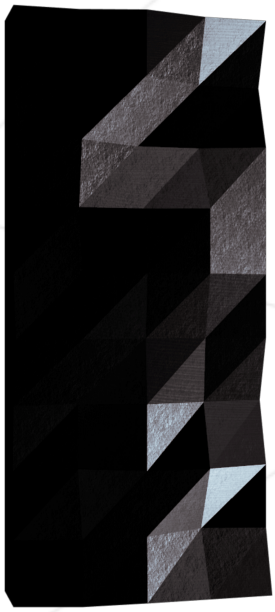

Formation PHP - Symfony

D05 - SQL vs ORM

Andrei Husar andrei.husar@pitechplus.com
42 Staff pedago@staff.42.fr

Résumé: Dans cette journée de la formation vous allez apprendre à utiliser SQL et ORM (Object-Relational Mapping) dans le cadre du framework Symfony.

Table des matières

I	Préambule	2
II	Consignes	3
III	Day-specific rules	4
IV	Exercice 00	5
V	Exercice 01	7
VI	Exercice 02	9
VII	Exercice 03	10
VIII	Exercice 04	12
IX	Exercice 05	13
X	Exercice 06	14
XI	Exercice 07	16
XII	Exercice 08	17
XIII	Exercice 09	19
XIV	Exercice 10	20
XV	Exercice 11	21
XVI	Exercice 12	23
XVII	Exercice 13	24
XVIII	Exercice 14	26

Chapitre I

Préambule

Une des tâches les plus communes pour toute application concerne le stockage persistent en base de données et la lecture/écriture à partir d'une base de données. Même si le framework Symfony n'intègre pas de composant natif pour interagir avec les bases de données, il fournit une intégration très poussée avec une librairie tierce appelée Doctrine. L'objectif principal de cette dernière est d'offrir de puissants outils permettant de rendre les interactions avec les bases de données aisées et flexibles.

Chapitre II

Consignes

Sauf contradiction explicite, les consignes suivantes seront valables pour tous les jours de cette Piscine.

- Seul ce sujet sert de référence : ne vous fiez pas aux bruits de couloir.
- Le sujet peut changer jusqu'à une heure avant le rendu.
- Les exercices sont très précisément ordonnés du plus simple au plus complexe. En aucun cas nous ne porterons attention ni ne prendrons en compte un exercice complexe si un exercice plus simple n'est pas parfaitement réussi.
- Attention aux droits de vos fichiers et de vos répertoires.
- Vous devez suivre la procédure de rendu pour tous vos exercices. L'url de votre dépot `GIT` pour cette journée est disponible sur votre intranet.
- Vos exercices seront évalués par vos camarades de Piscine.
- En plus de vos camarades, vous pouvez être évalués par un programme appelé la Moulinette. La Moulinette est très stricte dans sa notation car elle est totalement automatisée. Il est donc impossible de discuter de sa note avec elle. Soyez d'une rigueur irréprochable pour éviter les mauvaises surprises.
- Les exercices shell doivent s'éxcuter avec `/bin/sh`.
- Vous ne devez laisser aucun autre fichier que ceux explicitement specifiés par les énoncés des exercices dans votre dépot de rendu.
- Vous avez une question ? Demandez à votre voisin de droite. Sinon, essayez avec votre voisin de gauche.
- Toutes les réponses à vos questions techniques se trouvent dans les `man` ou sur Internet.
- Pensez à discuter sur le forum Piscine de votre Intra et sur Slack !
- Lisez attentivement les exemples car ils peuvent vous permettre d'identifier un travail à réaliser qui n'est pas précisé dans le sujet à première vue.
- Réfléchissez. Par pitié, par Thor, par Odin !

