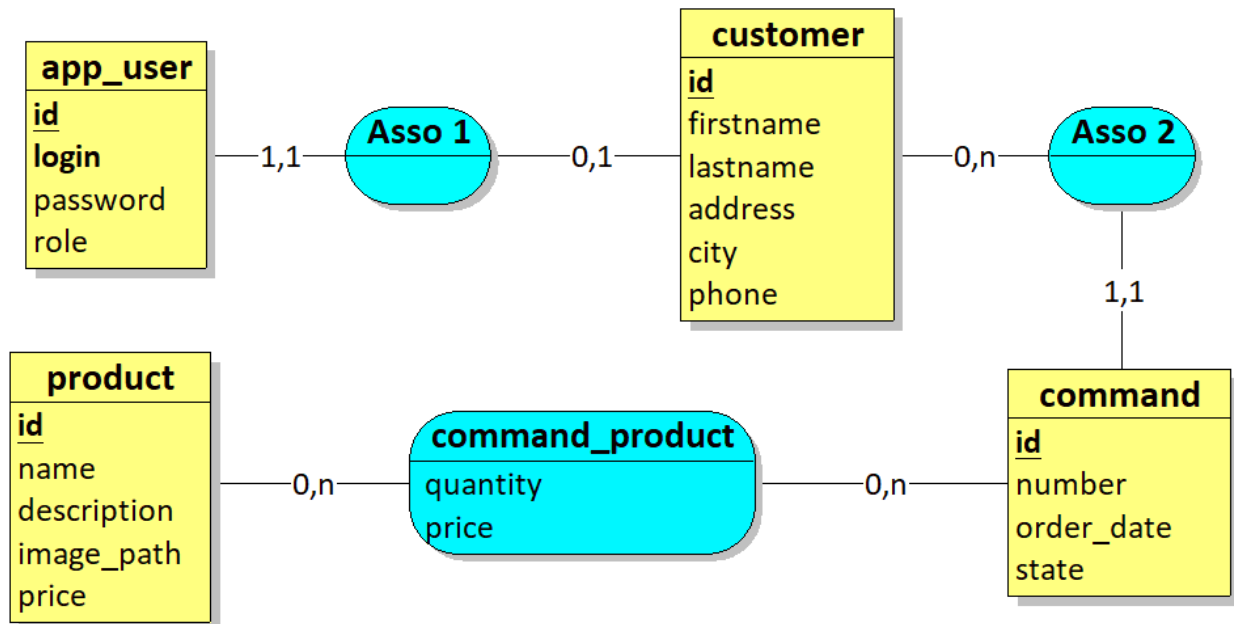


TD PHP MVC

Étape 19 : Passer des commandes ... (les lire dans un premier temps)

Avant de pouvoir ajouter un produit dans un panier nous devons créer 2 tables supplémentaires. Une table command et une table command_product faisant le lien entre une commande et les différents produits qui la composent.



Voici les scripts SQL :

```
DROP TABLE IF EXISTS `command`;
CREATE TABLE IF NOT EXISTS `command` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `number` varchar(255) COLLATE utf8mb4_general_ci DEFAULT NULL,
  `order_date` datetime DEFAULT NULL,
  `state` tinyint(4) DEFAULT NULL,
  `customer_id` int(11) NOT NULL,
  PRIMARY KEY (`id`),
  KEY `customer_id` (`customer_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
```

```
DROP TABLE IF EXISTS `command_product`;
CREATE TABLE IF NOT EXISTS `command_product` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `command_id` int(11) DEFAULT NULL,
  `product_id` int(11) DEFAULT NULL,
  `quantity` int(11) DEFAULT NULL,
  `price` float DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `product_id` (`product_id`),
  KEY `command_id` (`command_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
```

Ajoutons les contraintes :



```
ALTER TABLE `command` ADD CONSTRAINT `command_ibfk_1` FOREIGN KEY (`customer_id`) REFERENCES `customer` (`id`);
```

```
ALTER TABLE `command_product`  
ADD CONSTRAINT `command_product_ibfk_1` FOREIGN KEY (`product_id`) REFERENCES `product` (`id`),  
ADD CONSTRAINT `command_product_ibfk_2` FOREIGN KEY (`command_id`) REFERENCES `command` (`id`);
```

La relation « command_product » entre command et product est de type ManyToMany. Cette relation va devenir une table lors de la création de la base de données. Une ligne de cette table « command_product » fera référence à un produit commandé (avec sa quantité et le prix total) et appartiendra à une commande.

Il existe aussi une relation ManyToOne/OneToMany entre command et customer. Une commande ne pouvant être passée que par un seul client et, un client pouvant avoir réalisé plusieurs commandes.

