2025/11/11 03:44 1/10 Modèles

Modèles

Un modèle est une classe métier, représentant une partie des données d'une application. Dans la plupart des cas, un modèle est associé à une table de la base de données.

Phalcon\Mvc\Model est la classe de base des models d'une application. Cette classe met à disposition des fonctionnalités CRUD, offre des possibilités de recherche avancées, et permet de gérer les relations entre models, le tout sans avoir besoin d'utiliser SQL.

-- Création de models

```
<?php
class Utilisateur extends \Phalcon\Mvc\Model
{
}</pre>
```

```
<?php
class Utilisateur extends \Phalcon\Mvc\Model{
    /**
    * @var string
    */
    protected $prenom;

    /**
    * @var string
    */
    protected $nom;

/**
    * Method to set the value of field prenom
    *
    * @param string $prenom
    * @return $this
    */
    public function setPrenom($prenom)
    {
        $this->prenom = $prenom;
        return $this;
    }
}
```

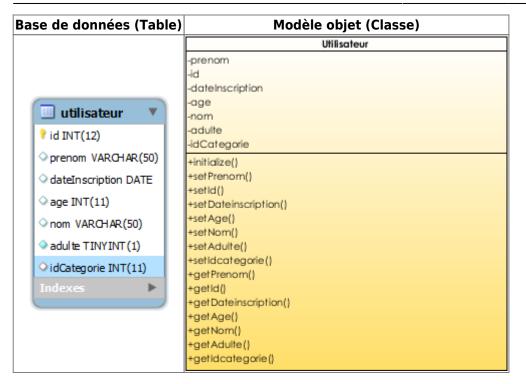
```
/**
     * Method to set the value of field nom
     * @param string $nom
     * @return $this
    public function setNom($nom)
        $this->nom = $nom;
        return $this;
    }
     * Returns the value of field prenom
     * @return string
    public function getPrenom()
        return $this->prenom;
    }
     * Returns the value of field nom
     * @return string
    public function getNom()
        return $this->nom;
    }
}
```

-- Mappage Objet <=> Relationnel

Par défaut, Phalcon effectue un mappage entre classes et tables de la base de données de la façon suivante :

- Table ⇔ Classe du même nom
- Enregistrement ⇔ instance de classe (objet métier)
- Colonne (champ) ⇔ membre de données du même nom

2025/11/11 03:44 3/10 Modèles



-- Mappage nom de table/classe

Si le nom de la table de la base de données ne correspond pas au nom de la classe, il est possible de surdéfinir la méthode **getSource** :

```
class Users extends \Phalcon\Mvc\Model{
    //Retourne le nom de la table correspondant à la classe
    public function getSource(){
        return "Utilisateur";
    }
}
```

-- Mappage des noms de champs/membres

De même, si les noms de champ de la table ne correspondent pas aux membres de données de la classe :

```
<?php

class Utilisateur extends \Phalcon\Mvc\Model
{
    protected $code;
    protected $name;
    public function columnMap()
    {
        //Les clés correspondent aux noms dans la table
}</pre>
```

```
//Les valeurs aux noms dans l'application
return array(
    'id' => 'code',
    'nom' => 'name'
);
}
```

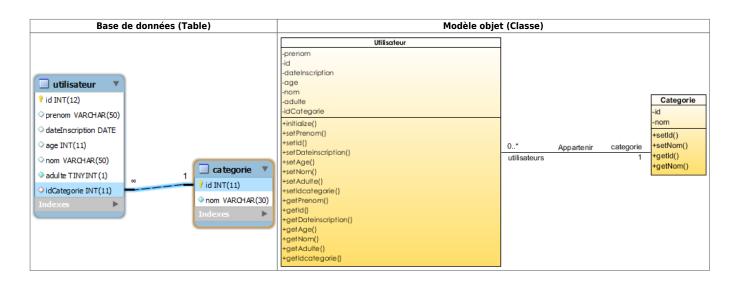
-- Relations

Avec Phalcon, les relations peuvent être définies grâce à la méthode **initialize()** du modèle. Les méthodes **belongsTo()**, **hasOne()**, **hasMany()** and **hasManyToMany()** definissent des relations entre 1 ou plusieurs membres du modèle courant et des membres d'un autre modèle. Chacune de ces méthodes requiert 3 paramètres : le membre local, le modèle cible, les membres cibles.

