

Introduction au passage UML - relationnel

Table des matières

I. Contexte	3
II. Les classes	3
III. Exercice : Appliquez la notion	5
IV. Relations One-to-Many	6
V. Exercice : Appliquez la notion	9
VI. Relations One-to-One	9
VII. Exercice : Appliquez la notion	11
VIII. Relations Many-to-Many	12
IX. Exercice : Appliquez la notion	14
X. L'héritage	14
XI. Exercice : Appliquez la notion	16
XII. Essentiel	16
XIII. Auto-évaluation	16
A. Exercice final	16
B. Exercice : Défi	18
Solutions des exercices	19

I. Contexte

Durée : 1 h

Environnement de travail : Local

Pré-requis : Savoir faire un diagramme de classes

Contexte

Établir un diagramme de classes permet de définir la structure de notre application, mais peut également servir à déterminer les tables de notre bases de données. En effet, grâce à quelques règles simples, il est possible de convertir le diagramme en modèle relationnel.

II. Les classes

Objectif

- Déterminer les tables de notre base de données

Mise en situation

La première étape pour convertir le diagramme de classes en modèle relationnel est d'en extraire les différentes tables. C'est une étape simple, mais qui va servir de base au reste de la structure.

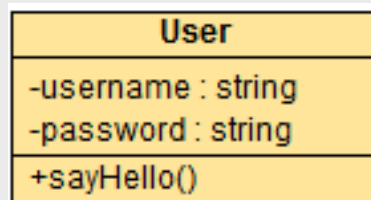
Méthode Une classe = une table

Chaque classe de notre diagramme représente une table qui sera présente dans la base de données.

Le nom de la table issue d'une classe est le nom de la classe mise au pluriel. En effet, une instance représente une ligne de données : en faisant `new Classe()`, on ne crée qu'un seul élément, il est donc logique que le nom de la classe soit au singulier. En revanche, une table contient plusieurs éléments, son nom doit donc être au pluriel. Le format du nom est également différent : les noms de classe sont en `PascalCase`, tandis que les noms de table sont en `snake_case`.

Exemple

Soit la classe `User` définie par le diagramme suivant :



La table associée devra se nommer `users`.

Remarque

Toutes les classes du diagramme ne doivent pas être transformées en bases de données : seules celles représentant des structures de données doivent l'être.

Par exemple, si notre application possède une classe `Product` pour gérer les données de produits à vendre, ainsi que plusieurs classes de formatage pour afficher de manières différentes les informations du produit, seules les données de la classe `Product` devront être stockées en base.

Méthode Une propriété = un champ

Les champs de la table doivent correspondre aux propriétés de la classe. Le type de chaque champ doit également correspondre au type de la propriété. Si le type ne correspond pas, alors il faut déterminer le type le plus proche.

Les règles métiers pourront permettre de déterminer si un champ est obligatoire ou non.

Les méthodes, quant à elles, ne sont pas implémentées dans le modèle relationnel.

Exemple

Notre classe `User` possède deux propriétés de type `string` : `username` et `password`. Ce type n'a pas de correspondance exacte en SQL : nous avons le choix entre `CHAR`, `VARCHAR` et `TEXT`. Il faut donc déterminer le type exact et sa taille grâce à des règles métier et/ou techniques.

- Un `username` a une taille variable : le type `VARCHAR` est donc préférable. La taille maximale devra être déterminée par le design du site et par la place réservée à l'affichage du nom.
- Pour le mot de passe, l'algorithme de cryptage recommandé est `BCRYPT`, qui génère des chaînes de 60 caractères. Le champ `password` sera donc un `CHAR(60)`.

Ces deux champs seront obligatoires à la saisie.

Méthode La clé primaire

La dernière étape est de déterminer la clé primaire de notre table à partir des propriétés de notre classe. Pour cela, deux solutions existent :

- Déterminer une **clé primaire naturelle**, c'est-à-dire une clé primaire composée uniquement des données présentes dans la classe. Si la classe possédait déjà une information unique et qui ne sera jamais modifiée, alors elle pourra être utilisée comme clé primaire.
- Créer une nouvelle colonne qui accueillera une **clé primaire artificielle**, soit auto-incrémentée, soit de type `UUID`. Dans ce cas, il faudra également rajouter la nouvelle colonne dans la classe.

Conseil

Il est très souvent préférable d'avoir une clé artificielle. Cet identifiant est généralement réfléchi en amont et donc déjà inclus dans le diagramme de classes, mais il ne faut pas oublier de le rajouter dans le cas contraire.

Dans la grande majorité des cas vous retrouverez cet identifiant sous le nom `id`.

Exemple

La requête permettant de créer la table représentant la classe `Users` est :

```
1 CREATE TABLE users (
2     id INT(11) AUTO_INCREMENT PRIMARY KEY,
3     username VARCHAR(50) NOT NULL,
4     password CHAR(60) NOT NULL
5 );
```

La commande `AUTO_INCREMENT` permet d'incrémenter automatiquement l'id à chaque ajout d'un utilisateur en base de données sans action de notre part, ce qui permet de garantir l'unicité de notre clé primaire.

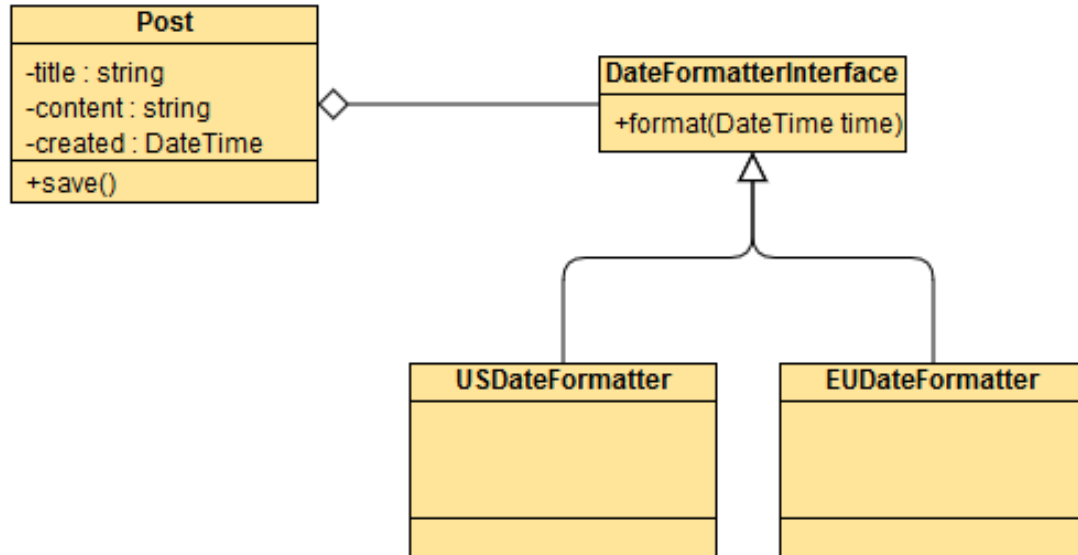
Syntaxe **À retenir**

- Une classe dans le diagramme UML donne une table dans la base de données.
- Chaque propriété donne un champ de la table.
- Les méthodes ne sont pas représentées.
- Il faut également déterminer (ou ajouter) une clé primaire.