Chapitre III

Day-specific rules

- Utilisez aussi bien les annotations que les fichiers de configuration pour la définition des routes.
- Utilisez des types de formulaires définis dans le controller.
- Chaque page affichée doit être formatée proprement et devra contenir des balises DocType et HTML telles que `html`, `body`, `head`.
- Le serveur utilisé pour cette journée est celui intégré dans Symfony. Il devrait être démarré et arrêté à l'aide des commande console de Symfony.
- Seulement les URLs demandées devraient créer et afficher une page sans erreur. Les URLs non configurées devraient générer une erreur 404.
- Les URLs demandées devraient également fonctionner sans le slash final. Ex `/ex00` et `/ex00/` doivent fonctionner.
- Pour résoudre cet exercice vous ne devrez utiliser que SQL ou ORM ou les deux. Si cette condition n'est pas remplie, vous ne recevrez aucun point pour la solution que vous proposez.
- Chaque exercice doit avoir sa propre table dans la base de données.
- Les noms des tables dans la base de données devront être explicits et correctement formulés. Ex : si vous voulez créer une table pour un modèle appelé **Person**, la table devra s'appeler **persons**.

Chapitre IV

Exercice 00

	Exercice : 00
Exercice 00 : Table de base de données(BD) avec SQL	
Dossier de rendu : $ex00/$	
Fichiers à rendre : `Fichiers et répértoires de votre application`	
Fonctions Autorisées : `Toute méthode`	
Remarques : `n/a`	

Cet exercice vous aidera à apprendre les bases de l'utilisation de SQL dans le framework Symfony. Les attentes de cet exercice sont simples :

- Créer un bundle pour cet exercice, nommé ex00 ;
- Créer la requête SQL qui va créer une nouvelle table dans la BD ;
- Configurer la connexion à la BD ;
- Créer la route qui va appeler la génération de la table ;
- **Aucun ORM n'est permis dans cet exercice** ;
- La structure de la table devrait correspondre à celle décrite dans l'exercice.

Le résultat de cet exercice devra être une page contenant un bouton/lien permettant de créer la table et un message de succès/erreur selon la réussite ou non de la création de la table.

La structure de la table est la suivante :

- id - integer - primary key
- username - string - unique
- name - string
- email - string - unique
- enable - boolean
- birthdate - datetime

- address - long text

L'appel de la création de la table ne devrait pas échouer si la table existe déjà.

Chapitre V

Exercice 01

	Exercice : 01
Exercice 01 : Créer une table de BD avec ORM	
Dossier de rendu : $ex01/$	
Fichiers à rendre : `Fichiers et répértoires de votre application`	
Fonctions Autorisées : `Toute méthode`	
Remarques : `n/a`	

Après l'introduction du SQL avec Symfony, il est temps de passer à l'ORM (Object Relational Mapping). Dans cet exercice vous devrez créer la même table décrite dans l'**Exercice 00** Les attentes de cet exercice sont simples :

- Créer un bundle pour cet exercice, nommé ex01 ;
- Créer l'entité correspondant à la structure demandée ;
- Créer la route qui va appeler la génération de la table ;
- **Aucun SQL n'est permis dans cet exercice** ;
- La structure de la table devrait correspondre à celle décrite dans l'exercice.

Le résultat de cet exercice devra être une page contenant un bouton/lien permettant de créer la table et un message de succès/erreur selon la réussite ou non de la création de la table.

La structure de la table est la suivante :

- id - integer - primary key
- username - string - unique
- name - string
- email - string - unique
- enable - boolean
- birthdate - datetime

- address - long text

L'appel de la création de la table ne devrait pas échouer si la table existe déjà.

Astuce : La génération d'entité et la création de la table peuvent être réalisés en utilisant des commandes de l'ORM doctrine. Les commandes peuvent être trouvées à l'aide de la commande **php appconsole doctrine** dans le terminal. Aussi, les commandes terminal peuvent être appelées dans les controleurs Symfony.

Chapitre VI

Exercice 02

	Exercice : 02
Exercice 02 : Insertion et lecture avec SQL	
Dossier de rendu : $ex02/$	
Fichiers à rendre : `Fichiers et répértoires de votre application`	
Fonctions Autorisées : `Toute méthode`	
Remarques : `n/a`	

En utilisant les connaissances précédemment acquises, vous devrez être capables de faire d'autres opérations depuis le paquet CRUD. Commençons par Insert et Read.

Cet exercice vous permettra d'apprendre à mapper des données depuis les formulaires web vers des requêtes SQL. Les attentes de cet exercice sont les suivantes :

- Créer un bundle pour cet exercice, nommé ex02 ;
- Créer une table séparée pour cet exercice utilisant la même structure fournie dans les exercices précédents.
- **Dans cet exercice vous avez droit seulement au SQL, pas d'ORM autorisé ;**
- Créer un formulaire Symfony et une action de type "show form" dans le controleur ;
- Créer un endpoint pour le controleur qui gérera les données transmises par le formulaire ;
- Assurez vous que les champs à valeurs uniques ne généreront pas d'exceptions lors de l'insertion ;
- Créer une page web contenant une table HTML dans laquelle vous montrerez le contenu de la table.

