Faire coexister le monde relationnel et objet

Plan

Pour la pratique

Séparation en couches

Accès aux données

Idée de base

Les problèmes

Les avantages

Exemples

Pour la pratique

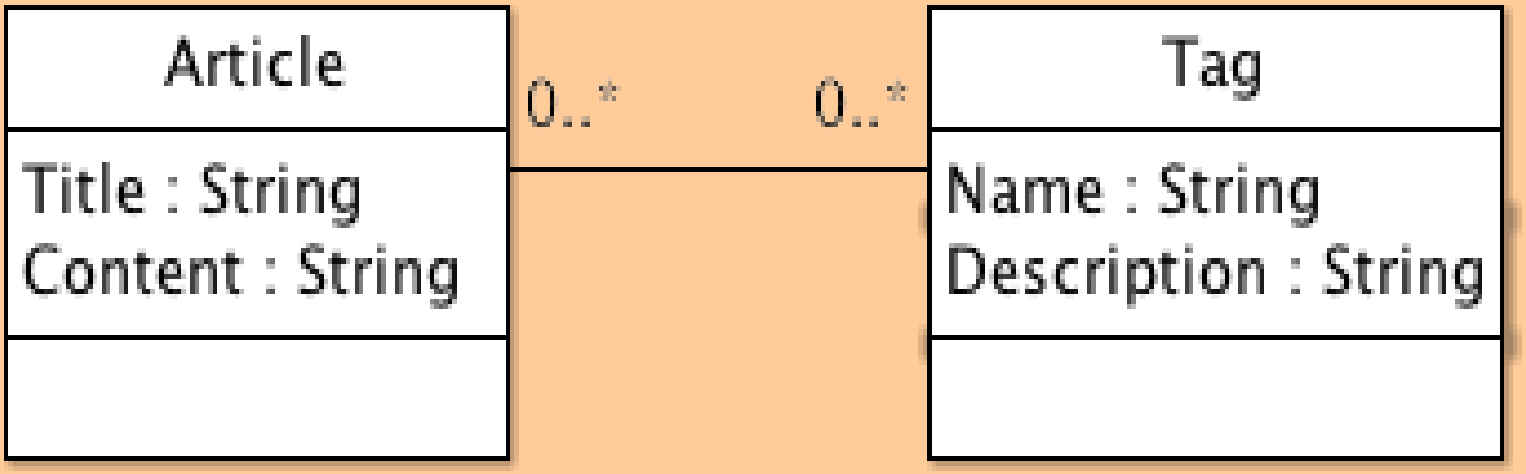
Un blog tout simple basé « 2 pages » :

- **une affichera un article**
- **une affichera un tag**

hébergé sur un serveur http

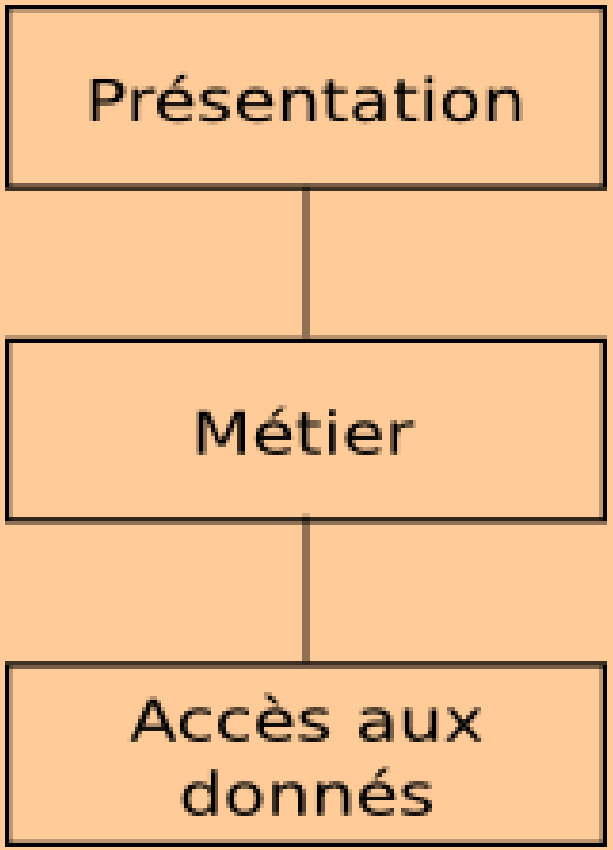
Pour la pratique

Un blog tout simple



Séparation en couches

Le modèle à 3 couches



Avantages :