III. Exercice : Appliquez la notion**Question**

[solution n°1 p.21]

Le Tech Lead de votre équipe a créé un diagramme de classes pour gérer et afficher des articles dans une application de blog ses articles seront stockés en base de données :



Écrivez les requêtes nécessaires à la création du modèle relationnel correspondant.

Indice :

Toutes les tables ne devront pas apparaître dans le modèle relationnel.

Indice :

Les informations de la classe `Post` ne suffisent pas à déterminer une clé primaire.

IV. Relations One-to-Many

Objectifs

- Comprendre la notion de « One-to-Many »
- Déterminer l'impact de cette cardinalité sur le modèle relationnel

Mise en situation

Une fois les tables déterminées, il faut convertir les associations du diagramme de classes en relations et contraintes dans le modèle relationnel. Pour cela, il faut se baser sur la cardinalité des associations. Il existe trois cas possibles : **One-To-Many**, **One-To-One** et **Many-To-Many**.

Définition One-To-Many

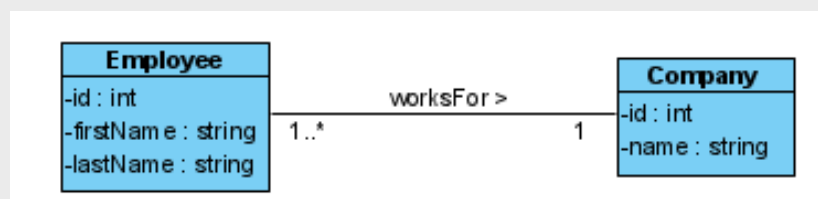
Une relation One-To-Many est une association qui a une cardinalité unique d'un côté (donc 1 ou 0..1) et une cardinalité multiple de l'autre (0..*, 1..*). C'est l'association la plus courante.

0...1	Aucune ou une instance
1	Une instance exactement

0..* ou *	Aucune ou plusieurs instances
1..*	Une instance ou plusieurs (au moins une)

Exemple

La relation entre une entreprise et ses employés peut être définie par une relation One-To-Many :



Un employé travaille pour une seule (one) entreprise, on ajoute donc la cardinalité représentant une instance exactement du côté entreprise => 1, chaque entreprise possède un ou plusieurs (many) employés, on ajoute donc la cardinalité représentant une instance ou plusieurs côté employé => 1..*

On se trouve bien dans une relation cardinalité unique vers multiple donc One-To-Many.

Remarque

Une relation One-To-Many peut également être appelée Many-To-One : cela dépend du sens de lecture. Ainsi, une entreprise a une relation One-To-Many avec ses employés, mais un employé a une relation Many-To-One avec une entreprise.

Cette subtilité n'a aucune incidence sur le modèle relationnel : dans le reste de ce chapitre, nous n'utiliserons que le terme de One-To-Many.

Méthode Les clés étrangères

La clé étrangère va nous permettre de créer cette relation entre les deux tables dans notre base de données. Pour cela, il faut créer un champ dans la table du côté « Many » de l'association, qui sera une clé étrangère vers la table du côté « One ».

Exemple

Dans notre exemple précédent, on a pu mettre en évidence qu'un employé travaillait pour une seule compagnie, on va donc ajouter un champ dans notre table employé pour nous permettre d'identifier cette compagnie.

Le diagramme de classes nous a permis de créer une table `companies` et une table `employees`. Pour gérer l'association One-To-Many entre les deux, il faut rajouter un champ `works_for` dans la table `employees`, qui sera une clé étrangère vers la table `companies` :

```
1 CREATE TABLE companies (  
2     id INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,  
3     name VARCHAR(50) NOT NULL  
4 );  
5  
6 CREATE TABLE employees (  
7     id INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,  
8     first_name VARCHAR(50) NOT NULL,  
9     last_name VARCHAR(50) NOT NULL,  
10    works_for INT(11) NOT NULL,  
11    FOREIGN KEY (works_for) REFERENCES companies(id)  
12 );
```

Maintenant notre table employé contient l'id d'une entreprise stocké dans le champ `work_for`, cette id nous permet de retrouver la société ayant le même id dans notre base de données et donc d'accéder à son nom par exemple.

Remarque Composition et agrégations

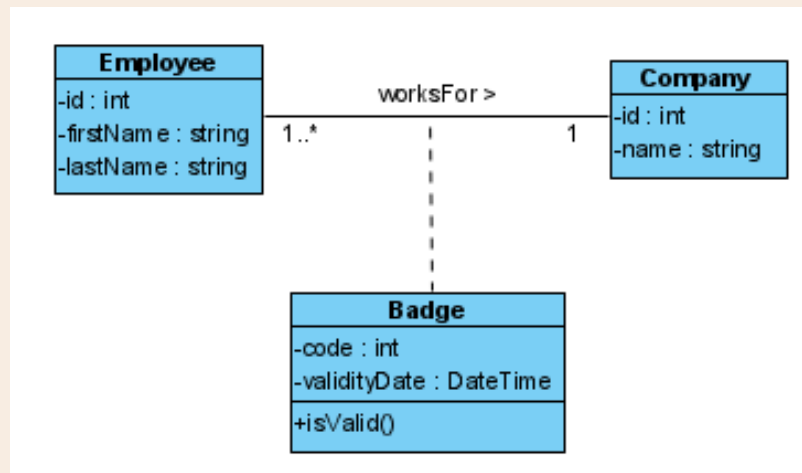
L'agrégation et la composition sont deux types d'associations particulières, et le choix de l'une ou l'autre peut avoir une incidence sur le modèle relationnel. En effet, si l'agrégation est à considérer comme une association normale, une subtilité existe avec la composition : la clé étrangère générée doit faire partie de la clé primaire. Cela s'explique par le fait que la composition exprime une dépendance forte entre les deux éléments : l'un ne peut pas exister si l'autre est détruit.

Cette dépendance peut également être gérée grâce à l'instruction SQL `CASCADE`¹, qui permet de définir l'impact de la suppression d'une ligne sur toutes ses relations. En cas de composition, la suppression d'une ligne de la table cible devra entraîner la suppression de toutes ses compositions.

