Modèle relationnel

Résumé

Dernière mise à jour le : 02/10/2024

Introduction

Le développement de traitements informatiques nécessite la manipulation de données de plus en plus nombreuses. Leur organisation et leur stockage constituent un enjeu essentiel de performance.

Comment gérer (mémoriser et traiter) un ensemble volumineux de données ?

La vidéo ci-dessous répond à cette question :

▶️ Source vidéo : https://dai.ly/x71hy5c

En classe de Première, on a vu comment gérer des données représentées de manière tabulaire (avec des fichiers CSV). Il était possible d'utiliser un langage de programmation pour effectuer les traitements. Cette façon de faire convient pour des requêtes simples dès lors que les données ne sont pas trop nombreuses, mais devient rapidement insuffisante pour répondre aux attentes actuelles :

- souvent le volume des données est gigantesque (voir l'article : 16000 malades oubliés à cause d'Excel);
- les requêtes peuvent être complexes ;
- les données peuvent être simultanément utilisées par différents programmes ou différents utilisateurs (exemples : sites marchands, réservations en ligne, etc.)

Il est donc nécessaire d'utiliser des solutions plus performantes et l'utilisation de **bases de données relationnelles** est aujourd'hui la solution la plus répandue.

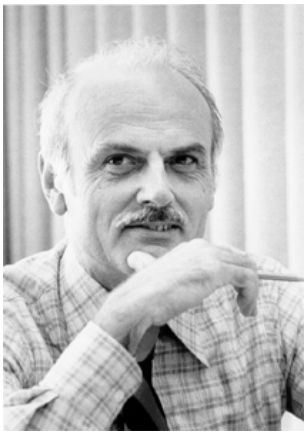
Seules les bases de données *relationnelles* sont au programme de Terminale NSI, mais il existe d'autres types de bases de données : les bases réseaux, les bases objets, les bases « no-sql », etc.

Dans une base de données, les informations sont stockées dans des fichiers, mais à la différence des fichiers au format CSV, il est impossible de travailler avec ces fichiers avec un éditeur de texte. Pour manipuler les données présentes dans une base de données, il faut utiliser un logiciel appelé *système de gestion de bases de données*, abrégé SGBD. Il en existe plusieurs, des gratuits, des payants, des libres, des propriétaires (nous en utiliserons dans le chapitre suivant).

Modèle relationnel

Les bases de données relationnelles sont basées sur ce qu'on appelle le **modèle relationnel**. Il s'agit d'un modèle *logique* (basé sur des concepts mathématiques) défini en 1970 par l'informaticien britannique Edgard F. Codd (1923-2003), lors de

ses travaux chez IBM. Il a reçu le prix Turing en 1981.

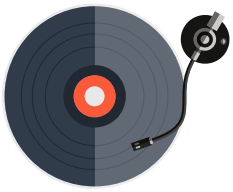


Edgar F. Codd
Source : wikimedia.org

Le modèle relationnel est une manière de modéliser les **relations** existantes entre plusieurs informations et de les ordonner entre elles.

Un modèle relationnel est donc basé sur des **relations**, terme que nous allons définir dans le paragraphe suivant.

Relation, attribut, domaine, schéma



Prenons l'exemple d'un disquaire permettant d'emprunter des albums de musique. L'ensemble de ses albums peut être représenté par l'ensemble :

```
Album = {
    ("I Still Do", "Eric Clapton", 2016, Vrai),
    ("Axis: Bold as Love", "Jimi Hendrix", 1967, Faux),
    ("Continumm", "John Mayer", 2006, Faux),
    ("Riding With The King", "Eric Clapton et B.B. King", 2000, Faux),
    ("Don't explain", "Joe Bonamassa et Beth Hart", 2011, Vrai),
    ...
}
```

Un tel ensemble s'appelle une **relation** (la relation *Album* en l'occurrence).

Attention : cette relation *Album* n'est pour le moment pas satisfaisante ! Nous verrons plus loin pourquoi et comment y remédier.