Compréhensibilité

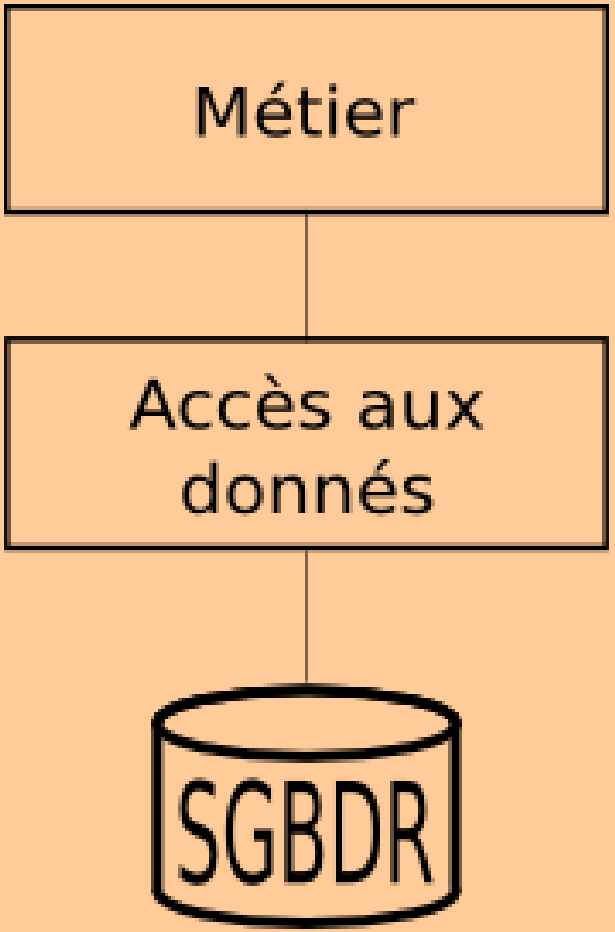
Maintenabilité

Réutilisabilité

...

Accès aux données

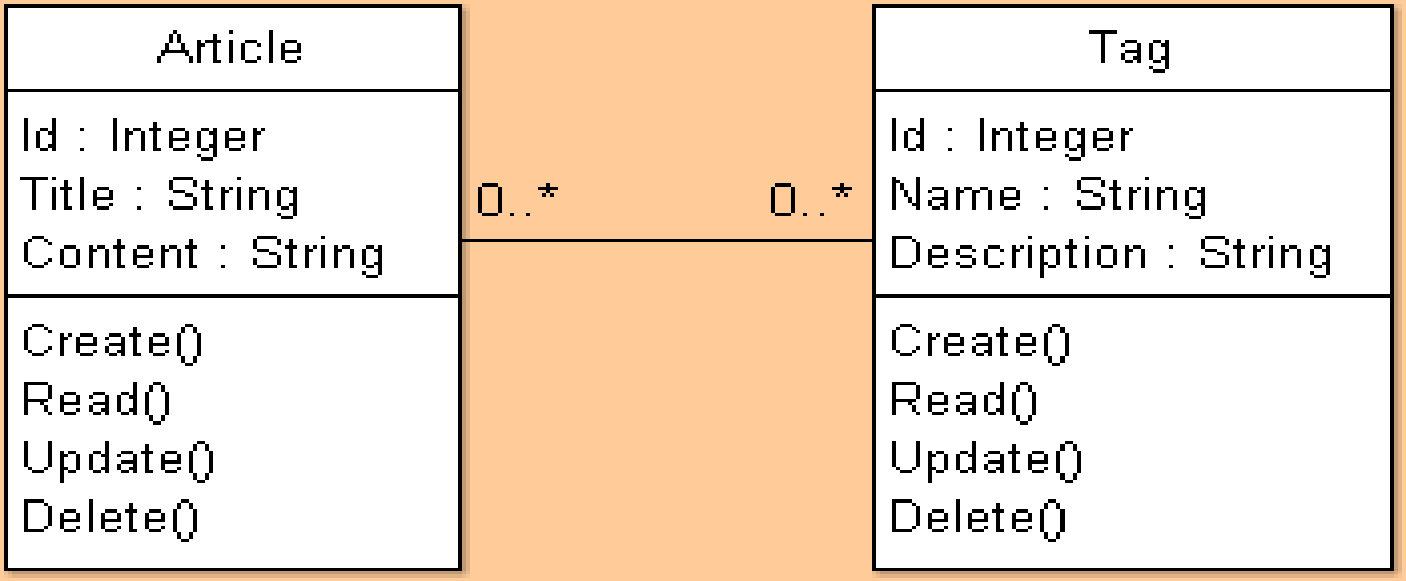
Métier / objet



SGBDR / relationnel

Idée de base

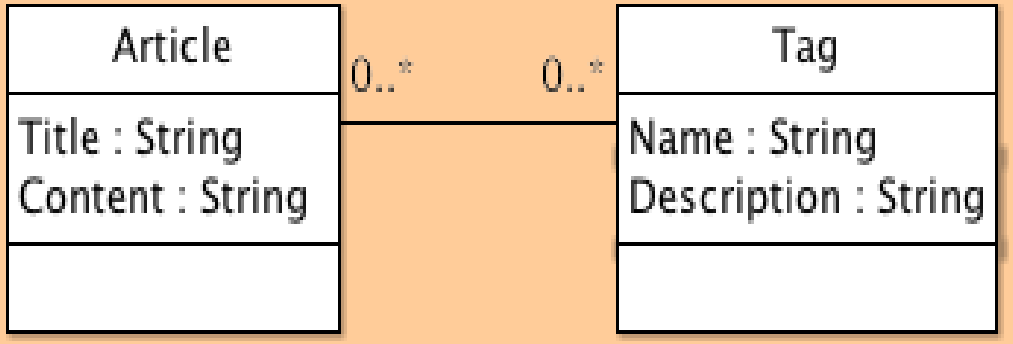
Objets persistants



Pattern CRUD (Create, Read, Update, Delete)

Idée de base

```
-- Structure de la table Articles

CREATE TABLE Article (
  ID int(11) NOT NULL auto_increment,
  Title varchar(100) NOT NULL,
  Content mediumtext NOT NULL,
  PRIMARY KEY  (ID)
);

-- Structure de la table Tags

CREATE TABLE Tag (
  ID int(11) NOT NULL auto_increment,
  Name varchar(100) NOT NULL,
  Description varchar(500) default NULL,
  PRIMARY KEY  (ID)
);

-- Structure de la table TagsArticles

CREATE TABLE TagArticle (
  IdArticle int(11) NOT NULL default '0',
  IdTag int(11) NOT NULL default '0',
  PRIMARY KEY  (IdArticle,IdTag)
);
```