¹ <https://dev.mysql.com/doc/refman/8.0/en/create-table-foreign-keys.html>

Complément	Le cas particulier de la classe d'association
-------------------	--

Dans le cas d'une classe d'association, les champs de cette classe doivent être rajoutés du côté « Many ». Par exemple, rajoutons un badge à notre employé :



Dans ce cas, les propriétés `code` et `validityDate` donneront deux nouveaux champs dans la table `employees`.

Méthode Impact des cardinalités nulles

Pour chaque côté de l'association, il y a deux choix de cardinalités possibles : soit la valeur 0 est admise (0..1 et 0..*), soit elle ne l'est pas (1 et 1..*).

- Une cardinalité nulle du côté **Many** (0..*) n'a aucune influence sur le modèle relationnel : on aura simplement une clé primaire qui ne sera pas référencée en tant que clé étrangère. Par exemple, si une entreprise possède une cardinalité 0..* vers ses employés, alors il pourra exister une entreprise pour laquelle aucun employé n'est stocké en base.
- Une cardinalité nulle du côté **One** (0..1), en revanche, indique que la clé étrangère peut être nulle, et qu'il faut donc modifier cette contrainte au moment de la création de la table. C'est le cas, par exemple, si un employé peut ne pas être rattaché à une entreprise : la valeur de `works_for` sera nulle.

Example

Si on remplace les cardinalités de notre exemple par leur équivalent nul, alors il faut préciser que la clé étrangère de la table `employees` peut ne pas être définie :

```
1 CREATE TABLE companies (
2     id INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
3     name VARCHAR(50) NOT NULL
4 );
5
6 CREATE TABLE employees (
7     id INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
8     first_name VARCHAR(50) NOT NULL,
9     last_name VARCHAR(50) NOT NULL,
10    works_for INT(11) NULL, -- La clé étrangère peut être nulle
11    FOREIGN KEY (works_for) REFERENCES companies(id)
12 );
```


Syntaxe **À retenir**

- Une association One-To-Many permet de définir la clé étrangère du côté Many.
- Une cardinalité nulle côté One indique qu'elle peut recevoir la valeur NULL.

V. Exercice : Appliquez la notion**Question**

[solution n°2 p.21]

Une association organisant des expositions de peinture souhaite gérer informatiquement les œuvres à exposer. Après analyse, votre Tech Lead vous donne le diagramme de classes suivant :



Écrivez les requêtes nécessaires à la création du modèle relationnel correspondant.

Indice :

Attention à l'ordre de création des tables : celle référençant la clé étrangère doit être créée après, sinon il faut `ALTER TABLE` pour rajouter la clé.

VI. Relations One-to-One**Objectifs**

- Comprendre la notion de « One-to-One »
- Déterminer l'impact de cette cardinalité sur le modèle relationnel

Mise en situation

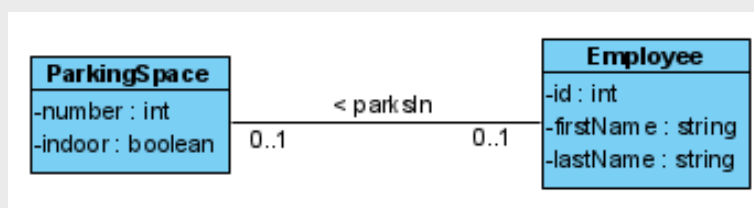
Le deuxième cas de relation possible est le **One-To-One**. Ce cas est particulier puisqu'il existe deux interprétations possibles lors de la création du modèle relationnel. Nous allons voir quelles sont ces deux méthodes et dans quels cas il faut les utiliser.

Définition **One-To-One**

Une relation One-To-One est une association qui a une cardinalité unique (0..1 ou 1) des deux côtés de l'association.

Exemple

À chaque employé est assignée une place de parking. Certains employés peuvent ne pas en vouloir, et certaines places peuvent être vacantes :



Méthode La méthode One-To-Many

La première solution est de traiter les One-To-One de la même manière que des One-To-Many : en choisissant une table qui recevra une clé étrangère vers l'autre. Le choix de la table est arbitraire et dépend de la sémantique des éléments. Ainsi, dans l'exemple ci-dessus, il appartiendrait au développeur de déterminer si un employé possède une place de parking, ou si une place de parking contient un employé.

Exemple

Si notre application gère en premier lieu des employés, alors il est plus logique qu'un employé possède une place de parking :

```
1 CREATE TABLE parking_space(
2     number INT(11) NOT NULL PRIMARY KEY,
3     indoor TINYINT(1) NOT NULL -- Le type "boolean" n'est pas disponible sur tous les SGBD et
    est souvent remplacé par un TINYINT ou une énumération
4 );
5
6 CREATE TABLE employees (
7     id INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
8     first_name VARCHAR(50) NOT NULL,
9     last_name VARCHAR(50) NOT NULL,
10    parks_in INT(11) NULL,
11    FOREIGN KEY (parks_in) REFERENCES parking_space(number)
12 );
```

À l'inverse, si le point central de notre application est de gérer des places de parking, alors la clé étrangère doit être dans la table `parking_space`.

Méthode La méthode de fusion

Une autre méthode consiste en la fusion de toutes les propriétés des deux classes en une seule table. La clé primaire de cette table dépend des informations des tables :

- Si aucune clé naturelle ne peut être trouvée, alors une clé artificielle doit être créée.
- Si une clé naturelle est trouvée dans une table, mais pas dans l'autre, alors elle peut être utilisée comme clé primaire.
- Si chaque table possède une clé naturelle, alors l'une d'entre elles est désignée arbitrairement comme clé primaire, et l'autre devra avoir une contrainte d'unicité.

Exemple

Dans notre exemple, la table `parking_space` possède un identifiant naturel : un numéro de place. Il est donc possible de retirer la clé artificielle `id` de la table `employees` dans la table fusionnée :

```
1 CREATE TABLE employees_parking_space(
2     number INT(11) NOT NULL PRIMARY KEY,
3     indoor TINYINT(1) NOT NULL,
4     first_name VARCHAR(50) NOT NULL,
5     last_name VARCHAR(50) NOT NULL
6 );
```

Conseil

Attention aux noms des propriétés au moment de la fusion : il ne faut pas que deux colonnes aient le même nom.

Méthode Quelle méthode choisir ?