Les différents éléments d'une relation s'appellent des **enregistrements**, ou **tuple**, ou **n-uplet**, ou **t-uplet**, ou **vecteur**. Les enregistrements d'une relation possèdent les mêmes composantes, que l'on appelle les **attributs** de la relation.

Une relation se conforme toujours à un **schéma** qui est une description indiquant pour chaque attribut de la relation son *nom* et son **domaine** (= le « type » de l'attribut : un entier, une chaîne de caractères, une date, etc.)

Ainsi, pour le moment, la relation *Album* possède 4 **attributs** :

- `titre` : le titre de l'album, une chaîne de caractères
- `artiste` : le ou les artistes de l'album, une chaîne de caractères
- `annee` : l'année de parution de l'album, un entier naturel
- `dispo` : disponibilité de l'album, un booléen

On note le schéma de la relation *Album* de la façon suivante :

```
Album(titre TEXT, artiste TEXT, annee INT, dispo BOOL)
```

On a choisi de noter ici le *domaine* de chaque attribut avec les mots INT, TEXT, BOOL mais on aurait pu également les écrire Entier, Chaîne de caractères, Booléen ou Int, String, Bool, etc. Cela n'a pas vraiment d'importance car le modèle relationnel est indépendant de toute considération informatique.

Une relation peut aussi se représenter sous forme d'une **table**, et d'ailleurs on utilise souvent de manière équivalente les deux termes : *relation* ou *table*. Par exemple, la table correspondant à notre relation *Album* (qui n'est pas encore satisfaisante) ressemble à ceci :

titre	artiste	annee	dispo
I Still Do	Eric Clapton	2016	Vrai
Axis: Bold as Love	Jimi Hendrix	1967	Faux
Continuum	John Mayer	2006	Faux
Riding With The King	Eric Clapton et B.B. King	2000	Faux
Don't explain	Joe Bonamassa et Beth Hart	2011	Vrai
...	...	...	...

Base de données relationnelle

Une base de données relationnelle est un *ensemble de relations*. Par exemple, la base de données de notre disquaire ne contiendra pas uniquement la relation *Album*. Elle peut par exemple contenir deux autres relations : *Client* et *Emprunt* qui correspondent respectivement à l'ensemble des clients du disquaire et à l'ensemble des emprunts en cours.

Le *schéma*, ou la *structure*, d'une base de données relationnelle est l'ensemble des schémas des relations de la base. Ainsi, pour le moment, la structure (ou schéma) de la base de données du disquaire, est :

```
Album(titre TEXT, artiste TEXT, annee INT, dispo BOOL)
Client(...)
Emprunt(...)
```

où on n'a pour le moment pas complété les schémas des relations *Client* et *Emprunt* (et où le schéma de la relation *Album* n'est pas satisfaisant pour l'instant)

Concevoir la structure d'une base de données relationnelle

La conception de la structure d'une base de données n'est pas toujours aisée mais c'est un travail absolument nécessaire pour obtenir une base ne souffrant d'aucune anomalie et offrant des performances optimales.

On trouve en général ces trois étapes :

1. Déterminer les entités (objets, actions, personnes, ...) que l'on souhaite manipuler.
2. Modéliser chaque ensemble d'entités comme une relation en donnant son schéma : attributs et domaine de chaque attribut.
3. Définir les *contraintes d'intégrité* (domaine, relation, référence) de la base de données, c'est-à-dire toutes les propriétés logiques vérifiées par les données à chaque instant.

On réalise souvent ces opérations en parallèle de manière à peaufiner au fur et à mesure la structure de la base.

Nous allons expliquer ces mécanismes de conception en s'appuyant sur la base de données de notre disquaire que l'on va affiner au fur et à mesure.

Domaine d'un attribut

Le domaine d'un attribut a déjà été abordé, il s'agit du "type" de l'attribut. Dans le cas de la modélisation d'une base de données, la façon de noter les domaines n'est pas primordiale (INT ou Entier ou Int ou Integer, etc. pour désigner un attribut dont les valeurs sont des entiers), mais elle le deviendra lorsque l'on créera concrètement les tables en base de données car il faudra respecter la syntaxe du SGBD utilisé.

