexAttribute> nas associações
- Inclusão da descrição de <orderAttribute> nas associações
- Inclusão da descrição de <attributeIndex> nos atributos

2.0beta4

- Inclusão da descrição de <extends> no mapeamento de classe
- Inclusão do método retrieveFromQuery nos objetos persistentes
- Inclusão do método retrieveFromCriteria nos objetos persistentes
- Inclusão do método convertColumn na interface Iconverter
- Inclusão do parâmetro $alias no método addColumnAttribute das queries
- Inclusão do método deleteAssociation nos objetos persistentes
- Inclusão do método saveAssociation nos objetos persistentes