Ni l'appel pour créer la table ni l'insertion ne devrait échouer si la table existe déjà ou si les données sont déjà présentes dans la table.

Chapitre VII

Exercice 03

	Exercice : 03
Exercice 03 : Insertion et lecture avec ORM	
Dossier de rendu : $ex03/$	
Fichiers à rendre : `Fichiers et répértoires de votre application`	
Fonctions Autorisées : `Toute méthode`	
Remarques : `n/a`	

De retour à l'ORM, les opérations demandées dans l'exercice précédent peuvent être réalisées en utilisant seulement des commandes spécifique de l'ORM Doctrine. Pour insérer et lire avec ORM la liste des opérations requises est la suivante :

- Créer un bundle pour cet exercice, nommé ex03 ;
- Créer une nouvelle entité avec sa propre table utilisant la même structure fournie dans les exercices précédents ;
- **Cet exercice sera résolu en utilisant seulement l'ORM Doctrine ;**
- Créer un formulaire Symfony et et mappez le à l'entité créée précédemment ;
- Créer des endpoints spécifiques dans le controleur pour l'affichage du formulaire (show form), la gestion des données insérées et l'affichage des données de la table ;
- Assurez vous que les champs a valeurs uniques ne généreront pas d'exceptions lors des insertions ;
- Créer une page web contenant une table HTML dans laquelle vous montrerez le contenu de la table ;
- Le formulaire devra contenir une validation ;
- Les commandes ORM ne devront pas être géré dans le controleur.

Ni l'appel pour créer la table ni l'insertion ne devrait échouer si la table existe déjà ou si les données sont déjà présentes dans la table.

Le version finale du résultat de cet exercice devra permettre à un utilisateur d'insérer des informations dans la BD par le formulaire fourni et afficher la liste de toutes les entités

présentes dans la BD.

Astuce : Pour la liste des commandes Doctrine vous pouvez les afficher à l'aide de la commande **php appconsole doctrine** dans le terminal.

Chapitre VIII

Exercice 04

	Exercice : 04
Exercice 04 : SQL - supprimer en utilisant un request conditionné	
Dossier de rendu : *ex*04/	
Fichiers à rendre : `Fichiers et répértoires de votre application`	
Fonctions Autorisées : `Toute méthode`	
Remarques : `n/a`	

Pour la suite des opérations spécifiques CRUD, nous arrivons à la suppression (Delete). Pour cet exercice, en utilisant seulement SQL et PHP, vous devrez écrire un bundle spécifique Symfony qui gérera cette opération. La suppression devra être réalisée en utilisant un endpoint de l'application qui accepte un request à condition, tel que */delete/ ?id=* ou */delete/*{*id*} , ou **id** représente un identifiant unique de l'objet à supprimer.

La liste des opérations requises par cet exercice est la suivante :

- Créer un bundle nommé ex04 ;
- Créer une table séparée pour cet exercice ;
- Créer une méthode de controleur qui a une route avec un paramètre pour supprimer des données ;
- Créer le code SQL pour supprimer les données de la base de données en utilisant la condition ;
- Vérifier qu'un élément existe dans la BD ;
- Créer une page HTML dans la quelle on voit les données existantes dans la table et ajouter les boutons permettant de supprimer chaque ligne ;
- Utilisez seulement du code SQL pour supprimer des données dans la BD, **pas d'ORM autorisé** ;
- Ajouter un message de succès/erreur sur la page web afin d'afficher le résultat de l'opération de suppression.