Contrainte de domaine

Concrètement, un SGBD doit s'assurer à chaque instant de la validité des valeurs d'un attribut, autrement dit que ces valeurs correspondent toujours au domaine de l'attribut, on appelle cela les *contraintes de domaines*. C'est pourquoi en pratique, la commande de création d'une table doit préciser en plus du nom des attributs, leurs domaines.

Les contraintes de domaines doivent être respectées en permanence par le SGBD : si un attribut a pour domaine INT et que l'on essaie de saisir une valeur de type FLOAT pour cet attribut, cela provoquera une erreur du SGBD. Il est donc important de bien penser le domaine de chaque attribut dès le départ.

Bien que le domaine d'un attribut paraisse assez simple à déterminer, il faut être prudent dans certains cas. Par exemple, si le domaine d'un attribut correspondant à un code postal est INT, alors si on enregistre un code postal `05000` alors celui-ci sera converti en `5000` (car 05000 = 5000 pour les entiers), ce qui ne correspond pas à un code postal valide... Il est donc nécessaire de donner le domaine TEXT à un code postal.

Clé primaire

> **Définition** : Une **clé primaire** est un attribut (ou une réunion d'attributs) qui permet d'identifier de manière *unique* un enregistrement d'une relation.

La relation *Album*

Et là c'est le drame : quel attribut de notre relation *Album* peut-il jouer le rôle de clé primaire ? ... aucun !

En effet, il est (fort) possible que plusieurs albums aient le même titre (ne serait-ce qu'un album disponible en plusieurs exemplaires), que plusieurs albums concernent le même artiste et que plusieurs albums soient sortis la même année ! De manière évidente, l'attribut `dispo` ne permet pas d'identifier un album de manière unique non plus.

Pour y remédier, on va créer "artificiellement" un attribut `id_album` (de type INT) qui va jouer le rôle de clé primaire, chaque album possédant un attribut `id_album` différent (on utilise "id" pour "identifiant").

Pour symboliser la clé primaire dans le schéma d'une relation, il est de coutume de la souligner. Ainsi, notre relation *Album* a pour schéma :

```
Album(id_album INT, titre TEXT, artiste TEXT, annee INT, dispo BOOL)
```

et correspond à la table suivante :

id_album	titre	artiste	annee	dispo
2	I Still Do	Eric Clapton	2016	Vrai
5	Axis: Bold as Love	Jimi Hendrix	1967	Faux
24	Continuum	John Mayer	2006	Faux
25	Continuum	John Mayer	2006	Faux

8	Riding With The King	Eric Clapton et B.B. King	2000	Faux
11	Don't explain	Joe Bonamassa et Beth Hart	2011	Vrai
...	...	...	...	...

La relation *Client*

On suppose que le disquaire récolte les informations suivantes sur ses clients : un nom, un prénom et une adresse email.

- Si on choisit le nom ou le prénom comme clé primaire, il sera impossible d'enregistrer deux clients portant le même nom ou portant le même prénom, ce qui n'est pas rare.
- De même, si on choisit le couple (nom, prénom) comme clé primaire, cela empêche d'enregistrer des homonymes, ce qui peut très bien arriver également.
- Si on choisit l'adresse email comme clé primaire, cela impliquerait que deux clients ne peuvent pas avoir la même adresse email. Cela peut sembler convenir... mais on se heurterait au cas où un client ne possède pas d'adresse email (un jeune enfant par exemple, d'ailleurs ses parents ne pourraient même pas lui créer un compte à son nom avec leur propre adresse email s'ils sont eux-mêmes clients)