Méthode	Description
hasMany	Defines a 1-n relationship
hasOne	Defines a 1-1 relationship
belongsTo	Defines a n-1 relationship
hasManyToMany	Defines a n-n relationship

--belongsTo (relation n-1) & hasMany (relation 1-n)

Exemple:



-- belongsTo

Chaque utilisateur appartient à une catégorie :

```
class Utilisateur extends \Phalcon\Mvc\Model{
   ...
```

2025/11/11 03:44 5/10 Modèles

```
/**
    * @var integer
    */
protected $idCategorie;

public function initialize()
{
    $this->belongsTo("idCategorie", "Categorie", "id");
}
...
```

Les paramètres passés à la méthode belongsTo sont :

- 1. **idCategorie** : membre local intervenant dans l'association (clé étrangère)
- 2. Categorie : Classe référencée associée
- 3. id: membre référencé dans la classe associée

Création d'une action dans le contrôleur IndexController pour afficher un utilisateur et sa catégorie :

```
<?php

class IndexController extends \Phalcon\Mvc\Controller{
    ...
        public function showUserAction($id){
        $user=Utilisateur::findFirst($id);
        echo $user->getNom()." : ".$user->getCategorie()->getNom();
    }
}
```

Affichage de la réponse obtenue :



Phalcon charge l'utilisateur, et l'instance de catégorie correspondant, accessible grâce aux méthodes magiques _set et _get

-- hasMany

Chaque catégorie est associée à 1 ou plusieurs utilisateurs :

```
class Categorie extends \Phalcon\Mvc\Model{
    /**
    * @var integer
    */
    protected $id;

    /**
    * @var string
    */
    protected $nom;

public function initialize(){
        $this->hasMany("id", "Utilisateur",
    "idCategorie",array("alias"=>"utilisateurs"));
    }
    ...
```

Les paramètres passés à la méthode hasMany sont :

- 1. **id** : membre local intervenant dans l'association (clé primaire)
- 2. Utilisateur : Classe associée
- 3. **idCategorie** : membre associé
- 4. **utilisateurs** : alias du membre créé par l'association (collection d'Utilisateurs)

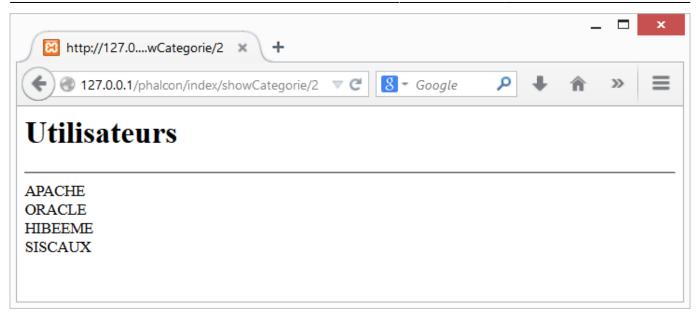
```
<?php

class IndexController extends \Phalcon\Mvc\Controller{
    ...
    public function showCategorieAction($id){
        $categorie=Categorie::findFirst($id);
        echo "<hl>".$categorie->getNom()."</hl>";
        echo "<hr>";
        foreach ($categorie->getUtilisateurs() as $user){
            echo($user->getNom()."<br>");
        }
    }
}
```

On obtient une réponse par l'intermédiaire du getter **getUtilisateurs()**, en référence à l'alias **utilisateurs**, sans qu'il ait été implémenté par nos soins, en passant par les méthodes magiques php **get** et **set**.