Idée de base

Une classe correspond à une table

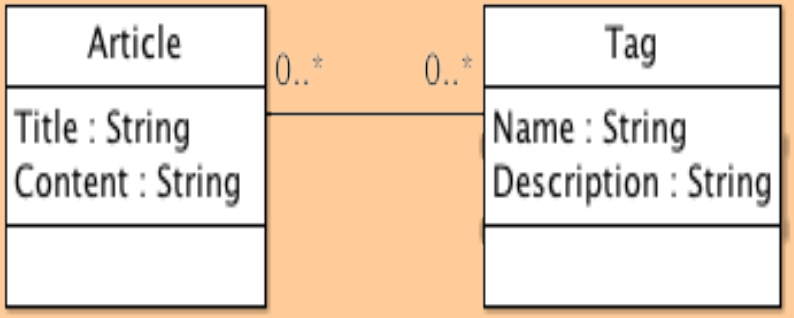
Un attribut correspond à un champ

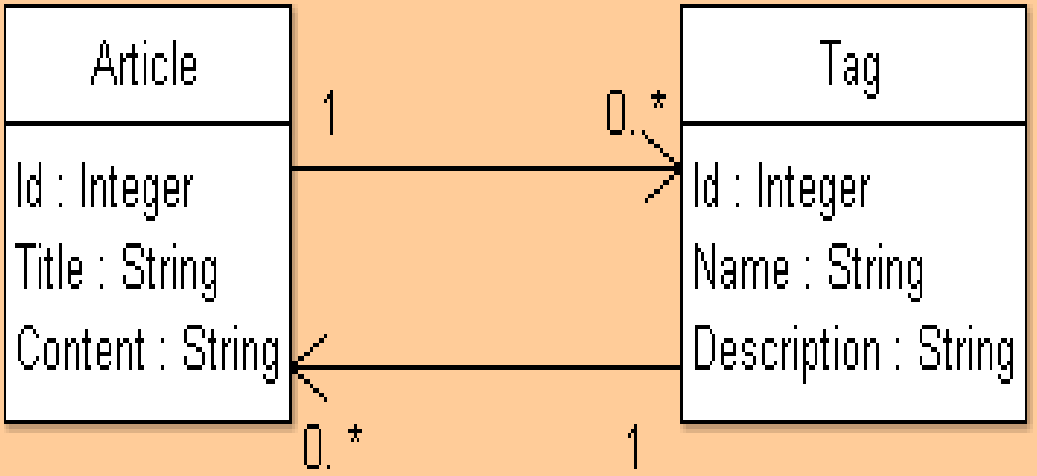
=> Facile de faire un petit automate qui « fabrique » les requêtes

Les problèmes

Les 2 mondes sont différents :

1 relation donne 1 ou 2 références

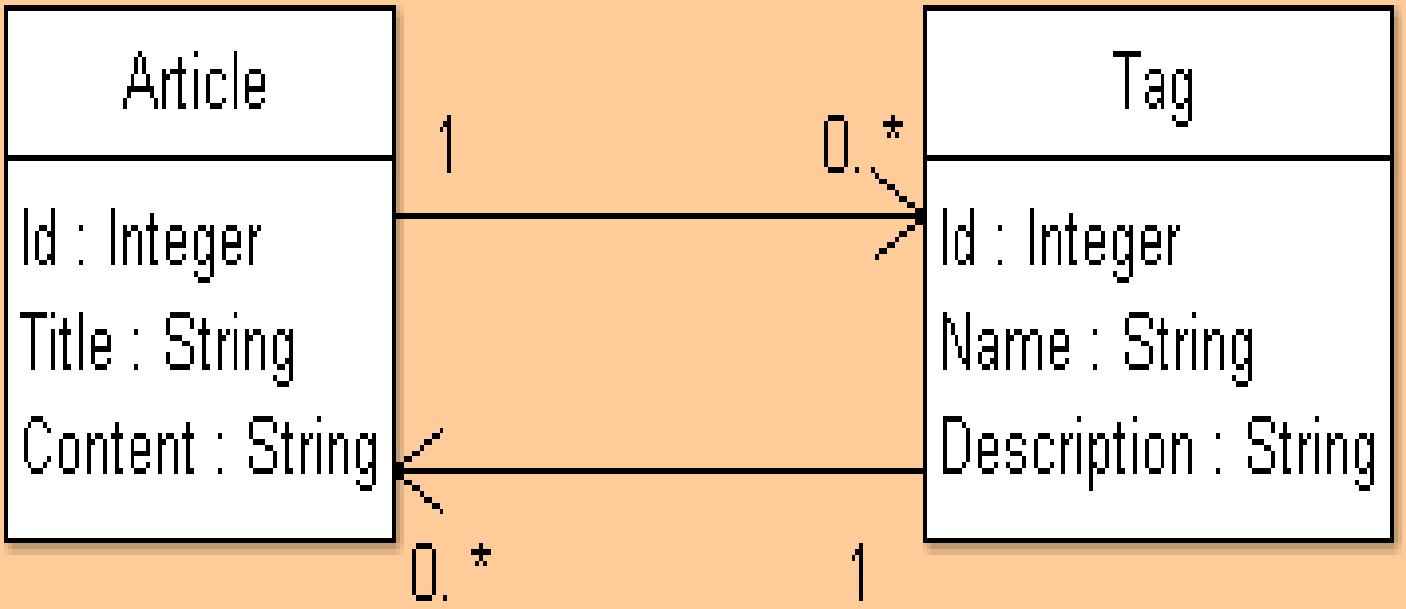




Les problèmes

Les 2 mondes sont différents :