Chapitre IX

Exercice 05

	Exercice : 05
Exercice 05 : ORM - supprimer en utilisant un request conditionné	
Dossier de rendu : $ex05/$	
Fichiers à rendre : `Fichiers et répértoires de votre application`	
Fonctions Autorisées : `Toute méthode`	
Remarques : `n/a`	

Pour finir avec CRUD dans Symfony, ORM fournit aussi des méthodes permettant de supprimer des entités dans la BD. Cet exercice vous aidera à trouver comment utiliser la suppression dans Doctrine ORM. A l'aide de demandes de l'exercice précedent, vous devrez implémenter l'option **Supprimer/Delete** en utilisant seulement ORM. Plus précisément, les éléments requis pour cet exercice sont :

- Créer un nouveau bundle nommé ex05 ;
- Créer une nouvelle entité pour cet exercice. Vous pouvez prendre l'entité de l'exercice 00 comme exemple ;
- Créer une méthode de controleur qui a une route avec un paramètre pour supprimer des données ;
- Vérifier qu'un élément existe dans la BD ;
- Créer une page HTML dans la quelle on voit les données existantes dans la table et ajouter les boutons permettant de supprimer chaque ligne ;
- Utiliser seulement des commandes ORM pour supprimer des données dans la BD ;
- Ajouter un message de succès/erreur sur la page web afin d'afficher le résultat de l'opération de suppression.

Chapitre X

Exercice 06

	Exercice : 06
Exercice 06 : SQL : mise à jour des informations en base de données	
Dossier de rendu : $ex06/$	
Fichiers à rendre : `Fichiers et répértoires de votre application`	
Fonctions Autorisées : `Toute méthode`	
Remarques : `n/a`	

Le dernier élément du package CRUD est **Update**. Pour cet exercice, en utilisant seulement SQL et PHP, vous devrez créer un bundle Symfony capable de mettre à jour une entité existante. Plus précisément, chaque entité est mappée à au moins une table dans la BD. DU coup vous devrez mettre à jour la/les table(s). Pour résoudre cet exercice vous devrez respecter les conditions suivantes :

- Créer un bundle nommé ex06 ;
- Créer une table séparée dans la base de données pour cet exercice. Vous pouvez utiliser comme exemple la structure de la table dans l'exercice 00 ;
- Créer une page web pour afficher les entités qui existent dans la base de données utilisant une table ;
- Pour chaque élément de la table HTML dans la page web, ajouter un bouton 'Update' qui amènera l'utilisateur à la page de modification/édition ;
- Créer un formulaire qui sera automatiquement rempli avec les détails de l'entité modifiée et permettra à l'utilisateur de changer chacun des champs ;
- Concevoir un controleur avec les méthodes nécessaires pour afficher et mettre à jour les entités dans la base de données ;
- Utiliser seulement du code SQL pour gérer les mises à jour dans la base de données, pas de ORM permis ;
- Aucune validation ou autres conditions ne sont requises ici (champs uniques, valeurs insérées correctement, etc.) ;

- Afficher un message de succès/erreur dans la page listant les entités, après la mise à jour d'une entité.

Dans la version finale, cet exercice fournira une option pour mettre à jour des informations dans la base de données. En la réalisant, vous aurez finalisé les opérations de type CRUD utilisant SQL.

Chapitre XI

Exercice 07

	Exercice : 07
Exercice 07 : ORM : mise à jour des informations en base de données	
Dossier de rendu : $ex07/$	
Fichiers à rendre : `Fichiers et répértoires de votre application`	
Fonctions Autorisées : `Toute méthode`	
Remarques : `n/a`	

Comme chaque opération CRUD aura été implémentée en utilisant aussi bien SQL que ORM, cet exercice va mettre en pratique l'utilisation l'utilisation de l'opération de mise à jour (update) dans ORM. Vous devez concevoir un nouveau bundle Symfony capable d'afficher et mettre à jour des informations dans la base de données. Du fait de l'usage d'ORM la base de données est reflétée dans le code PHP par des entités. En conclusion, les conditions suivantes doivent être satisfaites afin de résoudre cet exercice :

- Créer un bundle nommé ex07 ;
- Créer une nouvelle entité Doctrine (vous pouvez utiliser la structure des exercices précédents) mappée à une nouvelle table dans la base de données ;
- Concevoir une table HTML dans laquelle vous afficherez les informations présentes dans la base de données ;
- Créer un formulaire Symfony mappé sur l'entité créée précédemment ;
- Créer les méthodes controleur capables de gérer le listing„ ainsi que l'affichage et la mise à jour des entités ;
- Aucune validation n'est requise pour cet exercice ;
- Seulement les commandes ORM sont autorisées pour modifier les informations dans la base de données ;
- Afficher un message de succès/erreur suivant le résultat de l'opération.

