**Modélisez, implémentez et requêtez une base de données relationnelle avec UML et SQL.**

Les bases de données servent à la collecte et à l'enregistrement d'informations. Avant de commencer à enregistrer, vous devez décider quoi enregistrer : c'est le **domaine fonctionnel** de votre projet !

**UML** signifie « Unified Modeling Language ». Il s'agit d'un langage de modélisation graphique normalisé permettant de représenter les multiples aspects de la conception d'un système. Il propose plusieurs types de diagrammes, chacun permettant de décrire les différentes facettes du système (fonctionnalité, architecture logique ou physique...).

**Que va-t-on enregistrer dans la base de données ?**

Grâce au diagramme de classes UML, vous allez décrire ce que vous pensez devoir enregistrer comme information mais aussi comment l'enregistrer (la structuration de l'information)

**La notion d'objet**

Un objet est un élément autonome. Il peut être identifié comme un élément physique « réel » (un crayon, une voiture...) ou comme un élément abstrait ou conceptuel (un ensemble, une liste...).

L'objet possède plusieurs caractéristiques, entre autres :

* un nom : nom commun unique, il donne le type de l'objet (ex : voiture)
* des attributs : ce sont les propriétés de l'objet (ex : couleur, masse à vide, longueur...)

##### La notion de classe

Une classe est le modèle abstrait d'un objet. Elle définit les attributs et les opérations de cet objet. C'est à partir de ce modèle que seront créés les objets « concrets » possédant des valeurs particulières pour leurs attributs.

Ainsi la classe définit les caractéristiques de tous les objets de cette classe (nom, liste des attributs).

##### La notion d'instance

L'instanciation est l'action de création d'un objet à partir d'une classe. Le résultat de cette création est une instance de la classe.

Dans la pratique les termes d’objet et d’instance sont souvent équivalents.

#### Les notions manipulées par la base de données

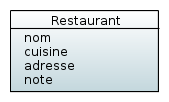
Une base de données relationnelle est un ensemble de tableaux (appelés « **tables** »). Ces **tables** correspondent globalement aux **classes** en AOO.

Chaque **colonne** d'une table représente un **attribut**.

Chaque **ligne** (appelée **tuple** ou **n-uplet**), représente une **instance** donnant ainsi la valeur de chaque attributs de l'objet.

#### Décrire vos premières classes

Voici un exemple de classe en UML (nous allons voir cela plus en détail juste en dessous) :



Dans un diagramme de classes, une classe est représentée par une boîte rectangulaire, comprenant plusieurs parties séparées par des traits horizontaux :

* dans la partie haute : le **nom** de la classe
* dans la partie en dessous : la liste des **attributs**.

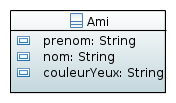
Si nous traduisons cela en base de données, nous allons créer une **table**restaurant qui contiendra les colonnes nom, cuisine, adresse, note (chaque attribut devient une colonne). Chaque **instance** de restaurant sera représentée par une ligne dans cette table :

| **nom** | **cuisine** | **adresse** | **note** |
| --- | --- | --- | --- |
| Chez Luigi | Italien | 1 Rue d'Issy | 4 |
| Burger quid | Fast food | 2 Chai Watt | 1 |
| La cuisine au beurre | Terroir | Le Faubourg Ville | 3 |
| Au lit on dort | Asiatique | 10B Rue Seully | 3 |
| La mère Michelle | Fruits de mer | Port Salle Hutte | 5 |

Nous avons là 5 tuples qui correspondent à 5 instances de la classe restaurant.

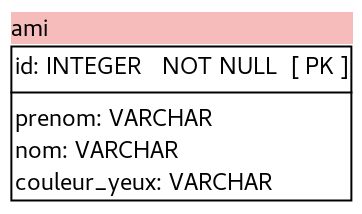
**Passez du diagramme de classes au modèle physique de données.**

Voici la classe Ami, avec en précision supplémentaire le type de donnée de chaque attribut (String ce qui correspond à du texte) :



MPD 🡪 Le modèle physique de données provient de la méthode d'analyse et conception de projet informatique Merise.