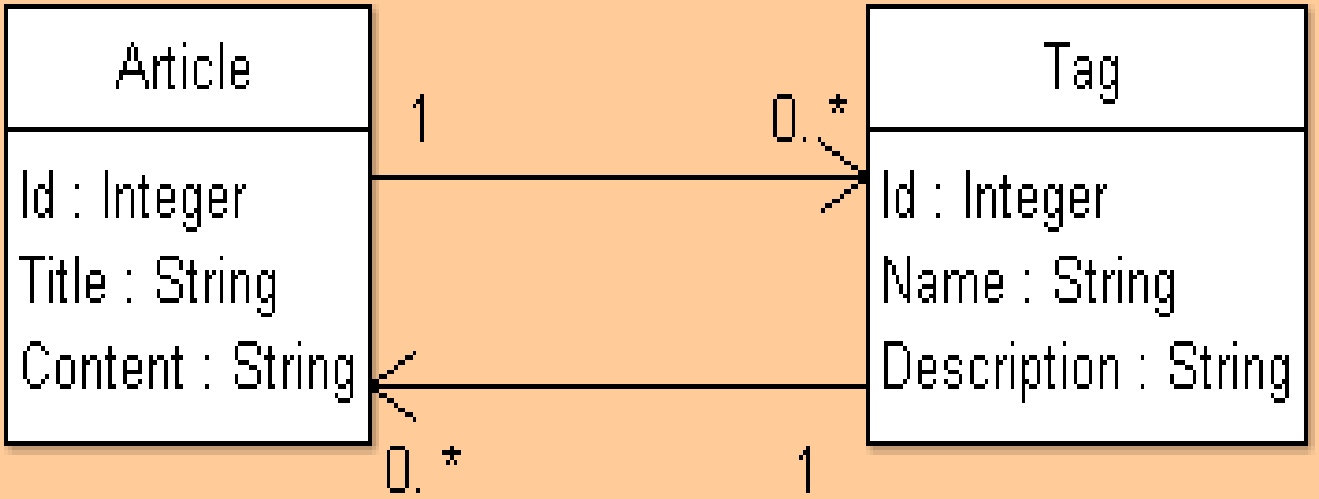
1 relation (0,n) donne 1 liste ou tableau



Les problèmes

Les 2 mondes sont différents :

Relation (n,n) donne deux listes ou 1 container à double entrée.

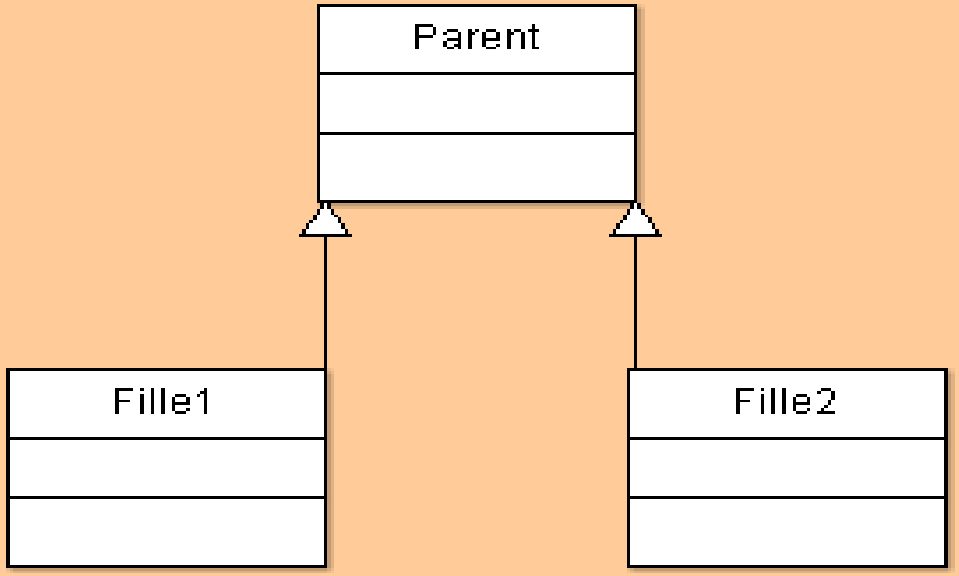


Les problèmes

Les 2 mondes sont différents :

Pas d'héritage en relationnel

=> Solution à 1 ou N tables



Les problèmes

Un début de solution :

```
Créer un nouvel objet

Récupérer les attributs simples

Récupérer les attributs « composés »

Retourner l'objet
```

Les problèmes

Cohérence entre objet en mémoire et contenu base de données :

Un id en BDD donne un et un seul objet en mémoire

Exemple :

```
my $Tag1 = Metier::Tag->retrieve(1);
  ...
my $Tag2 = Metier::Tag->retrieve(1);
```

Les problèmes

Cohérence entre objet en mémoire et contenu base de données :

