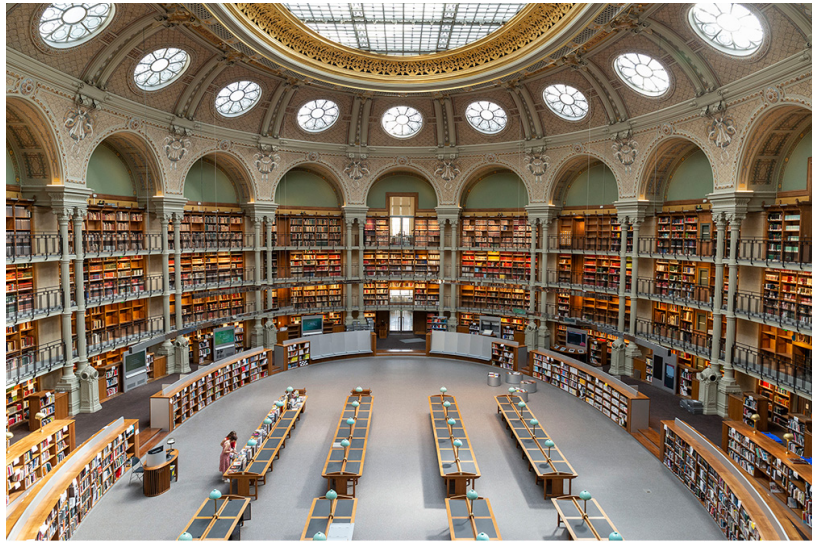
Séquence : **Base de données**

Gestion des emprunts d'une médiathèque

Problématique

Comment sont conçu les **systèmes d'information** qui sont capables de gérer les données relatives à des millions d'objets (articles, livres, morceaux de musique, personnes, etc)?

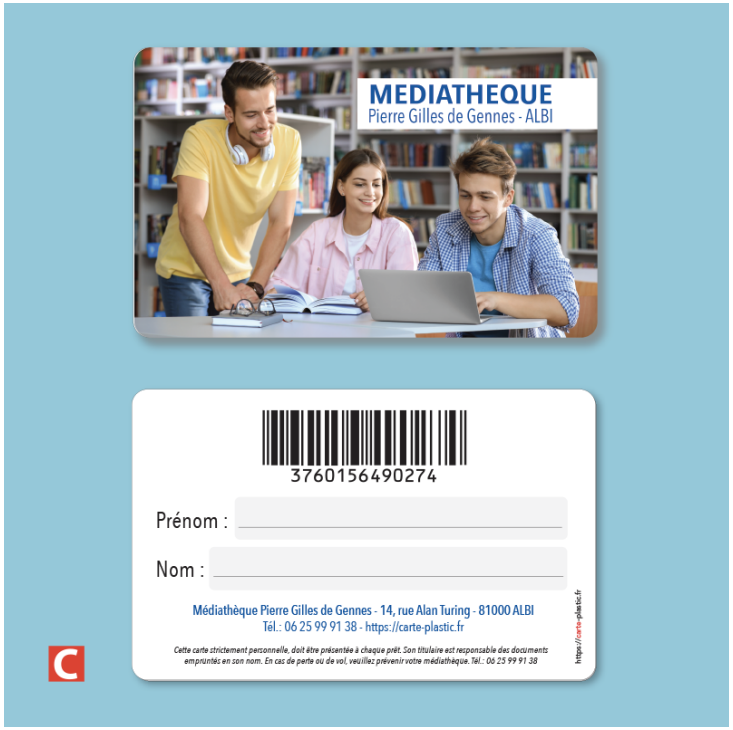


Dans cette activité, nous nous proposons d'analyser le fonctionnement d'une médiathèque pour ensuite construire une **base de données** qui pourrait être utilisées par son *système informatique* pour gérer les emprunts de ses usagers.

Cas d'utilisation

On donne ci-dessous deux *cas d'utilisation* (https://fr.wikipedia.org/wiki/Cas_d%27utilisation), ou *use-case* en anglais, à partir desquels on pourra analyser les besoins fonctionnels liés à notre système d'information.

- **Cas 1** : À la médiathèque municipale, **Gaston** trouve le **livre** qu'il souhaite **emprunter** et approche d'une borne d'emprunt automatique. Il scanne le **code barre** de sa carte de médiathèque et des informations le concernant apparaissent : « Gaston, aucun livre en cours de prêt ». Il scanne ensuite le **code barre** apposé sur le livre. Les informations affichées sont alors mises à jour pour afficher la liste des livres empruntés : « Harry Potter à l'école des sorciers, J. K. Rolling, Gallimard, 1998, ISNB 2070518426 ».