Voyons comment est représentée la table ami dans un MPD :



Modèle physique de données ‒ Table « ami »

La méthodologie :

La démarche est assez simple en réalité, comme nous l'avons vu, vous vous basez sur une approche orientée objet :

1. Vous commencez par identifier les concepts ou entités de votre domaine fonctionnel.  
   => Vous listez ainsi les **classes**.
2. Pour chaque classe, vous identifiez ensuite les informations à enregistrer et leur type (texte, nombre...).  
   => Vous obtenez ainsi un **diagramme de classes** avec leurs **attributs**.
3. Vous créez le **modèle physique de données** à partir du diagramme de classes en :
   1. créant une **table** pour chaque classe ;
   2. créant, dans chaque table, une **colonne** pour chaque attribut de la classe correspondante.

**Bien nommer les choses.**

#### Conventions de nommage

Les conventions de nommages sont un peu comme les règles de grammaire et d'orthographe. Vous choisissez le lexique, mais les règles d'orthographe vous indiquent comment écrire.

Il n'y a pas de règles absolues, mais voici celles qui sont généralement observées :

* **En UML :**
  + Pour le nom des **classes** :
    - au singulier
    - avec des lettres non accentuées
    - commençant par une majuscule
    - utilisant la notation chameau (une majuscule au début de chaque mot suivant)

Ex : PierrePrecieuse

* + Pour les **attributs** :
    - au singulier
    - avec des lettres non accentuées
    - commençant par une minuscule
    - utilisant la notation chameau (une majuscule au début de chaque mot suivant)

Ex : couleurYeux

* **Dans le MPD :**
  + Pour le nom des **tables** et des **attributs** :
    - au singulier
    - avec des lettres non accentuées
    - en minuscule
    - les mots étant séparés par des underscores (\_)

Ex : pierre\_precieuse, couleur\_yeux

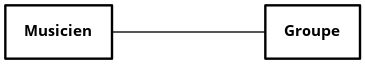
Le but de ces règles est de réduire les possibles erreurs de typographie et d'être compatible avec leur mise en œuvre tant dans le code de l'application que dans la base de données.

**Décrivez les relations**

Une base de données relationnelle s'appuie sur un modèle relationnel et donc il s'agit d'un système possédant des relations entre ses différentes parties (les tables).

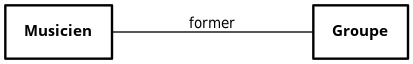
En UML, la relation entre deux classes s'appelle une **association**.

Pour modéliser une association entre deux classes, vous tracez simplement un trait entre-elles.



Une association entre deux classes est matérialisée par un trait

Vous pouvez, si besoin, préciser un nom pour cette association au milieu du trait (généralement un verbe) :

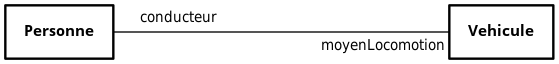


Une association peut avoir un nom

On lit alors la relation comme ceci :

* « Un musicien forme des groupes »
* « Un groupe est formé par des musiciens »

Vous pouvez également donner un rôle à chaque classe en le notant du côté des classes concernées :



Il est possible de préciser le rôle de chaque classe dans une association

On interprète alors la relation comme ceci :

* « Une personne, en tant que conducteur, conduit un véhicule »
* « Un véhicule, en tant moyen de locomotion, est conduit par une personne »

**Posez des limites avec les multiplicités**

Les **multiplicités** servent à apposer des contraintes numériques sur l'association, ou plus précisément, combien d'instances d'une classe peuvent être liées à une instance de la classe de l'autre côté de l'association. En d'autres termes, elles bornent les **cardinalités** des instances liées entre elles.

Si on reprend l'exemple du dessus avec les musiciens, il est possible de fixer le nombre de *musiciens* minimum et maximum qui peuvent jouer dans un *groupe*. De même il est possible de définir dans combien de *groupes* différents (min et max) un même *musicien* peut jouer.