Si la BDD est partagée par plusieurs applications

Si un objet est détruit

Si un objet est partagé et modifié

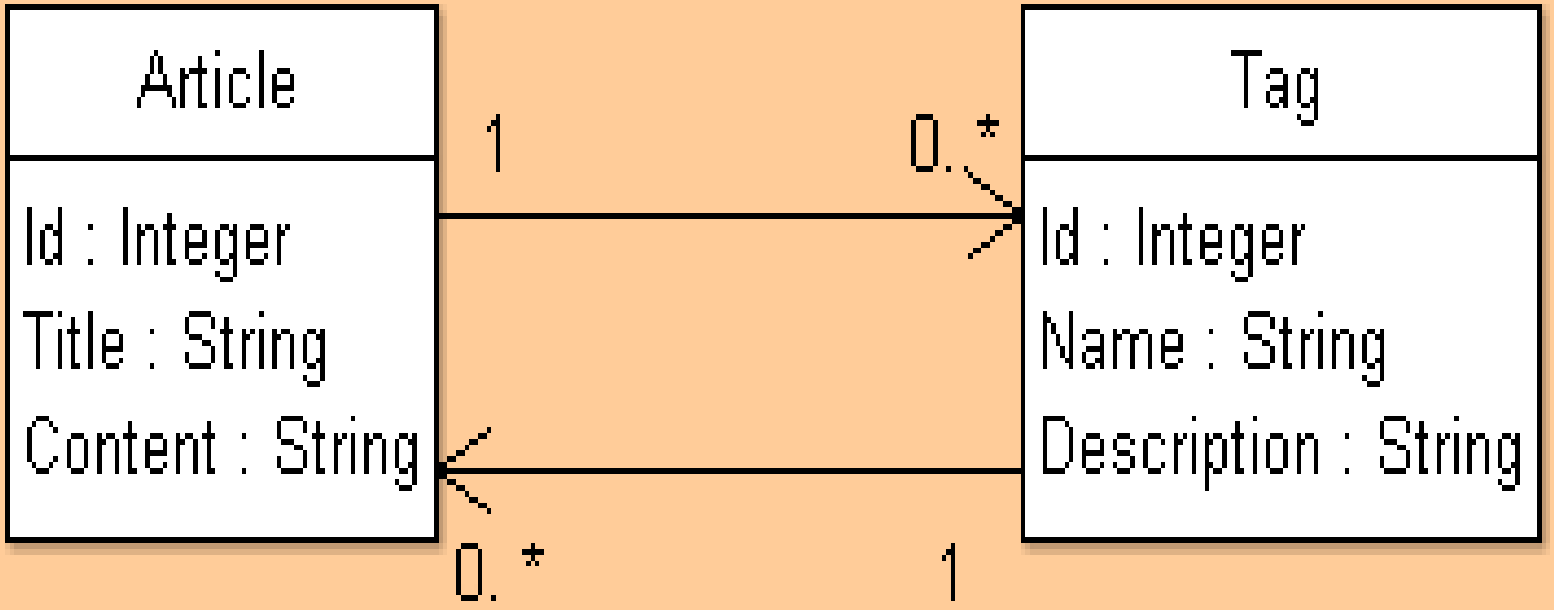
Les problèmes

Le chargement des objets :

Minimiser le nombre de requêtes

Les problèmes

Le chargement des objets :



Les problèmes

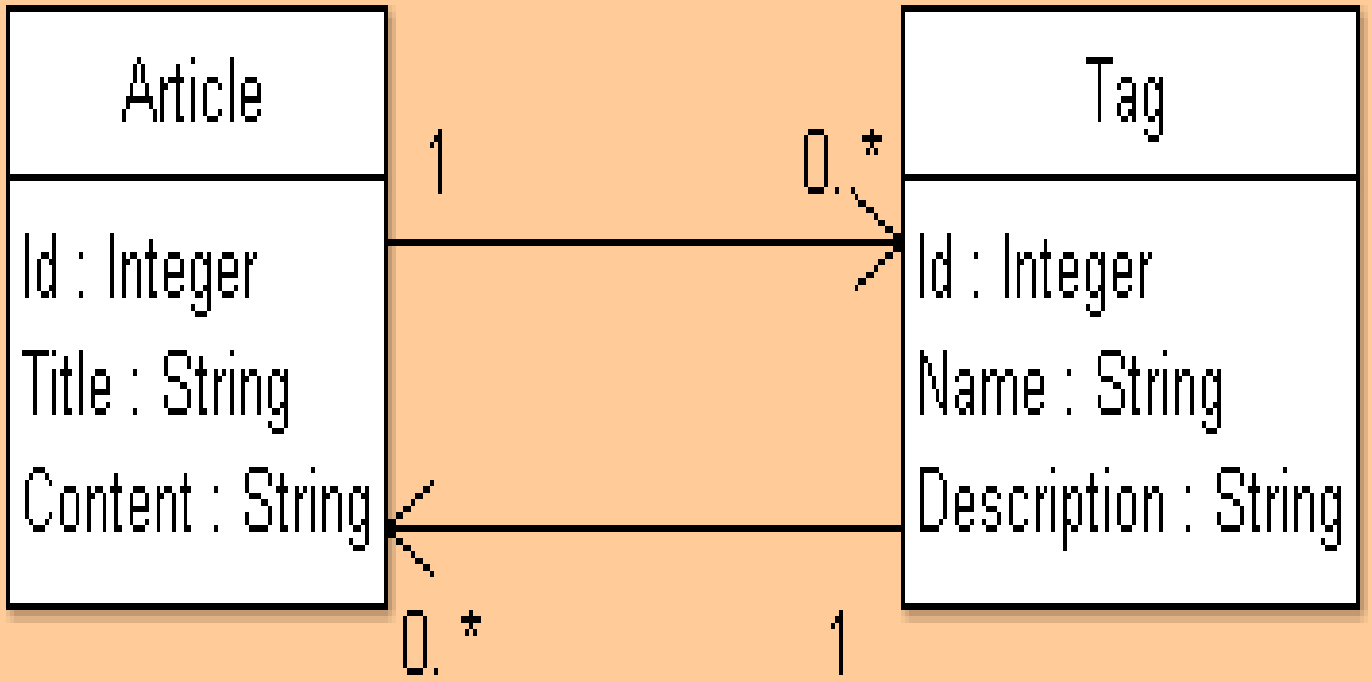
Le chargement des objets :