Chapitre XII

Exercice 08

	Exercice : 08
Exercice 08 : SQL - Mise à jour la structure d'une table, création de relations	
Dossier de rendu : *ex*08/	
Fichiers à rendre : `Fichiers et répértoires de votre application`	
Fonctions Autorisées : `Toute méthode`	
Remarques : `n/a`	

Félicitations, vous maitrisez maintenant les opérations CRUD aussi bien avec SQL qu'avec ORM. Vous devriez être capable d'implémenter ces opérations par les 2 méthodes, sans hésitation. Mais, du fait que les spécifications d'un projet peuvent changer, un développeur devrait aussi savoir comment changer la structure de la base de données et, encore plus important, créer des relations entre les tables d'une base de données.

Comme vous le save peut-être déjà, dans une base de données relationnelle, nous avons 3 types de relations entre les tables :

- Un-à-un (one-to-one), exemple : une roue à un pneu et vice-versa ;
- Un-à-plusieurs (one-to-many), exemple : une voiture à un propriétaire, un propriétaire peut avoir plusieurs voitures ;
- Plusieurs-à-plusieurs (many-to-many), exemple : une commande peut avoir plusieurs produits, un produit peut appartenir à plusieurs commandes.

Pour cet exercice, vous devez concevoir un bundle Symfony qui est capable de créer une table avec une structure donnée, modifier la structure de cette table et ajouter une nouvelle colonne mais aussi créer deux relations pour deux autres table (à créer aussi) utilisant des clés étrangères.

Plus concrètement, vous devez :

- Créer un bundle nommé ex08 ;
- Concevoir le code SQL nécessaire pour créer et modifier la structure d'une table et pour créer des relations entre tables ;
- Créer une table initiale (named *persons*) utilisant la structure de l'ex.00 mais n'ajoutez pas le champs adresse. La création de la table devrait être faite en

utilisant un lien dans la page d'accueil et un message de succès/erreur devra être affiché ;

- Créer une nouvelle méthode de controleur qui va ajouter une nouvelle colonne dans la table (ex : *marital_status :ENUM(single, married, widower)*)
- Créer une nouvelle méthode de controleur pour créer 2 nouvelles tables : **bank_accounts** et **addresses**. Vous avez la liberté de la structure de ces 2 tables ;
- Créer une relation un-à-un entre les tables **bank_accounts** et **person** ainsi qu'une relation un-à-plusieurs ;
- Utilisez **seulement du code SQL** pour la création des tables et des relation ;
- Pour chaque opération (création de table, modification, création de relation retournez un message de succès/erreur ;

Chapitre XIII

Exercice 09

	Exercice : 09
Exercice 09 : ORM - Mise à jour la structure d'une table, création de relations	
Dossier de rendu : *ex*09/	
Fichiers à rendre : `Fichiers et répértoires de votre application`	
Fonctions Autorisées : `Toute méthode`	
Remarques : `n/a`	

Il est temps maintenant de réaliser les même opérations de mise à jour d'entité et de création de relation avec ORM. Comme mentionné dans l'exercice précédent, il y a 3 types de relations entre les entités (tables). Par contre, en Symfony ces relations sont faites entre les entités et se reflètent dans la base de données par le mappage automatique des clé étrangères entre les tables (ou la création de tables de liaison pour les relations de type plusieurs-à-plusieurs).

Vous devez accomplir les tâches suivantes :

- Créer un bundle nommé ex09 ;
- Créer une entité utilisant la structure des exercices précédents (la structure de l'entité *Person*)
- Ajouter un nouveau champ (comme demandé dans l'exercice précédent) dans l'entité et mettre à jour la table de la base de données ;
- Créer deux nouvelles entités (*BankAccount* et *Address*) et créer des relations bidirectionnelles entre ces 2 entités et l'entité *Person* ;
- Utiliser **seulement des commandes ORM** pour gérer la structure de la base de données ;
- Créer des méthodes controleur pour gérer la création, la mise à jour et les relations.

Note : comme Doctrine ne fournit pas de moyens de modifier les structures des tables et garder une trace, vous devrez utiliser des migrations Doctrine afin d'ajouter une nouvelle colonne à une table.