Affichage de la réponse obtenue :

2025/11/11 03:44 7/10 Modèles

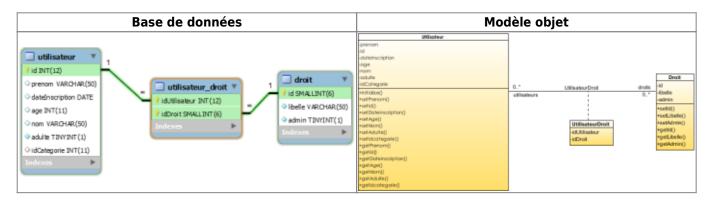


Le getter **getUtilisateurs()** peut également être utilisé pour filtrer les utilisateurs de la catégorie affichée :

```
<?php

class IndexController extends \Phalcon\Mvc\Controller{
    ...
    public function showCategorieAction($id){
        $categorie=Categorie::findFirst($id);
        echo "<h1>".$categorie->getNom()."</h1>";
        echo "<hr>";
        //Affichage des utilisateurs dont le nom contient CA
        foreach ($categorie->getUtilisateurs("nom like '%CA%'") as $user){
             echo($user->getNom()."<br>");
        }
    }
}
```

-- hasManyToMany (relation n-n)



Les utilisateurs disposent de droits :

```
<?php
class Utilisateur extends \Phalcon\Mvc\Model{</pre>
```

Création d'une action dans le contrôleur IndexController pour afficher un utilisateur et ses droits :

```
<?php

class IndexController extends \Phalcon\Mvc\Controller{
    ...
    public function showUserDroitsAction($id){
        $user=Utilisateur::findFirst($id);
        echo "<hl>".$user->getNom()." : ".$user->getCategorie()->getNom()."</hl>";
        echo "";
        foreach ($user->getDroits() as $droit){
            echo """:, $droit->getLibelle()."";
        }
        echo "";
    }
    ...
}
```



-- Opérations CRUD

-- Lecture/recherche

-- find() et findFirst()

Lister tous les enregistrements d'une table :

2025/11/11 03:44 9/10 Modèles

```
$utilisateurs = Utilisateur::find();
foreach($utilisateurs as $utilisateur){
    echo $utilisateur->getNom()."<br>;
}
echo "Nombre d'utilisateurs : ", count($utilisateurs), "\n";
```

Poser une condition:

```
// Comptage du nombre d'adultes
$utilisateurs = Utilisateur::find("adulte = 1");
echo "Adultes ", count($utilisateurs), "\n";
```

Trier:

```
$utilisateurs = Utilisateur::find(array(
    "adulte = 1",
    "order" => "nom"
));
foreach ($utilisateurs as $utilisateur) {
    echo $utilisateur->nom, "\n";
}
```

Tri et limite:

```
$utilisateurs = Robots::find(array(
    "adulte = 1",
    "order" => "nom",
    "limit" => 5
));
foreach ($utilisateurs as $utilisateur) {
    echo $utilisateur->nom, "\n";
}
```

L'utilisation de la méthode **findFirst()** permet d'obtenir le premier enregistrement répondant au(x) critère(s) :

```
// Premier utilisateur ?
$utilisateur = Utilisateur::findFirst();
echo "Le nom du premier utilisateur est ", $utilisateur->getNom(), "\n";
```

Premier utilisateur répondant à un critère :

```
$utilisateur = Utilisateur::findFirst("adulte = 1");
echo "Le premier adulte est : ", $utilisateur->getNom(), "\n";
```

Premier utilisateur répondant à un critère, avec classement par nom :

```
$utilisateur = Utilisateur::findFirst(array("adulte = 1", "order" => "nom"));
echo "Le premier adulte est : ", $utilisateur->getNom(), "\n";
```

Les méthodes find() et findFirst() permettent de définir des critères à partir d'un tableau associatif :

```
$utilisateur = Utilisateur::findFirst(array(
    "adulte = 1",
    "order" => "nom DESC",
    "limit" => 30
));
```

```
$utilisateurs = Utilisateur::find(array(
    "conditions" => "adulte = ?1",
    "bind" => array(1 => 1)
));
```

Il est également possible de créer des requêtes de façon orientée objet :

```
$utilisateurs = Utilisateur::query()
   ->where("adulte = :value:")
   ->andWhere("nom like 'C%'")
   ->bind(array("value" => 1))
   ->order("nom")
   ->execute();
```

-- PHQL

PHQL : Phalcon Query Language, SQL orienté objet, offre également la possibilité d'interroger la base de données :

From:

http://slamwiki2.kobject.net/ - SlamWiki 2.1

Permanent link:

http://slamwiki2.kobject.net/slam4/php/phalcon/models?rev=1421419396

Last update: 2019/08/31 14:41