Il existe plusieurs critères permettant de choisir l'une ou l'autre des méthodes :

- Si l'une des deux tables possède des cardinalités nulles, alors il est préférable de garder les deux tables en utilisant la méthode One-To-Many. En effet, l'exemple de fusion ci-dessus possède un inconvénient majeur : il est spécifié que certains employés peuvent ne pas avoir de place de parking. Or, le numéro de place étant la clé primaire, les employés n'ayant pas de place ne pourront pas être insérés dans la base. De plus, même avec une clé primaire artificielle, fusionner ces informations dans la même table signifie que de nombreuses colonnes seront à NULL.
- La sémantique et la facilité de manipulation des informations est également importante : il faut juger de la pertinence de la fusion des informations. Si une classe `Person` a une relation One-To-One avec une classe `Address`, la fusion peut sembler pertinente. En revanche, si chaque `Person` a une relation One-To-One avec une mère et un père (deux autres `Person`), alors il est préférable de ne pas rajouter les informations des parents dans la table `Persons`, mais plutôt de rajouter des clés étrangères.

Chaque cas est différent et il n'existe pas de solution qui fonctionne à tous les coups. Il faut prendre le temps de traiter chaque association et de déterminer la méthode qui sera la plus pratique à utiliser au moment du développement.

Syntaxe À retenir

- Une association One-To-One peut être résolue de deux manières différentes : soit en la considérant comme une One-To-Many, soit en fusionnant les informations des deux tables dans une seule.
- Le choix de la méthode dépend des cardinalités, mais aussi de la sémantique de la fusion et de la facilité de manipulation des données qu'elle permettrait.

VII. Exercice : Appliquez la notion**Question**

[solution n°3 p.21]

Un organisateur de courses de moto souhaite gérer les informations des participants et de leurs motos pendant la compétition.

Un participant possède un numéro à deux chiffres inscrit sur l'avant de sa moto, qu'il est le seul à avoir, ainsi qu'un nom et un prénom. Une moto possède une marque et une vitesse maximale.

Chaque participant va conduire la même moto pour toute la durée de la compétition, et chaque moto ne sera conduite que par son conducteur attitré, ce qui donne le diagramme de classes suivant :



Écrivez les requêtes nécessaires à la création du modèle relationnel correspondant.

Indice :

Avec une relation sans cardinalités nulles, on peut se poser la question de la pertinence de la fusion de ces informations.

Indice :

Il existe une clé naturelle du côté `Racer`, mais pas du côté `Moto` : une fusion de données est toute indiquée.

VIII. Relations Many-to-Many

Objectifs

- Comprendre la notion de « Many-to-Many »
- Déterminer l'impact de cette cardinalité sur le modèle relationnel

Mise en situation

Le troisième type de relation possible est le **Many-To-Many**. C'est une relation complexe qui ne peut pas être résolue par le simple ajout de clés étrangères : elle nécessite la création de tables dédiées.

Définition **Many-To-Many**

Une relation Many-To-Many est une association qui a une cardinalité multiple (donc 0..* ou 1..*) des deux côtés de l'association.

Exemple

Sur un site de vente en ligne, un utilisateur peut mettre plusieurs articles dans son panier et un article peut être présent dans le panier de plusieurs utilisateurs :



Méthode **Les tables associatives**

Pour résoudre cette association, ajouter une clé étrangère dans une des tables ne suffit pas : une clé étrangère seule permet de ne lier qu'une ligne de données à une autre. Il va falloir créer une table d'association, c'est-à-dire une table où chaque ligne de données sera une association entre les tables.

Une table associative possède une clé étrangère par table, qu'elle lie, et sa clé primaire est définie par l'ensemble de ces clés étrangères : cela permet de s'assurer qu'une association n'apparaîtra qu'une seule fois.

Rappel

Pour déclarer une clé primaire portant sur plusieurs colonnes dans une table, il faut utiliser la clause `PRIMARY KEY([liste des colonnes])` à la fin de la requête `CREATE TABLE`, ou faire un `ALTER TABLE` après sa création.

Exemple

Pour représenter les utilisateurs et leurs paniers, nous allons avoir besoin de trois tables : `users`, `products` et une table associative entre les deux. Étant donné qu'elle représente le contenu de chaque panier, nous pouvons l'appeler `carts` :

```

1 CREATE TABLE users (
2     id INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
3     username VARCHAR(50) NOT NULL,
4     password CHAR(60) NOT NULL
5 );
6
7 CREATE TABLE products (
8     id INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
9     name VARCHAR(50) NOT NULL
10 );
11
12 -- Table associative
13 CREATE TABLE carts (
14     user_id INT(11) NOT NULL,
15     product_id INT(11) NOT NULL,
16     PRIMARY KEY (user_id, product_id),
17     FOREIGN KEY (user_id) REFERENCES users(id),
18     FOREIGN KEY (product_id) REFERENCES products(id)
19 );

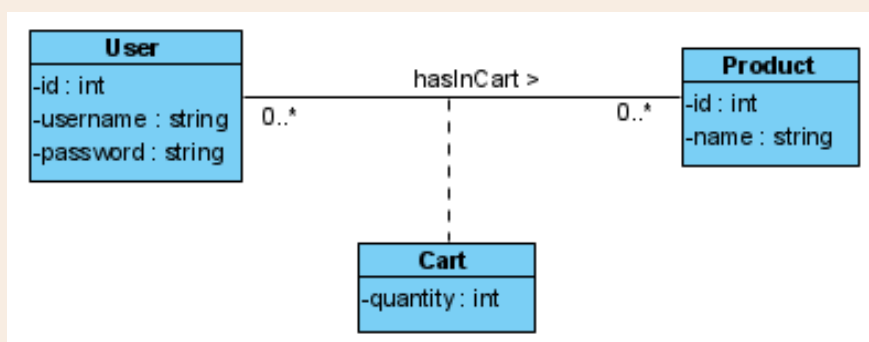
```

Remarque

Les cardinalités nulles n'ont aucun impact sur le modèle relationnel. Elles signifient seulement que certains éléments n'auront aucune correspondance dans la table d'association.

Complément Le cas particulier de la classe d'association

Dans le cas d'une classe d'association, les champs de cette classe doivent être rajoutés dans la table associative. Par exemple, si nous souhaitons ajouter, pour chaque utilisateur, la quantité du produit qui a été ajouté dans le panier, comme ceci :



Alors la propriété `quantity` sera un champ de la table `carts`.

Syntaxe À retenir

- Pour gérer les relations Many-To-Many, il faut créer une table d'association. Les champs de cette table sont les clés étrangères vers les tables de l'association et la clé primaire est composée de toutes ces clés étrangères.
- Dans le cas d'une classe d'association, ses champs sont ajoutés dans la table d'association.