Chapitre XIV

Exercice 10

	Exercice : 10
Exercice 10 : SQL & ORM - Insérer en base de données à partir d'un fichier	
Dossier de rendu : $ex10/$	
Fichiers à rendre : `Fichiers et répértoires de votre application`	
Fonctions Autorisées : `Toute méthode`	
Remarques : `n/a`	

Aujourd'hui, la source des informations stockées en base de données peut être de plusieurs types : entrée humaine, données reçues en utilisant des appels à des services web externes, données auto-générées ou des informations lues dans des fichiers.

Dans cet exercice, nous allons nous focaliser sur le stockage des informations obtenues par la lecture de fichiers.

Vous devez proposer une solution utilisant aussi bien SQL que ORM **en même temps**. Les opération nécessaires pour réussir cet exercice sont :

- Créer un bundle nommé ex10 ;
- Créer une nouvelle base de données pour SQL et une nouvelle entité (avec table associée) pour la partie ORM ;
- Concevoir une méthode controleur qui lira le contenu d'un fichier et l'insérera et dans la table SQL et dans la table associée à l'entité ORM ;
- La méthode controleur peut être appelée par un lien dans une page web. Les informations devraient ensuite être visibles dans un tableau HTML ;
- S'assurer que les informations sont correctement stockées et que des champs ne soient pas perdus pendant les opération de lecture dans le fichier et écriture dans la base de données.

Cet exercice doit être résolu en utilisant aussi bien SQL que ORM.

Astuce : s'assurer que le fichier dans lequel vous voulez lire les informations détient les permissions/droits correctes.

Chapitre XV

Exercice 11

	Exercice : 11
Exercice 11 : SQL - Requête complexe avec join, sort et condition	
Dossier de rendu : $ex11/$	
Fichiers à rendre : `Fichiers et répértoires de votre application`	
Fonctions Autorisées : `Toute méthode`	
Remarques : `n/a`	

De nos jours, les pages web sont plus complexes que des simples pages HTML, quand on utilise une base de données pour récupérer des informations, le tri et les requêtes conditionnelles sont très usuels. Aussi, du fait des principes de conception des bases de données, des jointures entre les tables doivent être faites afin de récupérer les informations voulues. Comme exemple, imaginons un site de e-commerce : pour qu'un client cherche un produit donné selon son type, sa catégorie et sa couleur, le back-end doit faire des jointures entre les tables des catégories et des produits, mais aussi filtrer les résultats selon la couleurs et le type. En plus si le client veut afficher la liste dans l'ordre croissant des prix, un tri des résultats doit être effectué. Habituellement, ces jointures, conditions et tris ont meilleur compte à être effectués directement dans le language de requête car cela est bien plus rapide et optimisé que de récupérer toutes les informations en vrac les stoker dans des variable et appliquer des filtres et tris par la suite.

Pour cet exercice, vous devez :

- Créer un bundle nommé ex11 ;
- Vous pouvez utiliser les tables créés dans l'exercice 09 du fait des relations déjà existentes (vous pouvez utiliser les relations un-à-un) ;
- Concevoir une page HTML avec un tableau dans lequel vous présenterez les informations stockées dans la table ;
- Dans la tableau HTML, ajouter un formulaire proposant des options de filtre et de tri (ex. filtrer par date, trier par nom) ;
- Créer les méthodes nécessaires dans le controleur pour gérer le listing et la génération des résultats après validation des tris et filtres ;

- S'assurer que les informations affichées dans le tableau HTML provient de plus d'une seule table ;
- Utiliser **seulement SQL** pour les requêtes exécutées dans la base de données ;
- Pour la requête d'obtention des informations dans la base de données, s'assurer d'utiliser **jointure, condition et tri**. Si un de ces éléments n'est pas utilisé, aucun point de sera attribué pour cet exercice ;
- Ajouter des validations pour les paramètres de la requête.

Chapitre XVI

Exercice 12

	Exercice : 12
Exercice 12 : ORM - Requête complexe avec join, sort et condition	
Dossier de rendu : $ex12/$	
Fichiers à rendre : `Fichiers et répértoires de votre application`	
Fonctions Autorisées : `Toute méthode`	
Remarques : `n/a`	

Comme vous l'imaginez, étant donné que l'exercice précédent représente une partie importante d'une application web, cet exercice est également basé sur la même idée. La grande différence est que, cette fois, vous devrez utiliser **seulement des méthodes ORM** pour arriver au résultat demandé.