Lazy fetching (chargement à la demande) et les architectures state less

Chargement complet et architectures avec cache

Les problèmes

Parser un graphe d'objets :



Les avantages

Racine du problème

Les problèmes cités ne dépendent pas directement des ORMs mais des différences entre le monde objet et relationnel

Les avantages

Plus besoin d'écrire de requêtes

MAIS ATTENTION

ELLES EXISTENT TOUJOURS !

Les avantages

Incite aux bonnes pratiques

Toujours mieux que le mélange de code d'extraction / traitement / présentation

Les avantages

Outils libres et partagés

Fruit de l'expérience de plusieurs personnes

Continuent d'évoluer

=> Mieux qu'un outil bricolé pour ses besoins sur un coin de table.

Les avantages

Indépendances

du code d'accès aux données par rapport aux autres couches

(dans certaines limites) par rapport au SGBD utilisé.

ORM libres

Java : Hibernate, ...

.Net : NHibernate, ...

PHP : ezpdo, ...

Perl : Class::DBi, Tamgram, ...

Python : SQLAlchemy, Django ORM

...

Questions ?

Exemples

Le petit blog avec Class::DBI et HTML::Template

Questions ?

Les ORMs

Faire coexister le monde relationnel et objet

L'utilisation d'un ORM est quelque chose de simple, la documentation officielle et les exemples disponibles sur le Net permettent de passer rapidement à la pratique.
Malheureusement sans une connaissance des problèmes sous jacents à l'utilisation d'un ORM les performances s'écroulent rapidement.
La présentation va s'attacher à montrer ces problèmes.

Plan

Pour la pratique

Séparation en couches

Accès aux données

Idée de base

Les problèmes

Les avantages

Exemples

Pour la pratique

Un blog tout simple basé « 2 pages » :

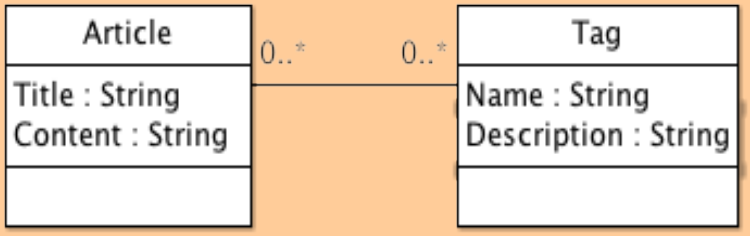
- **une affichera un article**
- **une affichera un tag**

hébergé sur un serveur http

C'est exemple à un gros avantage : tout le monde le comprend car beaucoup de personnes ont utilisé un blog.

Pour la pratique

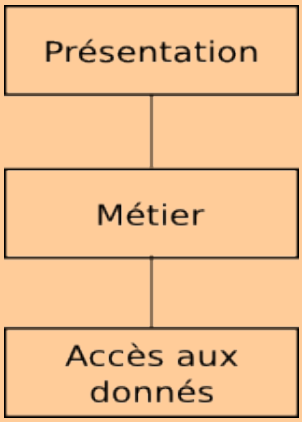
Un blog tout simple



Ce modèle a aussi des inconvenients, la volumetrie d'un blog ne permet pas d'appréhender les problèmes de « monter en charge » des ORMs.
De plus cet exemple est très minimaliste seulement 2 tables (le minimum donc), il ne permet pas de voir les problèmes poser par des modèles comprenant des dizaines voir des centaines de tables.

Séparation en couches

Le modèle à 3 couches



Avantages :