- **Cas 2** : Sur la borne d'à côté, **Charlie** vient rendre ses **livres**. Elle scanne elle aussi le **code barre** de sa carte et les informations la concernant apparaissent à l'écran : « Charlie, un livre en cours de prêt ». Charlie peut alors scanner le **code barre** du **livre** qu'elle rapporte et ce dernier est supprimé.

Système d'information

Les cas d'utilisation précédent permettent d'illustrer les caractéristiques **d'un système d'information** ; un système informatique et humain permettant de gérer l'information.

Dans le système informatique étudié ici, les caractéristiques de chaque objet (un livre), personne (un usager) ou processus (l'emprunt ou la restitution de livre) doit être décrit sous la forme d'une **entité** appartenant à un **modèle de données** qui permettra leur manipulation par un programme.

Ce système doit assurer le stockage de données massives (la Bibliothèque Nationale de France compte plsu de 150 000 000 documents référencés) tout en garantissant l'absence d'erreur lors de leur manipulation. Par exemple, si Gaston repose son livre en rayon sans passer par une borne et que Charlie tente de l'emprunter, le système doit pouvoir détecter le problème. Ces caractéristiques sont celles d'un **système de gestion de base de données** ou SGBD, qui est le composant logiciel principal autour duquel est construit le système d'information de la médiéthèque.

Modélisation

On donne ici la modélisation de chaque objet, ou entité, devant être manipulé par le système informatique.

Modèle relationnel

- Dans notre modèle, chaque entité est représentée sous la forme d'un n-uplet. Il est instancié à partir d'un **schéma**, qui fournit pour chaque composante du n-uplet leur *nom* et leur *domaine* (ensemble des valeurs possibles d'un certain type). On appelle une telle composante un **attribut**.

  Par exemple, on peut modéliser un livre par le *schéma* suivant :

  $$Livre(titre\ \texttt{String}, editeur\ \texttt{String}, annee\ \texttt{Int}, \underline{isbn\ \texttt{String}})$$

  dont une instance serait :

  $$(\texttt{'The C++ Programming Language', 'Addison Wesley', 2013, '03215}$$

  Pour chacun des attributs du schéma ; *titre*, *editeur*, *annee*, etc, ... on stocke ici une valeur de type `String` ou `Int`.

- L'ensemble des entités qui suivent le même schéma forme une **relation**.

  Par exmple, l'ensemble des livres de la médiathèque peut alors être représenté par la relation suivante :

  $$\begin{aligned} Livre = \{ & \\ & (\texttt{'Harry Potter à l'école des sorciers', 'Gallimard', 1998,} \\ & (\texttt{'Docteur Slump, Tome 01', 'Glénat', 2009,'2723472272'}), \\ & (\texttt{'The C++ Programming Language', 'Addison Wesley', 2013, '} \\ & \vdots \\ \} & \end{aligned}$$

- Une **base de données** est un *ensemble de relations*. Par exemple, le schéma de la base de données de la médiathèque peut être composée des schéma de relation suivants :

  $$Livre(titre\ \texttt{String}, editeur\ \texttt{String}, annee\ \texttt{Int}, \underline{isbn\ \texttt{String}})$$

  $$Usager(nom\ \texttt{String}, prénom\ \texttt{String}, adresse\ \texttt{String}, cp\ \texttt{String}, ville$$

  $$Emprunt(\#code\_barre\ \texttt{String}, \underline{\#isbn\ \texttt{String}}, retour\ \texttt{Date})$$

  $$Auteur\ (\underline{a\_id\ \texttt{'Int'}}, nom\ \texttt{String}, prenom\ \texttt{String})$$

  $$Auteur\_de\ (\underline{\#a\_id\ \texttt{'Int'}, \#isbn\ \texttt{String}})$$

Cette manière de représenter les données de la médiathèque sous la forme d'ensembles de n-uplets (les relations) consistue le **modèle relationnel**. On parle donc ici de **bases de données relationnnelles**.

Comment modéliser les objets d'un système?

La modélisation des données se décompose en plusieurs étapes :

1. déterminer les **entités** (objets, actions, personnes, etc...) que l'on souhaite manipuler ;
2. modéliser les ensembles d'entités comme des **relations** (ou tables) en donnant leur **schéma** (ou structure), en s'attachant en particulier à choisir le bon **domaine** pour chaque **attribut** ;
3. définir les **contraintes** de la base de données, c'est à dire l'ensemble des *propriétés logiques* que nos données doivent vérifier à tout moment.

Création de la base de données

Le langage SQL

Directement inspiré du modèle relationnel introduit par *E. Codd*, le **langage SQL** (standardisé selon la norme ISO/IEC 9075) permet la définition de relations, ou *tables*, dans une base de données relationnelle et d'effectuer un certain nombre d'opérations sur ces tables. On formule ainsi des ordres en SQL pour interagir avec un système de gestion de base de données, ou SGBD, qui réalisera effectivement les opérations demandées.