IX. Exercice : Appliquez la notion

Question

[solution n°4 p.21]

Un vétérinaire souhaite informatiser la gestion de ses clients et de leurs animaux. Un client peut posséder plusieurs animaux, et un même animal peut appartenir à plusieurs clients (généralement, des membres différents d'une même famille) :



Écrivez les requêtes nécessaires à la création du modèle relationnel correspondant.

X. L'héritage

Objectifs

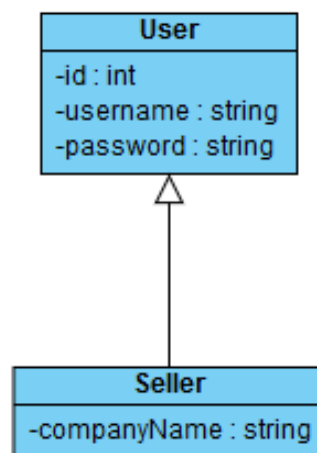
- Comprendre les impacts de l'héritage sur le modèle relationnel
- Distinguer une relation d'héritage avec une relation One-To-Many

Mise en situation

Dans le paradigme orienté objet, l'héritage est quelque chose de très particulier : ce n'est pas une association à proprement parler, mais plus une relation de hiérarchie entre deux classes. Cependant, il reste possible de les intégrer dans un modèle relationnel, mais la logique derrière cette intégration est plus complexe que pour les autres associations.

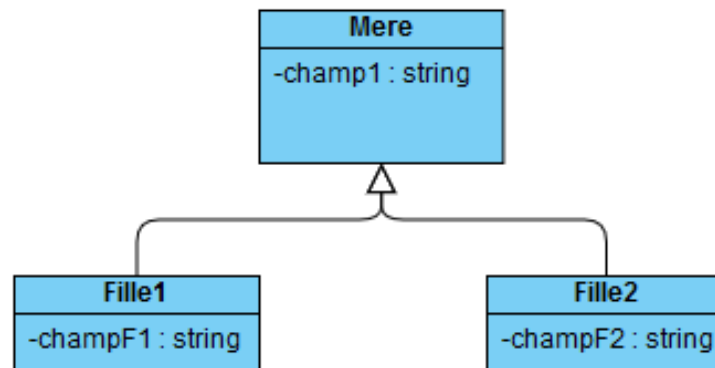
L'héritage : quelle association ?

Pour illustrer cette logique, prenons l'exemple d'un site de vente en ligne sur lequel nous allons distinguer deux types d'utilisateurs : les utilisateurs de base et les vendeurs. Un utilisateur de base possède des informations qui lui sont propres (id, login et mot de passe) et peut acheter des articles. Un vendeur **est** un utilisateur de base, mais il dispose également d'une information en plus : le nom de la société qu'il représente. La classe représentant les vendeurs est donc la classe fille de la classe des utilisateurs de base.



En réfléchissant en termes d'association entre les `Sellers` et les `Users`, on peut en déduire qu'un héritage est une relation One-To-One entre une classe fille et une classe mère. En effet, si on crée un vendeur appelé **John** et représentant la société **ACME Inc.**, on peut en réalité le voir comme deux entités liées entre elles : d'un côté le `User` John et de l'autre le `Sellers` de ACME Inc.. Or, **John** n'a accès qu'aux informations du vendeur de **ACME Inc.**, et inversement : nous sommes donc dans le cas d'une relation One-To-One.

Comme nous l'avons vu, une relation One-To-One peut être résolue de deux manières différentes : soit par fusion des propriétés en une seule table, soit en les traitant comme un One-To-Many. Or, dans le cas d'un héritage, la fusion n'est pas pratique à manipuler lorsqu'une classe possède plusieurs classes filles. Prenons l'exemple ci-dessous :



En utilisant la méthode de fusion des propriétés, le résultat sera une table possédant trois champs : `champ1`, `champF1` et `champF2`. Or, dans la pratique, un objet sera soit une instance de `Mere`, soit de `Fille1`, soit de `Fille2`, donc les champs correspondant aux autres classes filles seront forcément nuls, ce qui n'est pas optimal. Il faut donc traiter l'héritage avec la méthode One-To-Many.

Méthode Un One-To-Many particulier

Comme pour un One-To-Many, il va falloir créer deux tables : une représentant la classe fille et une autre la classe mère ; puis ajouter une clé étrangère dans la table fille vers la table mère.

Cependant, cette solution seule ne permet pas de représenter correctement l'héritage : en l'état, il serait possible de lier les données d'une ligne de table mère à plusieurs lignes de la classe fille. Pour éviter cela, et garder le caractère One-To-One de l'héritage, la clé primaire de la classe fille doit être la clé étrangère. Ainsi, pour une ligne de la classe mère, il ne pourra y avoir qu'une seule ligne de la classe fille correspondante.

Exemple

Si nous souhaitons créer le modèle relationnel du site de vente en ligne donné ci-dessus, alors nous allons devoir créer une table `users` et une table `sellers`, puis rajouter une clé étrangère dans `sellers`, qui sera également la clé primaire de cette table.

```
1 CREATE TABLE users (
2     id INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
3     username VARCHAR(50) NOT NULL,
4     password CHAR(60) NOT NULL
5 );
6
7 CREATE TABLE sellers (
8     user_id INT(11) NOT NULL PRIMARY KEY, - Clef primaire
9     company_name VARCHAR(50) NOT NULL,
10 FOREIGN KEY (user_id) REFERENCES users(id)
11 );
```

Syntaxe À retenir

- L'héritage doit être traité comme un One-To-Many, mais la clé étrangère de la table fille vers la table mère doit également être la clé primaire de la classe fille.

XI. Exercice : Appliquez la notion

Question

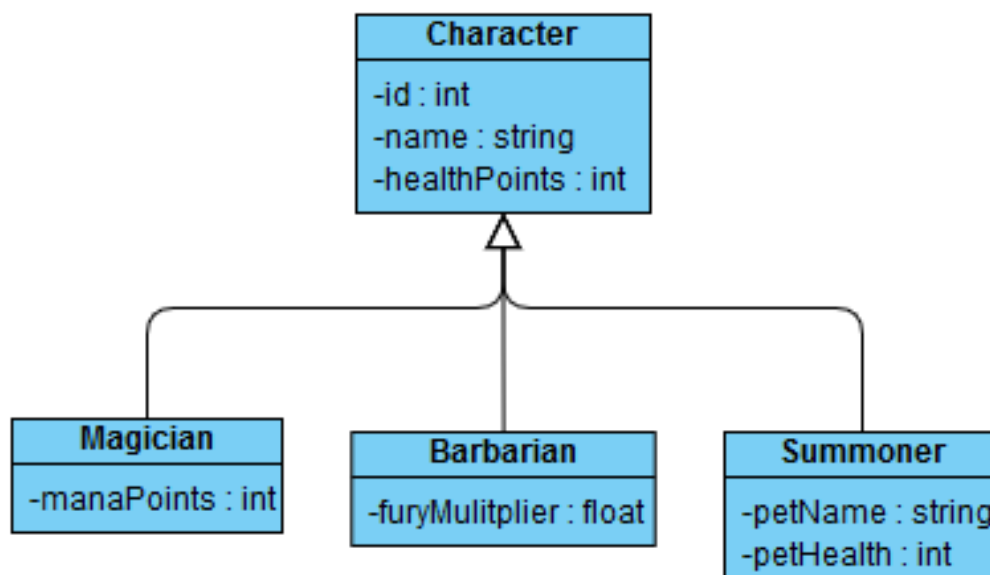
[solution n°5 p.22]

Une association étudiante de jeux de rôles souhaiterait une application permettant de gérer les personnages de leurs joueurs.

