Modèles, Entités et Gestion des Données avec l'ORM dans Symfony

1. ## Objectifs

   - Développer et intégrer efficacement les modèles, les entités et la gestion des données avec l'ORM dans Symfony pour un projet spécifique, en suivant les meilleures pratiques de développement.

2. ## Introduction générale

   - Dans le développement d'applications web, la gestion des données est une composante essentielle. Symfony, un framework PHP populaire, offre une infrastructure robuste pour la manipulation des données à travers son système d'ORM (Object-Relational Mapping) basé sur Doctrine.

3. ## Introduction à l'ORM et Symfony

   1. **Qu'est-ce que l'ORM ? :**

      - L'ORM, ou Object-Relational Mapping, est une technique de programmation qui permet de mapper les objets de votre application aux tables de votre base de données relationnelle.
      - L'objectif principal de l'ORM est de simplifier et de rationaliser le processus d'interaction entre le code applicatif écrit dans un langage de programmation orienté objet (comme PHP, Java, Python, etc.) et les données stockées dans une base de données relationnelle (comme MySQL, PostgreSQL, Oracle, etc.).

        1. **Abstraction de la base de données :**

           - L'ORM permet d'abstraire les détails spécifiques à la base de données, tels que la syntaxe SQL et les requêtes, en fournissant une interface orientée objet pour interagir avec les données. Cela permet aux développeurs de se concentrer sur la logique métier de leur application plutôt que sur les détails de bas niveau de la base de données.

        2. **Mapping Objet-Relationnel**

apcpedagogie.com

- L'ORM réalise le mapping entre les objets de votre application et les tables de la base de données. Chaque classe d'objet (entité) est associée à une table dans la base de données, et les propriétés de la classe (attributs) sont associées aux colonnes de la table. Les relations entre les objets (telles que les associations One-to-One, One-to-Many, Many-to-Many) sont également gérées par l'ORM.

3. **Opérations CRUD**

   - L'ORM fournit des méthodes pour effectuer les opérations CRUD (Create, Read, Update, Delete) sur les objets de votre application. Ces opérations sont ensuite traduites en requêtes SQL appropriées et exécutées sur la base de données.

4. **Gestion des relations**

   - L'ORM facilite la gestion des relations entre les objets. Par exemple, si une entité A est associée à une entité B via une relation Many-to-One, l'ORM se charge de récupérer automatiquement les objets associés lorsque cela est nécessaire, en suivant les règles de la relation définie dans le mapping.

5. **Portabilité et flexibilité**

   - En utilisant un ORM, vous pouvez rendre votre application plus portable, car elle n'est pas directement liée à une base de données spécifique. Si vous devez changer de système de gestion de base de données, vous pouvez souvent le faire en modifiant simplement la configuration de l'ORM, sans avoir à réécrire de grandes parties de votre code applicatif.

2. **Pourquoi utiliser un ORM ? :**

   - Simplifie le développement en évitant d'écrire des requêtes SQL complexes.
   - Abstrait la base de données, ce qui permet de changer de système de gestion de base de données sans changer le code métier.
   - Fournit des outils pour la validation, la gestion des relations, etc.

3. **Symfony et Doctrine ORM :**

   - Symfony utilise Doctrine comme ORM par défaut.

apcpedagogie.com

- Doctrine offre des fonctionnalités puissantes pour manipuler les entités, les relations et les requêtes.

4. **Mise en application**

1. **Configuration**

- Comme à chaque fois, il est d'abord nécessaire d'installer les bundles nécessaires pour manipuler la base de données avec un ORM. Il vous faut donc exécuter la commande ci_dessous :

```
composer require symfony/orm-pack
```

- On va également installer, si vous ne l'avez pas encore fait, le bundle "maker" qui contient des outils pour générer du code sous Symfony grâce à la console.

```
composer require symfony/maker-bundle --dev
```

- Une fois ces deux éléments installés, il faut configurer la connexion à la base de données. Pour ce faire, il faut éditer le fichier .env à la racine de votre projet, qui doit normalement contenir une ligne d'exemple.

```
# .env

# customize this line!
DATABASE_URL="mysql://db_user:db_password@127.0.0.1:3306/db_name"

# to use sqlite:
# DATABASE_URL="sqlite:///%kernel.project_dir%/var/app.db"
```

2. **Création de la base de données avec Doctrine :**

- Une fois le fichier à jour avec vos données, vous pouvez créer votre base de données depuis la console.
- Symfony propose une commande pour créer la base de données à partir de la configuration définie. Exécutez la commande suivante dans votre terminal :

apcpedagogie.com

```
php bin/console doctrine:database:create
```

- Cette commande utilisera les informations de configuration que vous avez définies pour créer une nouvelle base de données dans votre serveur MySQL.
- Les modifications de structure de votre base de données devront être réalisées avec la console pour que Symfony puisse faire le lien entre les tables et l'ORM

4. Entités dans Symfony :

1. **Qu'est-ce qu'une entité ? :**

   - Une entité est une classe PHP qui représente un objet métier et est mappée à une table de base de données.