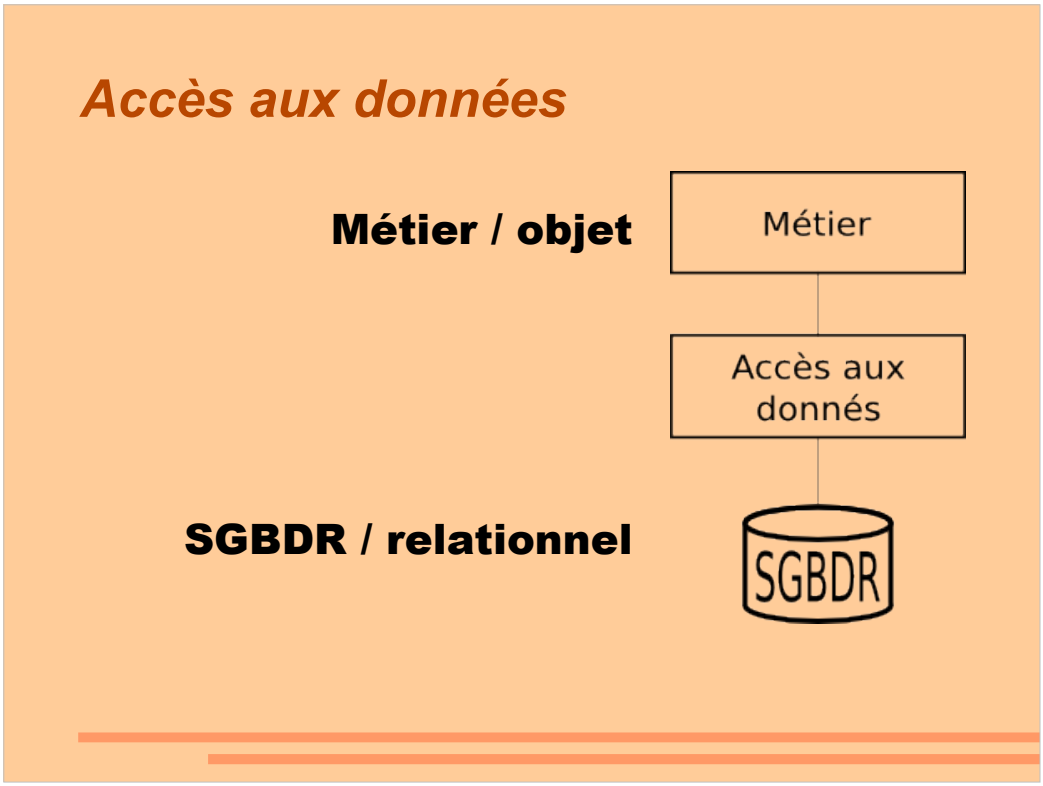
Compréhensibilité

Maintenabilité

Réutilisabilité

...

Pour plus d'informations voir : http://fr.wikipedia.org/wiki/Architecture_trois_tiers



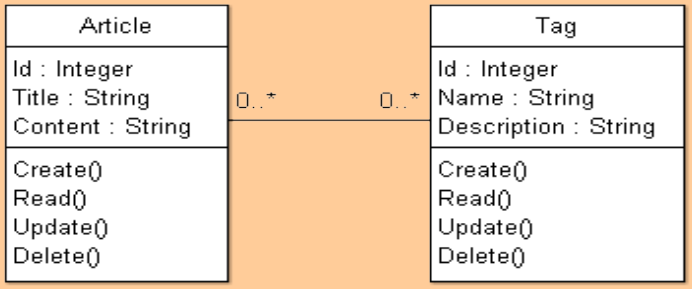
La couche accès aux données peut obtenir des données de différentes sources :

- Lecture dans un fichier
- Entrées/sorties
- Réseau
- ...

Le cas qui nous intéresse dans cette présentation est la communication avec une base de données relationnelle.
http://fr.wikipedia.org/wiki/SGBDR

Idée de base

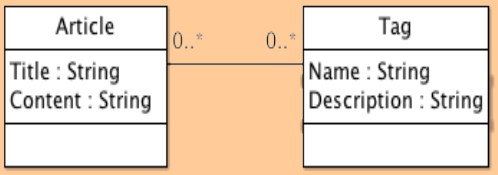
Objets persistants



Pattern CRUD (Create, Read, Update, Delete)

Idée de base

```
-- Structure de la table Articles

CREATE TABLE Article (
  ID int(11) NOT NULL auto_increment,
  Title varchar(100) NOT NULL,
  Content mediumtext NOT NULL,
  PRIMARY KEY  (ID)
);

-- Structure de la table Tags

CREATE TABLE Tag (
  ID int(11) NOT NULL auto_increment,
  Name varchar(100) NOT NULL,
  Description varchar(500) default NULL,
  PRIMARY KEY  (ID)
);

-- Structure de la table TagsArticles

CREATE TABLE TagArticle (
  IdArticle int(11) NOT NULL default '0',
  IdTag int(11) NOT NULL default '0',
  PRIMARY KEY  (IdArticle,IdTag)
);
```



Idée de base

Une classe correspond à une table

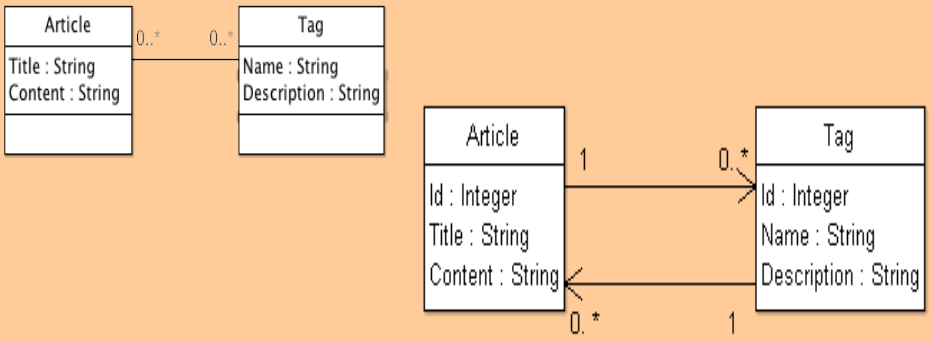
Un attribut correspond à un champ

=> Facile de faire un petit automate qui « fabrique » les requêtes

Les problèmes

Les 2 mondes sont différents :