Comme pour la relation *Album*, il semble judicieux de créer une clé primaire *artificielle*, nommée `id_client` la relation *Client* qui aurait alors pour schéma :

```
Client(id_client INT, nom TEXT, prenom TEXT, email TEXT)
```

et correspond à la table suivante :

id_client	nom	prenom	email
1	Dupont	Florine	dupont.florine@domaine.net
5	Pacot	Jean	jpacot@music.com
8	Rouger	Léa	NULL
3	Marchand	Grégoire	greg.marchand49@music.com

Remarque : Il est parfois possible de trouver une clé primaire sans avoir besoin d'en créer une artificiellement. Par exemple dans une relation *Livre*, le numéro ISBN pourrait jouer le rôle de clé primaire car il est unique pour chaque livre existant. Cependant en pratique, un SGBDR va souvent créer un identifiant unique pour chaque enregistrement d'une entité dans la base de données. Pour cela, un mécanisme d'*auto-incrément* est mis en oeuvre (si la clé primaire de la dernière entité créée est l'entier 57, alors la clé primaire d'une nouvelle entité créée sera 58)

Contrainte de relation

Une des contraintes d'intégrité d'une base de données s'appelle la *contrainte de relation*. Celle-ci impose que chaque enregistrement d'une relation soit *unique*. C'est donc la présence d'une *clé primaire* dans chaque relation qui permet de réaliser cette contrainte.

Clé étrangère

La relation *Emprunt*

Pour un emprunt, on aimerait connaître l'album emprunté, le client qui a emprunté l'album et la date d'emprunt.

On voit donc que les enregistrements de la relation *Emprunt* font référence à des enregistrements des relations *Album* et *Client*. On peut imaginer le schéma suivant pour la relation *Emprunt*, qui contient toutes les informations nécessaires :

```
Emprunt(id_client INT, nom TEXT, prenom TEXT, email TEXT, id_album INT,
        titre TEXT, artiste TEXT, annee INT, dispo BOOL, date DATE)
```

Cela donnerait une table *Emprunt* du genre :

id_client	nom	prenom	email	id_album	titre	artiste	annee	dispo	date
1	Dupont	Florine	dupontf@domaine.net	5	Axis: Bold as Love	Jimi Hendrix	1967	Faux	10/09/2021
3	Mira	Grégoire	gmira49@music.com	8	Riding With The King	Eric Clapton et B.B. King	2000	Faux	18/08/2021
3	Mira	Grégoire	gmira49@music.com	24	Continumm	John Mayer	2006	Faux	18/08/2021
5	Pacot	Jean	jpacot@music.com	25	Continumm	John Mayer	2006	Faux	12/09/2021

Attention : la relation *Emprunt* n'est pour le moment pas satisfaisante, nous allons l'améliorer un peu plus bas.

Définition : Une **clé étrangère** d'une relation est un attribut qui est clé primaire d'une autre relation de la base de données.

Ainsi, la relation *Emprunt* donnée plus haut possède deux *clés étrangères* : `id_client` et `id_album` (qui sont des clés primaires respectives des relations *Client* et *Album*.).

Que peut-on choisir comme clé primaire de la relation *Emprunt* ?

Une même personne pouvant emprunter plusieurs albums en même temps, il n'est pas possible d'utiliser les attributs correspondant à la relation *Client*. En revanche, comme un même album ne peut pas être emprunté par deux clients en même temps, on peut choisir `id_album` comme *clé primaire* de la relation *Emprunt*.

Redondance des données

Dans une base de données relationnelle, il faut éviter la *redondance des données* c'est-à-dire qu'une relation ne doit pas contenir des informations déjà disponibles dans d'autres relations (et de manière générale, éviter que des mêmes informations se retrouvent dans plusieurs enregistrements d'une même relation).

Par exemple, la relation *Emprunt* telle que nous l'avons définie plus haut contient beaucoup d'informations redondantes. En effet :

- pour faire le lien avec l'emprunteur, il est inutile de garder simultanément les attributs `id_client`, `nom`, `prenom` et `email` : il suffit de conserver l'attribut `id_client` qui fait entièrement référence à un unique client de la relation *Client* dans laquelle on retrouve le nom, le prénom et l'adresse email de celui-ci. Ainsi, on évite la redondance des attributs `nom`, `prenom` et `email` : on ne les garde que dans la relation *Client* ;
- de même, pour faire le lien avec l'album emprunté, il suffit de conserver l'attribut `id_album` qui caractérise entièrement un album et permet de retrouver le titre, l'artiste, l'année et la disponibilité dans la relation *Album*.