2. **Création d'une entité :**

   - Les entités sont généralement créées dans le répertoire src/Entity.
   - Chaque entité est annotée avec **`@ORM\Entity`**.
   - Les propriétés de l'entité représentent les colonnes de la table.
   - Pour créer une entité avec Symfony et Doctrine en utilisant la commande make:entity, suivez ces étapes :
   - Assurez-vous d'avoir installé le bundle "**`maker`**" de Symfony. Si ce n'est pas déjà fait, vous pouvez l'installer en exécutant la commande suivante :**`composer require symfony/maker-bundle --dev`**
   - Utilisez la commande **`make:entity`** (qui est dans le bundle maker) pour avoir une série de question vous permettant de créer votre entité avec l'utilisation de l'ORM Doctrine. Vous pouvez créer une nouvelle entité ou modifier (ajouter des champs) une entité déjà existante en saisissant son nom.

     ```
     php bin/console make:entity
     ```

   - Vous allez devoir répondre à une suite de question avec le nom de l'entité (par défaut cela donnera le nom de la table), et les champs à créer. Dans Symfony une entité possède toujours un champs id, qui est la clé primaire et qui est auto-incrémenté. Vous ne devez donc pas l'ajouter dans la console.
   - Pour la création d'un champs, il vous faudra donner :
     - son type

apcpedagogie.com

- sa taille le cas échéant
- si ce champs peut être null
- s'il doit être unique (index)

Vous pouvez obtenir la liste des types supportés en tapant "?" à la question du type. Une fois terminé, le fichier d'Entité et le repository associé sont générés.

3. **Annotations de Doctrine pour les entités :**

   - @ORM\Column : Définit une colonne de base de données.
   - @ORM\Id : Définit une clé primaire.
   - @ORM\GeneratedValue : Indique que la valeur de la clé primaire est générée automatiquement.
   - D'autres annotations pour définir les relations, les contraintes, etc.

5. Relations entre les Entités :

1. **Types de relations :**

   - OneToOne
   - OneToMany
   - ManyToOne
   - ManyToMany

2. **Définition des relations :**

   - Utilisation d'annotations telles que @ORM\OneToMany, @ORM\ManyToOne, etc.
   - Les relations sont généralement définies dans les deux sens (propriété inverse) pour faciliter la navigation.

3. **Cascades et suppression en cascade :**

   - Définir les actions à effectuer sur les entités associées lorsqu'une entité parent est supprimée.

6. Manipulation des Entités :

1. **EntityManager :**

apcpedagogie.com

- L'EntityManager est utilisé pour effectuer des opérations sur les entités, telles que la création, la mise à jour, la suppression, etc.

2. **Opérations CRUD :**
   - Créer une nouvelle entité avec persist().
   - Mettre à jour une entité avec flush().
   - Supprimer une entité avec remove().

3. **Composant Validator de Symfony :**
   - Utilisé pour valider les données avant de les enregistrer dans la base de données.
   - Définition de contraintes de validation dans les entités à l'aide d'annotations ou de fichiers de configuration YAML/XML.
   - Validation automatique lors de l'appel à flush().

4. **QueryBuilder :**
   - Utilisé pour construire des requêtes SQL de manière orientée objet.
   - Offre une syntaxe fluide et intuitive pour construire des requêtes complexes.

5. **Exemples de requêtes :**
   - Sélection de données avec select().
   - Filtrage de données avec where().
   - Joindre des entités avec join().
   - Utilisation de fonctions SQL avec addSelect(), andWhere(), etc.

6. **Doctrine Migrations :**
   - Utilisé pour gérer les changements de schéma de base de données de manière contrôlée.
   - Génère des migrations pour les changements de schéma.
   - Applique les migrations à la base de données pour mettre à jour le schéma.

7. **Sécurité des requêtes :**
   - Utilisation de requêtes paramétrées pour éviter les attaques par injection SQL.

8. **Bonnes pratiques de conception :**

apcpedagogie.com

- Séparation des préoccupations (principe de responsabilité unique).
- Utilisation de services pour encapsuler la logique métier.

Cours Symfony

Source : https://www.apcpedagogie.com/modeles-entites-et-gestion-des-donnees-avec-lorm-dans-symfony · téléchargé le 05/11/2026 · © APC Pédagogie · apcpedagogie.com

apcpedagogie.com