Création de la table `livre`

Le langage SQL va nous permettre de traduire le schéma d'une *relation* du modèle relationnel sous la forme d'un *ordre* correspondant à la création d'une *table* dans la base de données en spécifiant son nom, ses attibuts (les colonnes), les types de ces derniers et les **contraintes** associées à la table.

Ainsi le schéma de la relation

$$Livre(titre\ \texttt{String}, editeur\ \texttt{String}, annee\ \texttt{Int}, \underline{isbn\ \texttt{String}})$$

sera traduit en SQL pour créé la table `livre` correspondante par l'ordre suivant :

```
CREATE TABLE livre (titre VARCHAR(300),
                    editeur VARCHAR(90),
                    annee INT,
                    isbn CHAR(14));
```

Travail à faire

1. Exécute les ordres SQL des cellules suivantes et identifie ce que permet de faire chacun d'entres eux.
2. Quel anomalie observes-tu dans la base de données ainsi obtenue?

Entrée[ ]:
```
# Ordres SQL
```

Contrainte de relation

La **contrainte de relation** permet de s'assurer que chaque élément d'une relation est *unique* et identifie une entité (ou enregistrement dans une table) de manière non ambigüe.

La création de la table `livre` tel que formulé ci-dessus engendre une **anomalie de redondances de données** : la table possède deux enregistrements identiques.

Il faut donc définir un attribut (ou un ensemble d'attributs) comme **clé primaire** de la relation, dont la valeur dans la relation est unique et qui permet d'identifie sans ambiguïté une entité.

Dans le cas de le relation $Livre$, l'attribut $isbn$ répond excatement à ce critère puisqu'il s'agit d'un numéro identifiant de manière unique chaque édition de chaque livre publié dans le monde.

On va donc ajouter à l'ordre de création de la table les mots-clés `PRIMARY KEY` dans la spécification de l'attribut `isbn` :

```
CREATE TABLE livre (titre VARCHAR(300),
                    editeur VARCHAR(90),
                    annee INT,
                    isbn CHAR(14) PRIMARY KEY);
```

> **Travail à faire**
>
> Après avoir supprimer la table précédement créée avec l'ordre
>
> ```
> DROP TABLE livre;
> ```
>
> Recopie les quatres ordres précédents en ajout la clé primaire à la table `livre` puis exécute-les. Quelle différence de comportement observes-tu par rapport au cas précédent?

Entrée[ ]:
```
1 # Ordres SQL
```

Création de la table `usager`

On souhaite maintenant compléter notre base de données avec la relation *usager* :

$$Usager(nom\ \texttt{String}, prénom\ \texttt{String}, adresse\ \texttt{String}, cp\ \texttt{String}, ville\ \texttt{Stri}$$

L'attribut *code_barre*, qui correspond au code-barre unique de la carte fournie à chaque usager de la médiathèque lors de son inscription, a été défini comme clé primaire dans le schéma de la relation, comme l'indique son soulignement.

> **Travail à faire**
>
> 1. Exécute les ordres SQL donnés ci-dessus puis observe la table obtenue. Quelles anomalies peux-tu relever?
> 2. Quelles contraintes sur les valeurs des attributs permettraient de les éviter?

Entrée[ ]:
```
1 # Ordres SQL
```

> **Contraintes de domaine**
>
> Les **contraintes de domaine** sont toutes les propriétés que le domaine d'un attribut, c'est à dire l'ensemble des valeurs que peut prendre cet attribut, peut garantir.
>
> Elles doivent permettre de :
>
> - représenter excatement et sans perte d'information toutes les valeurs possibles pour un attribut ;
> - limiter autant que possible la saisie de valeurs illégales ou mal formées.

Par exemple, on peut relever les *anomalies d'insertion* suivantes dans le cas donné ci-dessus :

- il manque un chiffre au code postal d'Edgar Codd,
- Linus Torvalds possède la même adresse de courriel qu'Edgar Codd,
- auncun prénom n'est fourni pour Mme Lovelace.

On va donc mettre en place les contraintes de domaine suivantes pour éviter ces *anomalies* :

- on remplace le type `INT` par le type `CHAR` qui permet d'obtenir exactement cinq caractères ;
- on ajoute la contrainte `CONSTRAINT [format_cp]` qui vérifie que la valeur de `cp` possède exactement cinq caractère non vide avec `CHECK ([cp] LIKE '_____')` ;
- on déclare les attributs comme `NOT NULL` pour rendre obligatoire leur valeur ;
- on déclare l'attribut `courriel` comme `UNIQUE` .