Nous commençons par créer les classe entity correspondant aux tables que nous venons de créer et nous y ajoutons les relations.

```
entity >  command.entity.php >  Command  
1  <?php  
2  
3  class Command extends Model{  
4  
5      static $relations =  
6          [  
7              "Customer" => ['type'=>'hasOne',  
8                  'table'=>'customer',  
9                  'attribute'=>'customer',  
10                 'foreignKey'=>'customer_id'],  
11  
12             "Lines" => ['type'=>'hasMany',  
13                 'table'=>'command_product',  
14                 'attribute'=>'lines',  
15                 'foreignKey'=>'command_id']  
16         ];  
17  
18     }
```

```

entity > 🐘 command_product.entity.php > 🗑️ Command_product
1  <?php
2
3  class Command_product extends Model
4  {
5
6      static $relations =
7      [
8          "Command" => ['type'=>'hasOne',
9                        'table'=>'command',
10                       'attribute'=>'command',
11                       'foreignKey'=>'command_id'],
12
13          "Product" => ['type'=>'hasOne',
14                       'table'=>'product',
15                       'attribute'=>'product',
16                       'foreignKey'=>'product_id']
17      ];
18
19  }

```

Nous ajoutons également la relation entre command et customer dans l'entity customer

```

entity > 🐘 customer.entity.php > 🗑️ Customer
1  <?php
2
3  class Customer extends Model{
4
5      static $relations =
6      [
7          "User" => ['type'=>'hasOne',
8                    'table'=>'app_user',
9                    'attribute'=>'user',
10                   'foreignKey'=>'customer_id'],
11
12          "Commands" => ['type'=>'hasMany',
13                        'table'=>'command',
14                        'attribute'=>'commands',
15                        'foreignKey'=>'customer_id']
16      ];
17
18  }

```

Nous allons ajouter une commande et 3 lignes de commande (command_product) en base de données.

```
INSERT INTO `command` (`id`, `number`, `order_date`, `state`, `customer_id`) VALUES
(1, '210914-0001', NULL, NULL, 1);
```

```
INSERT INTO `command_product` (`id`, `command_id`, `product_id`, `quantity`, `price`) VALUES
(1, 1, 4, 1, 9.46),
(2, 1, 1, 3, 31.92),
(3, 1, 6, 2, 19.5);
```

Nous allons commencer par modifier le UserController pour afficher la liste des commandes d'un utilisateur.

Dans l'action login, plutôt que de stocker true en session quand l'authentification est réussie, nous allons stocker un objet qui nous permettra de connaître l'utilisateur authentifié et donc le customer correspondant.

```
controller > user.controller.php > UserController > login
57 public function login(){
58     $errors = [];
59     $posted = [];
60     if (isset($_SESSION['post'])){
61         $posted = $_SESSION['post'];
62         unset($_SESSION['post']);
63         $repository = new MainRepository('app_user');
64         $errors = $repository->validate($posted);
65         if (count($errors) == 0) {
66             $users = $repository->getAll("login = '". $posted['login'] ."'");
67             if(count($users) == 1){
68                 $user = array_pop($users);
69                 if(password_verify($posted['password'], self::$prefix . $user->password)){
70                     unset($user->login);
71                     unset($user->password);
72                     $repo = new MainRepository('customer');
73                     $customer = $repo->getOne($user->customer_id);
74                     $user->customer = $customer;
75                     $_SESSION['logged'] = serialize($user);
76                     header('Location: /home');
77                     die;
78                 }
79             }
80         }
81         $errors['bad'] = true;
82     }
83     $this->entities = [ 'errors' => $errors,
84                       'posted' => $posted ];
85     $this->render();
86 }
```

Pour stocker un objet (instance d'une classe) en session, nous devons le sérialiser (serialize L75), à l'inverse, lorsque nous souhaiterons récupérer la valeur de cet objet, nous devons le dé-sérialiser (unserialize L99 en page suivante).

Nous créons l'action commands dans UserController pour pouvoir afficher la liste des commandes de l'utilisateur authentifié. Si pas d'utilisateur authentifié, nous redirigeons vers la page de login.

```
controller > user.controller.php > UserController > commands
94 public function commands(){
95     if(!isset($_SESSION['logged'])){
96         header('Location: /user/login');
97         die;
98     }
99     $logged = unserialize($_SESSION['logged']);
100     $customer_id = $logged->customer_id;
101     $repository = new MainRepository('command');
102     $repository->with('Lines', ['Product']);
103     $commands = $repository->getAll("customer_id = " . $customer_id);
104
105     $this->entities = ['commands' => $commands];
106
107     $this->render();
108 }
109 }
```

L102, nous avons ajouté un 2ème argument à la fonction with. Ce tableau va contenir les relations pour lesquelles nous souhaitons récupérer des données pour le premier argument passé. Ici, nous voulons récupérer les commandes correspondant au client actuellement authentifié. Pour ces commandes nous souhaitons récupérer leur lignes (Lines) mais également le produit (Product) correspondant à ces lignes.