En plus des attentes de l'exercice précédent, pour cet exercice vous devrez respecter les consignes suivantes :

- Créer un bundle nommé ex12
- Vous pouvez utiliser les entités créées dans l'exercice 10, avec des relations un-à-un définies entre eux ;
- Assurez-vous de créer des requêtes spécifiques pour récupérer les informations voulues ;
- Ajouter des validations pours les paramètres des requêtes.

Chapitre XVII

Exercice 13

	Exercice : 13
Exercice 13 : ORM - Opérations CRUD complètes	
Dossier de rendu : $ex13/$	
Fichiers à rendre : `Fichiers et répértoires de votre application`	
Fonctions Autorisées : `Toute méthode`	
Remarques : `n/a`	

Bien joué ! Vous êtes presque à la fin de cette journée. Mais afin de finir, vous allez devoir résoudre 2 exercices sympa. Ils reflèteront tous ce que vous avez assimilé lors des exercices précédents. L'idée de l'exercice présent est d'implémenter toutes les opérations de type CRUD en utilisant des commandes ORM.

Pour valider cet exercice, vous devrez :

- Créer un bundle nommé ex13
- Créer une nouvelle entité de type `Employee` avec la structure suivante :
  - id - integer
  - firstname - string
  - lastname - string
  - email - string
  - birthdate - datetime
  - active - boolean
  - employed_since - datetime
  - employed_until - datetime
  - hours - ENUM (8,6,4)
  - salary - integer
  - position - ENUM(manager, account_manager, qa_manager, dev_manager,

ceo, coo, backend_dev, frontend_dev, qa_tester)

- Créer une relation un-à-plusieurs entre l'entité `Employee` et elle même qui représentera le supérieur hiérarchique d'un employé (chaque employé devra avoir un responsable/supérieur hiérarchique) ;
- Implémenter un endpoint de lecture, un de modification, un de suppression et un de création pour l'entité `Employee` ;
- Le formulaire pour créer/modifier une entité devrait contenir des validations différentes selon de type de champs (assurez vous que certains soient obligatoires, uniques, email au format correct, etc.) ;
- Pour chaque opération vous devrez créer des messages de succès/erreur ;
- L'application de devrait pas "crasher" si les demandes sont incorrectes (ex : suppression d'une information inexistante, insertion d'information dupliquée, etc.).

Chapitre XVIII

Exercice 14

	Exercice : 14
Exercice 14 : SQL - Injections fonctionnelles	
Dossier de rendu : $ex14/$	
Fichiers à rendre : `Fichiers et répértoires de votre application`	
Fonctions Autorisées : `Toute méthode`	
Remarques : `n/a`	

Comme promis, ceci est le dernier exercice du jour, mais un des plus interessants. Peut-être avez vous déjà entendu l'expression *injection SQL* mais vous n'avez probablement jamais essayé d'en créer une. Cet exercice est là pour prouver la vulnerabilité de SQL et HTML en absence d'une validation des données dans les sites web. Une injection SQL est une technique consistant à attaquer des sites web en exécutant des requêtes SQL sur la base de données de du site web (récupération de données spécifiques, changement des données, suppression des données, effacement de la base de données) lorsque il n'y a pas de vérification des informations provenant des champs des formulaires.

Pour cet exercice vous devez :

- Créer un bundle nommé ex14 ;
- Créer une méthode controleur qui va insérer une nouvelle table dans la base de données, si la table n'existe pas déjà ;
- Le statut de la table devra être visible dans la page web à l'aide d'un message (*Table existe* ou *Table n'existe pas*)
- Créer une formulaire HTML dans une page, mappé à une action du controleur permettant d'insérer des informations dans la base de données ;
- Ajouter une injection SQL fonctionnelle en utilisant Javascript sur l'évènement submit du formulaire.
- Le résultat de l'injection devrait être visible dans la page d'accueil ou, si l'information a été altéré, le résultat devra être visible dans le listing HTML des informations en provenance de la base de données.