Tous les personnages ont un nom et un nombre de points de vie, mais sont déclinés en trois types :

- Les magiciens, qui ont besoin de points de mana pour lancer leur sorts.
- Les barbares, qui peuvent activer une furie leur permettant de multiplier leurs dégâts.
- Les invocateurs, qui combattent avec leur animal de compagnie invoqué, qui dispose de son propre nom et de ses propres points de vie.

La modélisation UML de cette application est la suivante :



Écrivez les requêtes nécessaires à la création du modèle relationnel correspondant.

XII. Essentiel

XIII. Auto-évaluation

A. Exercice final

Exercice 1

[solution n°6 p.22]

Exercice

Parmi ces noms de tables, lequel respecte les bonnes pratiques ?

- ☐ User
- ☐ user
- ☐ Users
- ☐ users

Exercice

Parmi ces noms de champs de table, lequel respecte les bonnes pratiques ?

- ☐ FirstName
- ☐ firstName
- ☐ first_name
- ☐ first-name

Exercice

Comment modéliser les méthodes dans le modèle relationnel ?

- ☐ Grâce à des triggers
- ☐ Grâce à des procédures stockées
- ☐ Grâce à des programmes SQL
- ☐ Les méthodes ne sont pas représentées dans le modèle relationnel

Exercice

Une association avec une cardinalité 0..* d'un côté et 0..1 de l'autre, est une relation...

- ☐ One-to-One
- ☐ One-to-Many
- ☐ Many-to-Many

Exercice

Comment transformer une relation One-To-Many en modèle relationnel ?

- ☐ En créant deux tables, avec une clé étrangère dans la table `Many` vers la table `One`
- ☐ En créant deux tables, avec une clé étrangère dans la table `One` vers la table `Many`
- ☐ En créant trois tables : une `One`, une `Many` et une table d'association
- ☐ En fusionnant les données dans une seule table

Exercice

Dans une relation One-To-Many, quel est l'impact d'une cardinalité nulle côté `Many` sur le modèle relationnel ?

- ☐ La clé étrangère peut être `NULL`
- ☐ La clé étrangère doit faire partie de la clé primaire
- ☐ Elle n'a pas d'impact

Exercice

Laquelle de ces options n'est **pas** valable pour transformer une relation One-To-One en modèle relationnel ?

- ☐ Créer deux tables, avec une clé étrangère de l'une vers l'autre
- ☐ Fusionner toutes les données en une seule table
- ☐ Créer trois tables : deux tables correspondant aux classes et une table d'association

Exercice

Comment choisir parmi les méthodes de transformation des relations One-To-One ?

- ☐ Selon la présence de cardinalités nulles
- ☐ Selon le nombre de propriétés des classes
- ☐ Selon la pertinence d'une fusion

Exercice

Quelle est la clé primaire d'une table d'association ?

- ☐ Un identifiant auto-incrémenté
- ☐ Un UUID
- ☐ L'ensemble de ses clés étrangères

Exercice

Quelle est la différence entre une relation One-To-Many et un héritage dans le modèle relationnel ?

- ☐ Dans l'héritage, la clé étrangère est également la clé primaire de la table
- ☐ Dans l'héritage, la clé étrangère peut être nulle
- ☐ Dans l'héritage, la table fille ne possède pas de clé primaire

B. Exercice : Défi

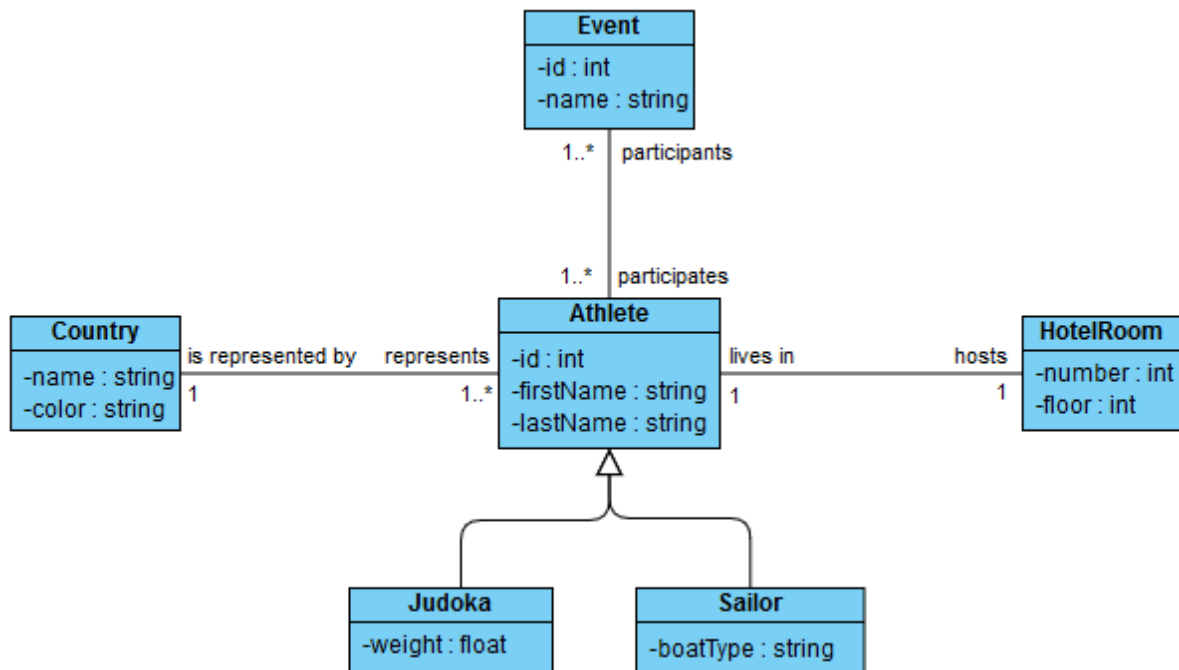
Vous avez décroché un contrat pour créer un logiciel permettant de gérer les prochains Jeux Olympiques ! Utilisez vos connaissances pour modéliser un modèle relationnel à partir du MCD donné afin que les Jeux se déroulent dans les meilleures conditions.