Ainsi dans notre template, nous pourrons passer directement d'un objet command à ses lignes (command_product) et d'un objet line (command_product) à son produit (product) pour récupérer par exemple le nom du produit.

```
template > user > commands.view.php
1 <?php foreach($commands as $command){
2     $totalCommand = 0;
3 }
4 <u><b>Commande N° <?= $command->number ?></b></u><br>
5 <?php foreach($command->lines as $line){
6     $totalCommand += $line->price;
7 }
8 <?= $line->product->name ?> x <?= $line->quantity ?> : <?= $line->price ?> € <br>
9 <?php } ?>
10 <u>Total Commande :</u> <b><?= $totalCommand ?>€</b>
11 <hr>
12 <?php } ?>
```

Avant de tester, il nous reste à modifier la fonction with de MainRepository pour qu'elle prenne en compte le 2ème niveau de relations.

```

repository > main.repository.php > MainRepository > with
238 function with($name, $withArray = [])
239 {
240     $relationToAdd = $this->entity::$relations[$name];
241     $relationToAdd['name'] = $name;
242     array_push($this->relations, $relationToAdd);
243     $this->subRelations[$name] = $withArray;
244     return $this;
245 }

```

Ne pas oublier d'initialiser subRelations dans le constructeur

```

repository > main.repository.php > MainRepository > __construct
1 <?php
2
3 class MainRepository
4 {
5     function __construct($table)
6     {
7         $this->table = $table;
8         $this->entity = ucfirst($this->table);
9         $this->db = null;
10
11         $this->relations = [];
12         $this->subRelations = [];
13     }

```

Il nous reste à récupérer les données pour ces « sous-relations » dans les méthodes getAll et getOne.

```

repository > main.repository.php > MainRepository > getAll
44 function getAll($where = "1")
45 {
46     $sql = "SELECT * FROM $this->table WHERE $where";
47     $resp = $this->connect()->query($sql);
48     $rows = $resp->fetchAll(PDO::FETCH_CLASS | PDO::FETCH_PROPS_LATE, $this->entity);
49     if (count($rows) == 0) {
50         return $rows;
51     }
52     foreach ($this->relations as $relation) {
53         if ($relation['type'] == 'hasMany') {
54             $repo = new MainRepository($relation['table']);
55             foreach ($this->subRelations[$relation['name']] as $subRel) {
56                 $repo->with($subRel);
57             }
58             $results = $repo->getAll();

```

repository > main.repository.php > MainRepository > getAll

```
66     if ($relation['type'] == 'hasOne') {
67         $repo = new MainRepository($relation['table']);
68         foreach ($this->subRelations[$relation['name']] as $subRel) {
69             $repo->with($subRel);
70         }
71         $results = $repo->getAll();
```

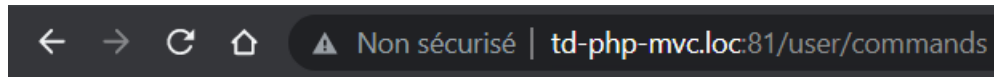
repository > main.repository.php > MainRepository > getAll

```
80     if ($relation['type'] == 'isOne') {
81         $repo = new MainRepository($relation['table']);
82         foreach ($this->subRelations[$relation['name']] as $subRel) {
83             $repo->with($subRel);
84         }
85         $results = $repo->getAll();
```

repository > main.repository.php > MainRepository > getOne

```
110
111     foreach ($this->relations as $relation) {
112         if ($relation['type'] == 'hasMany') {
113             $repo = new MainRepository($relation['table']);
114             foreach ($this->subRelations[$relation['name']] as $subRel) {
115                 $repo->with($subRel);
116             }
117             $results = $repo->getAll($relation['foreignKey']." = $row->id");
118             $row->{$relation['attribute']} = $results;
119         }
120         if ($relation['type'] == 'hasOne') {
121             $repo = new MainRepository($relation['table']);
122             foreach ($this->subRelations[$relation['name']] as $subRel) {
123                 $repo->with($subRel);
124             }
125             $result = $repo->getOne($row->{$relation['foreignKey']});
126             $row->{$relation['attribute']} = $result;
127         }
128         if ($relation['type'] == 'isOne') {
129             $repo = new MainRepository($relation['table']);
130             foreach ($this->subRelations[$relation['name']] as $subRel) {
131                 $repo->with($subRel);
132             }
133             $results = $repo->getAll($relation['foreignKey']." = $row->id");
134             $row->{$relation['attribute']} =
135                 count($results) == 1 ? array_shift($results) : null;
136         }
137     }
138
139     return $row;
140 }
```

Résultat en étant authentifié pour le user « admin »



[Logout](#)

Commande N° 210914-0001

Napolitaine x 1 : 9.46 €

4 Fromages x 3 : 31.92 €

Orientale x 2 : 19.5 €

Total Commande : 60.88€

FOOTER