Moralité : ce sont les clés étrangères (ici `id_client` et `id_album`) qui permettent à elles seules de faire le lien avec des entités d'autres relations, et on évite ainsi les redondances.

Sachant que l'on peut noter les clés étrangères d'une relation en utilisant un "#", on peut désormais écrire une version satisfaisante de la relation *Emprunt* :

```
Emprunt(#id_client INT, #id_album INT, date DATE)
```

La clé `id_album` est donc à la fois clé primaire et clé étrangère de la relation *Emprunt*. La clé `id_client` est une clé étrangère mais pas une clé primaire de la relation *Emprunt* : cela implique qu'un même client peut se trouver plusieurs fois dans la relation *Emprunt*, il peut donc emprunter plusieurs albums à la fois (et heureusement !).

On obtient ainsi la table correspondant à la relation *Emprunt* :

id_client	id_album	date
1	5	10/09/2021
3	8	18/08/2021
3	24	18/08/2021
5	25	12/09/2021

Pourquoi éviter la redondance des données ?

La redondance des données est considérée comme une *anomalie* d'une base de données, synonyme d'une mauvaise conception de la base. En effet, celle-ci est proscrite pour plusieurs raisons :

- faire apparaître des informations non nécessaires à plusieurs endroits (dans plusieurs relations) d'une base de données entraîne un coût en mémoire plus important et des performances moindres lorsqu'il s'agira d'effectuer des requêtes sur la base ;
- si des corrections doivent être faites, elles doivent être faites à un seul endroit : imaginez qu'un emprunteur change de nom (ou d'adresse email), si on a pris soin de ne pas le faire apparaître dans la table *Emprunt*, il suffit alors de le modifier (une seule fois) dans la table *Client* et on n'a pas à faire la modification sur chaque ligne de la table *Emprunt*. Cela permet d'amener de la *cohérence* à notre base de données.

Contrainte de référence

La cohérence et les relations entre les différentes tables sont assurées par les clés étrangères. Elles permettent de respecter ce qu'on appelle les *contraintes de référence* :

- une clé étrangère d'une relation doit nécessairement être la clé primaire d'une autre relation :

> 👉 cela permet de s'assurer de ne pas ajouter des valeurs fictives ne correspondant pas à des entités connues de la base de données ;

- un enregistrement ne peut être effacé que si sa clé primaire n'est pas associée à des enregistrements liés dans d'autres relations :

> 👉 on ne pourrait pas supprimer le client "Dupont Florine" de la relation *Client* car il apparaît dans les enregistrements de la relation *Emprunt*. En effet, sinon la valeur '1' de la clé étrangère `id_client` de la table *Emprunt* ne serait plus une clé primaire d'une autre table.

- une clé primaire ne peut pas être modifiée si l'enregistrement en question est associé à des enregistrements liés dans d'autres tables :

> 👉 on ne pourrait pas modifier la clé primaire `id_client` du client "Dupont Florine" dans la relation *Client* car elle apparaît dans les enregistrements de la relation *Emprunt*. En effet, sinon la valeur '1' de la clé étrangère `id_client` de la table *Emprunt* ne serait plus une clé primaire d'une autre table.

Concrètement, une tentative de violation de contrainte de référence provoquerait une erreur du SGBD.

Liens entre albums et artistes

On termine la modélisation de la structure de la base de données du disquaire en définissant un peu mieux le lien entre un album et l'artiste (ou les artistes) de l'album.