Question

[solution n°7 p.25]

Le Comité International Olympique aimerait pouvoir gérer le déroulement des prochains Jeux Olympiques. Pour cela, ils ont besoin d'une application permettant de stocker les **informations des sportifs** : leurs **épreuves** (un athlète peut participer à plusieurs épreuves), le **pays** qu'ils représentent, et certaines **spécificités** selon leur sport de prédilection. Pour faciliter l'organisation, ils aimeraient aussi pouvoir attribuer une **chambre d'hôtel** à chaque athlète (dont les numéros sont uniques dans tout l'hôtel) afin de les retrouver plus facilement en cas de problème.

Votre architecte logiciel a déduit le MCD suivant de ses échanges avec le comité :



Créez le modèle relationnel correspondant, en rajoutant des colonnes au besoin.

Indice :

Les athlètes ont une association One-To-One avec les chambres d'hôtel. Il faut donc réfléchir à la pertinence d'une fusion des informations.

Indice :

Attention à l'ordre de création des tables : il ne faut pas créer une table possédant une clé étrangère avant d'avoir créé la table contenant la clé primaire correspondante.

Indice :

Il n'est pas possible de définir une clé primaire portant sur plusieurs colonnes au niveau de chaque colonne. Il faut préciser l'instruction `PRIMARY KEY([liste des colonnes])` à la fin de la table.

Solutions des exercices

p. 5 Solution n°1

Parmi toutes les classes du diagramme, seuls les articles devront être stockés en base.

Le titre est un VARCHAR et le contenu peut être, au choix, un VARCHAR ou un TEXT.

En plus des informations de la classe, il faut également rajouter une colonne `id` comme clé primaire.

```
1 CREATE TABLE posts (
2   id INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
3   title VARCHAR(100) NOT NULL,
4   content TEXT NOT NULL,
5   created DATETIME NOT NULL
6 );
```

p. 9 Solution n°2

```
1 CREATE TABLE artists (
2   id INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
3   first_name VARCHAR(50) NOT NULL,
4   last_name VARCHAR(50) NOT NULL,
5   birth_date DATETIME NOT NULL
6 );
7
8 CREATE TABLE paintings (
9   id INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
10  title VARCHAR(50) NOT NULL,
11  completion_date DATETIME NOT NULL,
12  is_painted_by INT(11) NOT NULL,
13  FOREIGN KEY (is_painted_by) REFERENCES artists(id)
14 );
```

p. 11 Solution n°3

Le numéro est une bonne clé primaire naturelle. Avec une cardinalité non nulle et une absence de clé étrangère côté `Moto`, il est possible de fusionner les données en une seule table :

```
1 CREATE TABLE racers (
2   number INT(2) NOT NULL PRIMARY KEY,
3   first_name VARCHAR(50) NOT NULL,
4   last_name VARCHAR(50) NOT NULL,
5   brand VARCHAR(50) NOT NULL,
6   max_speed INT(4) NOT NULL
7 );
```

Bien sûr, ce n'est pas la seule solution. Il aurait également été possible de conserver les deux tables (en rajoutant une clé artificielle dans la table `Moto`) et de rajouter une clé étrangère dans l'une des deux.

p. 14 Solution n°4

```
1 CREATE TABLE clients (
2   id INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
3   first_name VARCHAR(50) NOT NULL,
4   last_name VARCHAR(50) NOT NULL
5 );
```

```

6
7 CREATE TABLE animals (
8     id INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
9     name VARCHAR(50) NOT NULL
10 );
11
12 CREATE TABLE owns (
13     client_id INT(11) NOT NULL,
14     animal_id INT(11) NOT NULL,
15     PRIMARY KEY (client_id, animal_id),
16     FOREIGN KEY (client_id) REFERENCES clients(id),
17     FOREIGN KEY (animal_id) REFERENCES animals(id)
18 );

```

p. 16 Solution n°5

Il faut 4 tables : une pour les personnages et une par type de personnage. La clé primaire de chaque type est la clé étrangère vers le personnage.

```

1 CREATE TABLE characters (
2     id INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
3     name VARCHAR(50) NOT NULL,
4     healthPoints INT(11) NOT NULL
5 );
6
7 CREATE TABLE sellers (
8     user_id INT(11) NOT NULL PRIMARY KEY, - Clef primaire
9     company_name VARCHAR(50) NOT NULL,
10    FOREIGN KEY (user_id) REFERENCES users(id)
11 );
12
13 CREATE TABLE barbarians (
14     character_id INT(11) NOT NULL PRIMARY KEY,
15     furyMultiplier DECIMAL(4,2) NOT NULL,
16     FOREIGN KEY (character_id) REFERENCES characters (id)
17 );
18
19 CREATE TABLE summoners (
20     character_id INT(11) NOT NULL PRIMARY KEY,
21     petName VARCHAR(50) NOT NULL,
22     petHealth INT(11) NOT NULL,
23     FOREIGN KEY (character_id) REFERENCES characters (id)
24 );
25


```

Exercice p. 16 Solution n°6

Exercice

Parmi ces noms de tables, lequel respecte les bonnes pratiques ?


- ☐ User
- ☐ user
- ☐ Users
- ☒ users

 Le nom de la table doit être en minuscules et au pluriel, puisqu'elle désigne l'ensemble des éléments qu'elle contient.

Exercice

Parmi ces noms de champs de table, lequel respecte les bonnes pratiques ?


- ☐ FirstName
- ☐ firstName
- ☒ first_name
- ☐ first-name

 Pour les propriétés de la classe, les bonnes pratiques UML utilisent la syntaxe camelCase, donc avec la première lettre en minuscule et chaque nouveau mot commençant par une majuscule. En revanche, pour les bases de données, il est recommandé d'utiliser la syntaxe snake_case, donc en minuscules et chaque mot séparé par un underscore.

Exercice

Comment modéliser les méthodes dans le modèle relationnel ?


- ☐ Grâce à des triggers
- ☐ Grâce à des procédures stockées
- ☐ Grâce à des programmes SQL
- ☒ Les méthodes ne sont pas représentées dans le modèle relationnel

 Les méthodes ne sont pas dans le modèle relationnel : la base de données doit seulement stocker des données. Leur manipulation appartient à l'application.