Au final on obtient l'ordre

```
CREATE TABLE usager (nom VARCHAR(90) NOT NULL,
                     prenom VARCHAR(90) NOT NULL,
                     adresse VARCHAR(300) NOT NULL,
                     cp CHAR(5) NOT NULL,
                     ville VARCHAR(60) NOT NULL,
                     courriel VARCHAR(60) NOT NULL UNIQUE,
                     code_barre CHAR(15) PRIMARY KEY,
                    CONSTRAINT format_cp CHECK (cp LIKE '___
__'));
```

> **Travail à faire**
>
> Applique une à une les modification proprosées puis observe les erreurs de non conformité obtenues en essayant d'insérer les mêmes enregistrements que précédemment.

Entrée[ ]:
```
1 # Ordres SQL
```

Création de la table `Emprunt`

Une autre relation issue de la modélisation de la médiathèque est la relation *emprunt* :

$$Emprunt(code\_barre\ \texttt{String}, \underline{isbn\ \texttt{String}}, retour\ \texttt{Date})$$

que l'on créera dans notre base de données avec lordre suivant :

```
CREATE TABLE emprunt (code_barre CHAR(15) REFERENCES Usager
(code_barre),
                      isbn CHAR(14) PRIMARY KEY REFERENCES L
ivre(isbn),
                      retour DATE NOT NULL);
```

Travail à faire

1. Observe le schéma de la relation $Emprunt$. Pour ajouter une entité à cette relation, quelles contraintes doit-elle respecter par rapport aux relations $Livre$ et $Usager$?

2. Crée la table `emprunt` avec l'ordre `CREATE TABLE` donné ci-dessus, puis essaie d'y insérer l'enregistrement suivant :

```
INSERT INTO emprunt VALUES ('934701281931582', '207051842
6', '2024-11-31');
```

Observe l'erreur obtenue et essaie d'en expliquer la cause. Modifier la base de données pour la faire disparaître.

Entrée[ ]:
```
# Ordres SQL
```

Contraintes de référence

Les **contraintes de référence** permettent de créer des associations entre deux relations, en garantissant qu'une entité d'une relation mentionne une entité existante dans une autre relation.

Pour garantir la satisfaction de cette contrainte, on définit un attribut de la première relation comme une **clé étrangère**.

En effet, une clé étrangère est un ensemble d'attributs d'une table qui sont une clé primaire dans une autre table.

Cela garantie donc que la valeur donnée à une clé étrangère existe effectivement dans une autre relation comme clé primaire et empêche la suppression de cette dernière tant qu'elle est référencée en tant que clé etrangère dans une autre relation.

Ici on définit les attributs $code\_barre$ et $isbn$ comme des *clés étrangères*, indiquant ainsi que ces attributs doivent obligatoirement figuer dans les relations $Usager$ et $Livre$ comme clé primaire.

Cela garanti que :

- chaque emprunt fait référence à un livre existant dans la médiathèque et est associé à un usager enregistré ;
- un livre ou un usager apparaissant ayant un emprunt en cours ne puisse pas être supprimé dans la base de données.

On remarque aussi que l'utilisation de clés étrangères permet de créer des associations multiples entre entités de plusieurs relations. En effet, on peut associer plusieurs livres en cours d'emprunt à un même usager, puisse $code\_barre$ n'est qu'une clé étrangère dans le schéma de $emprunt$.

On a utilisé cette technique dans notre modélisation pour associer des livres à leurs auteurs. En effet, le schéma de $Livre$ ne mentionne pas directement les auteurs d'un ouvrage :

$$Livre(titre\ \texttt{String}, editeur\ \texttt{String}, annee\ \texttt{Int}, \underline{isbn\ \texttt{String}})$$

Mais en définissant une relation $Auteur\_de$ selon le schéma :

$$Auteur\_de\ (\underline{a\_id\ \texttt{'Int'}, isbn\ \texttt{String}})$$

on obverse que les attributs a_id et $isbn$ étant des clé primaires des relations $Auteur$ et $Livre$, ce sont donc des clés étrangères de $Auteur\_de$. Ainsi la relation $Auteur\_de$ permet d'associer un ou plusieurs auteurs à un même livre.

Travail à faire

1. Sur le modèle de l'ordre de création de la table `emprunt` , crée les tables `auteur` et `auteur_de` en définissant bien les attributs comme clé étrangère lorsque cela est nécessaire.
2. Ajouter les enregistrements à `auteur` et `auteur_de` permettant d'associer aux enregistrements suivants de `Livre`

('Harry Potter à l'école des sorciers', 'Gallimard', 1998, '207051
('Docteur Slump, Tome 01', 'Glénat', 2009,'2723472272')
('The C++ Programming Language', 'Addison Wesley', 2013, '03215638

leur auteur respectif :

- Harry Potter à l'école des sorciers : J. K. Rollings
- Dr Slump : Akira Toriyama
- `The C++ Programming Language` : Bjarne Stroustrup

Entrée[ ]:
```
# Ordres SQL
```