Voici un exemple abstrait montrant comment sont représentées les multiplicités :

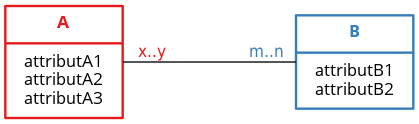


Diagramme de classes ‒ Multiplicités des classes A et B

Cela s'interprète comme ceci :

* Une instance de la classe A doit être associée à au moins m et au plus n instances de la classe B
* Une instance de la classe B doit être associée à au moins x et au plus y instances de la classe A

Si je reprend l'exemple des musiciens, voici, ci-dessous, le diagramme correspondant aux règles suivantes :

* Un musicien joue dans 5 groupes maximum, mais peut aussi ne jouer dans aucun groupe.
* Un groupe est constitué d'au moins 2 musiciens et 10 au maximum

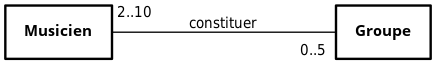
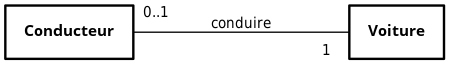


Diagramme de classes ‒ Exemple de multiplicités entre Musiciens et Groupes

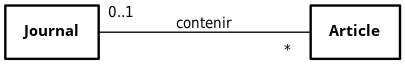
Les 3 catégories d’association

1. **un à un (one-to-one) :**



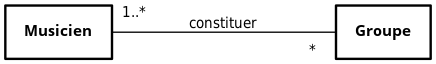
* + Un conducteur conduit une et **une seule**voiture à la fois
  + Une voiture n'est pas conduite (en stationnement) ou conduite par **un seul**conducteur à la fois

1. **un à plusieurs (one-to-many)** ou **plusieurs à un (many-to-one) :**



* + Un journal contient aucun (journal en préparation), un ou **plusieurs**articles
  + Un article est contenu dans aucun (en cours d'écriture) ou **un seul**journal

1. **plusieurs à plusieurs (many-to-many) :**



* + Un musicien fait partie d'aucun, un ou **plusieurs**groupes
  + Un groupe est constitué de un ou **plusieurs**musiciens

**Comprendre les clés primaires**

**L’attribut id**

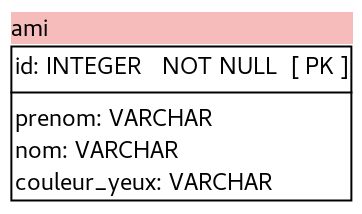
Cet attribut ne représente rien de concret, il ne sert que d'identifiant unique. Vous pouvez l'appeler comme vous le souhaitez, mais en général, il est nommé id ou code

Lors de la création de la table dans la base de données, cet attribut spécifique sera indiqué comme l'identifiant des tuples de la table : c'est la **clé primaire**, ou **primary key**(**PK**) en anglais.

Grâce à cette configuration, la base de données veillera à ce qu'il n'y ait aucun doublon dans cette colonne. Il me sera impossible d'ajouter deux tuples ayant la même valeur dans l'attribut id.

**La clé primaire (PK)**

Dans le modèle physique de données (MPD), la clé primaire est située dans la partie du milieu et porte l'indication PK :



Modèle physique de données ‒ Table « ami »

#### Unique et constante

Afin d'assurer la cohérence du modèle relationnel, la clé primaire, doit ‒ vous l'aurez compris ‒ être **unique** mais elle doit aussi **ne jamais changer**.

Une colonne unique signifie que la base de données ne vous laissera pas créer de doublon dans cette colonne : aucun tuple ne peut avoir la même valeur qu'un autre tuple pour l'attribut contenu dans cette colonne.

Ainsi, un attribut email dans une table utilisateur est unique, mais l'adresse email peut aussi changer au cours de temps. Cet attribut ne devra donc pas être utilisé en tant que clé primaire.

c