Exercice


Une association avec une cardinalité 0..* d'un côté et 0..1 de l'autre, est une relation...

- ☐ One-to-One
- ☒ One-to-Many
- ☐ Many-to-Many

 Cette relation est une relation One-To-Many. On pourrait également l'appeler Many-To-One, selon le sens de lecture de l'association.


Exercice

Comment transformer une relation One-To-Many en modèle relationnel ?

- ☒ En créant deux tables, avec une clé étrangère dans la table **Many** vers la table **One**
- ☐ En créant deux tables, avec une clé étrangère dans la table **One** vers la table **Many**
- ☐ En créant trois tables : une **One**, une **Many** et une table d'association
- ☐ En fusionnant les données dans une seule table
-  Une relation One-To-Many devient deux tables : une pour le côté **Many** et une pour le côté **One**. Une clé étrangère doit être ajoutée dans la table côté **Many** vers la table **One**.


Exercice

Dans une relation One-To-Many, quel est l'impact d'une cardinalité nulle côté **Many** sur le modèle relationnel ?

- ☐ La clé étrangère peut être **NULL**
- ☐ La clé étrangère doit faire partie de la clé primaire
- ☒ Elle n'a pas d'impact
-  Une cardinalité nulle côté **Many** n'a aucun impact sur le modèle relationnel. En revanche, côté **One**, cela signifie que la clé étrangère peut être nulle.


Exercice

Laquelle de ces options n'est **pas** valable pour transformer une relation One-To-One en modèle relationnel ?

- ☐ Créer deux tables, avec une clé étrangère de l'une vers l'autre
- ☐ Fusionner toutes les données en une seule table
- ☒ Créer trois tables : deux tables correspondant aux classes et une table d'association
-  La table d'association n'est utile que dans le cas d'une relation Many-To-Many.

Exercice

Comment choisir parmi les méthodes de transformation des relations One-To-One ?

- ☒ Selon la présence de cardinalités nulles
- ☐ Selon le nombre de propriétés des classes
- ☒ Selon la pertinence d'une fusion
-  La présence de cardinalités nulles indique que de nombreuses colonnes seront nulles en cas de fusion, ce qui n'est pas une structure optimale de données : c'est donc un élément important à prendre en compte pour choisir la méthode de conversion de l'UML en modèle relationnel. S'il n'y a pas de cardinalités nulles, alors le choix de la méthode dépend de ce qui est le plus pertinent : est-ce que la fusion des informations a du sens ? est-ce plus pratique à manipuler ?

Exercice

Quelle est la clé primaire d'une table d'association ?

- ☐ Un identifiant auto-incrémenté
- ☐ Un UUID
- ☒ L'ensemble de ses clés étrangères

Q La clé primaire d'une table d'association est définie par l'ensemble de ses clés étrangères.

Exercice

Quelle est la différence entre une relation One-To-Many et un héritage dans le modèle relationnel ?

- ☒ Dans l'héritage, la clé étrangère est également la clé primaire de la table
- ☐ Dans l'héritage, la clé étrangère peut être nulle
- ☐ Dans l'héritage, la table fille ne possède pas de clé primaire

Q Au moment du passage d'UML au modèle relationnel, l'héritage est à traiter comme une relation One-To-Many, à la différence que la clé étrangère de la table fille vers la table mère est aussi la clé primaire de la table fille. Cela permet de ne pas avoir plusieurs lignes dans la table fille qui correspondent à une ligne de la table mère.

p. 18 Solution n°7

Les trois premières tables à créer sont celles qui n'auront pas de clés étrangères :

- La table `countries`, à laquelle il est préférable d'ajouter un identifiant numérique : bien qu'en théorie, un nom de pays soit unique, il peut évoluer avec le temps (même si c'est un cas rare). Cette table n'a pas de clé étrangère : c'est la table `athletes` qui contiendra une référence vers cette table.
- La table `events`, qui sera référencée dans une table associative avec `athletes`.
- Le cas de la table `hotel_rooms` est particulier : c'est une relation One-To-One, donc la méthode à adopter est au choix du développeur. Cependant, fusionner les informations des athlètes et d'une chambre d'hôtel n'est pas pertinent dans notre cas : ce sont deux classes qui n'ont aucun point commun. Nous allons donc garder deux tables, et mettre une clé étrangère dans `athletes` vers `hotel_rooms`. L'inverse aurait également été possible, mais, les athlètes étant au centre de l'application, ce sens est plus cohérent. **À noter que ces trois solutions sont valides d'un point de vue relationnel** : l'exercice n'est pas faux si vous avez choisi une autre méthode.

Il faut ensuite créer la table `athletes`, avec ses clés étrangères vers `countries` et `hotel_rooms`, ainsi que les tables des classes filles, `judokas` et `sailors`, donc la clé primaire doit être leur clé étrangère.

Enfin, il faut résoudre l'association Many-To-Many avec la table `events` en créant une table d'association `participants`.

```

1 CREATE TABLE countries (
2   id INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
3   name VARCHAR(50) NOT NULL,
4   color VARCHAR(50) NOT NULL
5 );
6
7 CREATE TABLE events (
8   id INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
9   name VARCHAR(50) NOT NULL
10 );
11
12 CREATE TABLE hotel_rooms (
13   number INT(3) NOT NULL PRIMARY KEY,
14   floor int(2) NOT NULL
15 );
16
17 CREATE TABLE athletes (
18   id INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
19   first_name VARCHAR(50) NOT NULL,
20   last_name VARCHAR(50) NOT NULL,

```

```

21     country_id INT(11) NOT NULL,
22     FOREIGN KEY (country_id) REFERENCES countries(id),
23     hotel_room_id INT(11) NOT NULL,
24     FOREIGN KEY (hotel_room_id) REFERENCES hotel_rooms(number)
25 );
26
27 CREATE TABLE judokas (
28     athlete_id INT(11) NOT NULL PRIMARY KEY,
29     weight DECIMAL(3,2) NOT NULL,
30     FOREIGN KEY (athlete_id) REFERENCES athletes(id)
31 );
32
33 CREATE TABLE sailors (
34     athlete_id INT(11) NOT NULL PRIMARY KEY,
35     boat_type VARCHAR(50) NOT NULL,
36     FOREIGN KEY (athlete_id) REFERENCES athletes(id)
37 );
38
39 CREATE TABLE participants (
40     event_id INT(11) NOT NULL,
41     athlete_id INT(11) NOT NULL,
42     PRIMARY KEY (event_id, athlete_id),
43     FOREIGN KEY (athlete_id) REFERENCES athletes(id),
44     FOREIGN KEY (event_id) REFERENCES events(id)
45 );

```