Pour le moment, il a été choisi d'utiliser une chaîne de caractères pour l'artiste d'un album directement dans la relation *Album*. Cette façon de faire peut conduire à quelques problèmes :

- rien n'empêche d'associer plusieurs fois le même artiste à un album puisqu'on écrit ce que l'on veut dans une chaîne de caractères : on pourrait écrire `"Éric Clapton et Éric Clapton"` sans que le SGBD ne provoque une erreur, alors même qu'il y aurait un problème de cohérence.
- on a de plus un problème de redondance dans le cas (bien que rare) où un artiste changerait de nom car il faudrait le modifier pour chaque album de la table *Album*.

Pour pallier à ces problèmes, on peut :

- scinder la relation *Album* en trois relations : *Album*, *Artiste* et *Artiste_de* ;
- et utiliser les *clés étrangères* pour faire les associations nécessaires entre les artistes et les albums.

Concrètement :

- On retire l'attribut `artiste` de la relation *Album* :

```
Album(id_album INT, titre TEXT, annee INT, dispo BOOL)
```

- On crée un nouvelle relation, *Artiste*, correspondant uniquement aux différents artistes et ayant le schéma suivant :

```
Artiste(id_artiste INT, nom TEXT, prenom TEXT)
```

- On associe, grâce aux clés étrangères, les artistes aux albums en créant une nouvelle relation *Artiste_de* :

```
Artiste_de(#id_artiste INT, #id_album INT)
```

Dans cette dernière relation, les clés étrangères `id_artiste` et `id_album` permettent d'associer les relations *Artiste* et *Album*. Le *couple* (`id_artiste`, `id_album`) forme la clé primaire de la relation *Artiste_de*.

Ainsi, un même artiste **et** un même album ne peuvent se trouver plusieurs fois dans la relation, ce qui empêche d'associer deux fois le même artiste à un même album. Mais un même artiste peut être associé à plusieurs albums différents car `id_artiste` n'est pas clé primaire de la relation.

Ces transformations donnent des tables ressemblant à :

- Relation *Album* :

id_album	titre	annee	dispo
2	I Still Do	2016	Vrai
5	Axis: Bold as Love	1967	Faux
24	Continuum	2006	Faux
25	Continuum	2006	Faux
8	Riding With The King	2000	Faux
11	Don't explain	2011	Vrai

- Relation *Artiste* :

id_artiste	nom	prenom
1	Clapton	Éric
3	Hendrix	Jimi
4	Mayer	John
8	B.B. King	NULL
6	Hart	Beth
15	Bonamassa	Joe

- Relation *Artiste_de* :

id_artiste	id_album
1	2
1	8
8	2
4	24
4	25
3	5
6	11
15	11

Désormais, on n'enregistre qu'à un seul endroit les chaînes de caractères correspondant au nom et au prénom de chaque artiste, et lorsque l'on veut faire référence à cet artiste dans une autre relation, c'est l'entier correspondant à `id_artiste` qui est enregistré en mémoire. Ainsi, si un artiste change de nom, il suffit alors de modifier son nom une seule fois dans la relation *Artiste*.

Par ailleurs, on obtient également un gain :

- en mémoire, car un entier est (presque tout le temps) stocké sur moins de bits qu'une chaîne de caractères ;
- en performance, car lorsque l'on effectuera des recherches dans la base (comme la recherche de tous les albums d'un artiste donné par exemple), les comparaisons entre deux entiers sont plus rapides qu'entre deux chaînes de caractères.

Schéma (final) de notre base de données

Avec toutes les améliorations apportées, le schéma (ou structure) de la base de données du disquaire est le suivant :

```
Album(id_album INT, titre TEXT, annee INT, dispo BOOL)
```