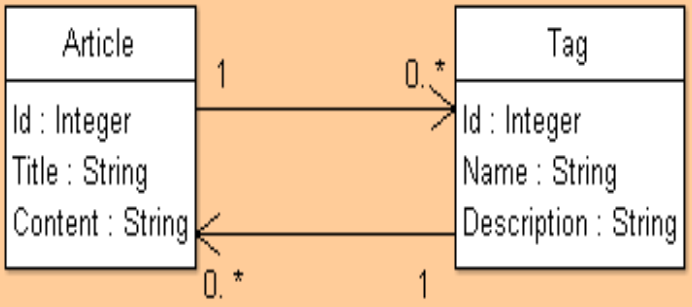
1 relation donne 1 ou 2 références



Les problèmes

Les 2 mondes sont différents :

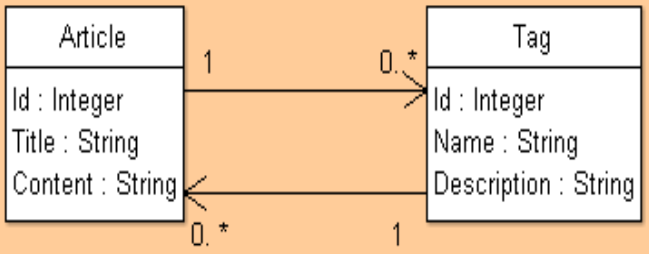
1 relation (0,n) donne 1 liste ou tableau



Les problèmes

Les 2 mondes sont différents :

Relation (n,n) donne deux listes ou 1 container à double entrée.

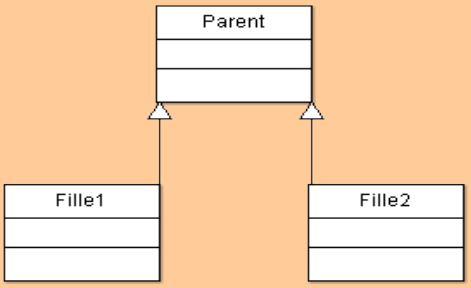


Les problèmes

Les 2 mondes sont différents :

Pas d'héritage en relationnel

=> Solution à 1 ou N tables



Les problèmes

Un début de solution :

```
Créer un nouvel objet

Récupérer les attributs simples

Récupérer les attributs « composés »

Retourner l'objet
```

Les problèmes

Cohérence entre objet en mémoire et contenu base de données :

Un id en BDD donne un et un seul objet en mémoire

Exemple :

```
my $Tag1 = Metier::Tag->retrieve(1);
 ...
my $Tag2 = Metier::Tag->retrieve(1);
```

Les problèmes

Cohérence entre objet en mémoire et contenu base de données :

Si la BDD est partagée par plusieurs applications

Si un objet est détruit

Si un objet est partagé et modifié

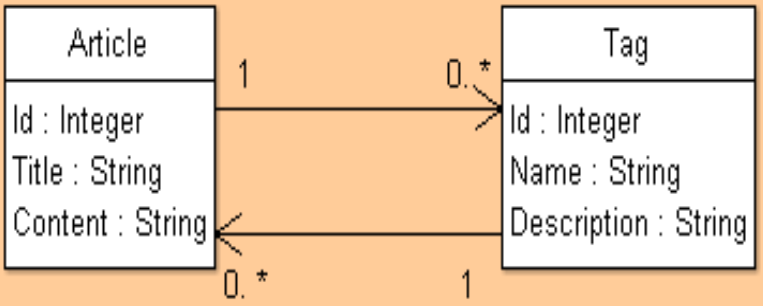
Les problèmes

Le chargement des objets :

Minimiser le nombre de requêtes

Les problèmes

Le chargement des objets :



Les problèmes

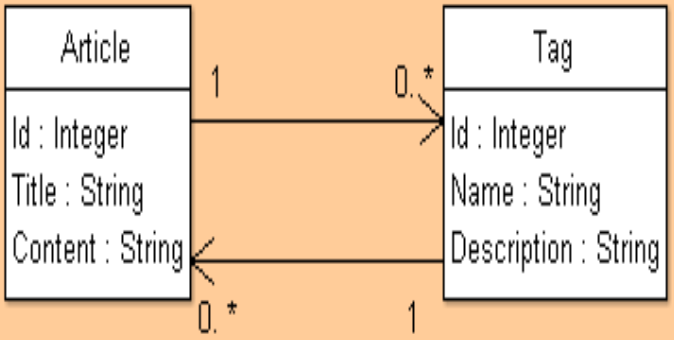
Le chargement des objets :

Lazy fetching (chargement à la demande) et les architectures state less

Chargement complet et architectures avec cache

Les problèmes

Parser un graphe d'objets :



Les avantages

Racine du problème

Les problèmes cités ne dépendent pas directement des ORMs mais des différences entre le monde objet et relationnel

Les avantages

Plus besoin d'écrire de requêtes

MAIS ATTENTION

ELLES EXISTENT TOUJOURS !

Les avantages

- **Incite aux bonnes pratiques**
- **Toujours mieux que le mélange de code d'extraction / traitement / présentation**

Les avantages

Outils libres et partagés

Fruit de l'expérience de plusieurs personnes

Continuent d'évoluer

=> Mieux qu'un outil bricolé pour ses besoins sur un coin de table.

Les avantages

Indépendances

du code d'accès aux données par rapport aux autres couches

(dans certaines limites) par rapport au SGBD utilisé.

ORM libres

Java : Hibernate, ...

.Net : NHibernate, ...

PHP : ezpdo, ...

Perl : Class::DBi, Tamgram, ...

Python : SQLAlchemy, Django ORM

...

Questions ?

Exemples

Le petit blog avec Class::DBI et HTML::Template

Questions ?