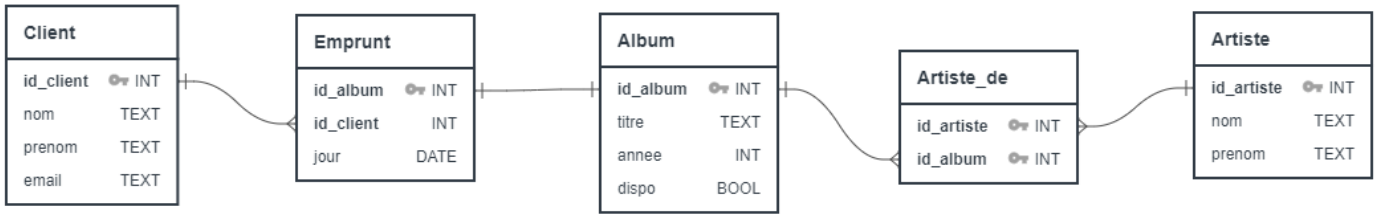
```
Artiste(id_artiste INT, nom TEXT, prenom TEXT)
```

```
Artiste_de(#id_artiste INT, #id_album INT)
```

```
Client(id_client INT, nom TEXT, prenom TEXT, email TEXT)
```

```
Emprunt(#id_client INT, #id_album INT, date DATE)
```

On peut aussi représenter graphiquement ce schéma par le diagramme suivant :



Réalisé avec l'application quickdatabasediagrams.com

Remarques : Dans ce diagramme :

- les clés primaires sont matérialisées ici par un symbole de clé (et en gras). Mais on trouve aussi parfois l'acronyme CP, pour *clé primaire*, ou plus souvent sa version anglaise PK, pour *primary key*.

- les clés étrangères sont matérialisées par un trait marquant les associations entre les différentes relations (et en gras). Mais on trouve aussi souvent l'acronyme FK (*foreign key*, traduction de *clé étrangère*).

Bilan

- Pour stocker, manipuler, traiter des données de plus en plus nombreuses, l'utilisation de fichiers texte ou tabulaire (CSV) ne suffit plus. Pour cela, on utilise des bases de données (relationnelles), beaucoup plus performantes. Les logiciels de type SGBD permettent aux utilisateurs d'interagir avec une base de données.
- Le **modèle relationnel** permet de modéliser les relations entres plusieurs informations et les relier entre elles. Une relation est un ensemble d'enregistrements possédant des **attributs**, chacun d'eux ayant un **domaine** défini qui permet de réaliser la *contrainte de domaine* de la base de données. Le *schéma d'une relation* est la liste de tous les attributs et de leurs domaines respectifs.
- Une **base de données relationnelle** n'est autre qu'un ensemble de relations et le schéma (structure) d'une base de données relationnelle est l'ensemble des schémas des relations la constituant.
- Chaque relation d'une base de données doit posséder une **clé primaire** permettant de caractériser de manière unique chaque entité de la relation. Ces clés primaires permettent de réaliser la *contrainte de relation* de la base de données.
- Certaines relations possèdent un lien entre elles. Ce lien est réalisé par des **clés étrangères** (qui sont des clés primaires d'autres relations) qui assurent les *contraintes de référence* de la base de données et permettent d'éviter les redondances.
- La conception de la structure d'une base de données (son schéma) est un travail indispensable pour s'assurer qu'elle ne contient pas d'anomalies. Vous devez être capable de repérer des anomalies dans le schéma d'une base de données, cela sera abordé plus en détails en exercices.
- Dans un prochain chapitre, nous utiliserons un SGBD pour interagir avec une vraie base de données, grâce au langage SQL (Structured Query Language).

Références :

- Equipe éducative DIU EIL, Université de Nantes.
- Cours OpenClassrooms pour l'idée de la base de données d'un disquaire : Découvrez le framework Django.
- Livre *Numérique et Sciences Informatiques, 24 leçons, Terminale*, T. BALABONSKI, S. CONCHON, J.-C. FILLIATRE, K. NGUYEN, éditions ELLIPSES.
- Livre *Prepabac NSI, Tle*, G. Connan, V. Petrov, G. Rozsavolgyi, L. Signac, éditions HATIER.
- Cours de Gilles Lassus sur le modèle relationnel

Germain BECKER, Lycée Mounier, ANGERS

Voir en ligne : info-mounier.fr/terminale_nsi/base_de_donnees/modele-relationnel