Modèle relationnel

Considérons une activité banale : l'emprunt d'un livre dans une médiathèque municipale. Alice trouve le livre qu'elle souhaite emprunter et approche d'une borne d'emprunt automatique. Elle scanne le code barre de sa carte de bibliothèque et des informations la concernant apparaissent : « Alice, aucun livre en cours de prêt ». Elle peut ensuite scanner le code barre apposé sur le livre. Les informations sont alors mises à jour pour afficher la liste des livres empruntés : « *La théorie de l'information*, A. Bellanger, Éd. Folio, 2014, ISBN 978-2070456260 ». Sur la borne d'à côté, Basile vient rendre ses livres. Il scanne lui aussi sa carte et les informations le concernant apparaissent à l'écran : « Basile, un livre en cours de prêt ». Basile peut alors scanner le code barre du livre qu'il rapporte et ce dernier est supprimé de la liste.

Cette simple activité nous permet d'illustrer les caractéristiques d'un système d'information, c'est-à-dire d'un système technique (ici informatique) et humain permettant de gérer de l'information. Que pouvons-nous dire sur le système informatique sous-jacent ? En premier lieu, il contient une description d'objets (les livres), de personnes physiques (les usagers) et des processus (l'emprunt et la restitution de livres). Le système ne contient cependant qu'une « approximation » de la réalité : d'Alice et Basile, il ne retient que leur nom et prénom, éventuellement leur date d'inscription et les livres en cours d'emprunt ; d'autres informations telle que leur taille, la couleur de leurs yeux ou leur plat préféré ne sont pas connus du système. De même pour les livres : seuls le titre, les auteurs, l'éditeur et l'ISBN (le numéro de série international unique attribué à chaque ouvrage publié) sont mémorisés. Le système ne retient donc que ce qui est nécessaire au bon fonctionnement des processus dont il facilite le déroulement : l'emprunt et le rendu de livres. Une telle « approximation » de la réalité s'appelle une modélisation.

Avant de préciser la façon de modéliser les données, finissons de passer en revue les différentes caractéristiques du système d'information de la bibliothèque. Outre le stockage des données de chaque objet considéré (livres, usagers, emprunts, etc.), ce dernier doit garantir l'absence d'erreur. Par exemple, si Basile repose son livre en rayon sans passer par une borne et qu'Alice tente de l'emprunter, le système doit détecter le problème et signaler que Basile a toujours le livre en sa possession. De manière symétrique, si Alice rend un livre, alors le système doit effectivement garantir que quelqu'un d'autre peut l'emprunter. Il y a un autre aspect que nous n'avons pas évoqué : les documentalistes doivent pouvoir ajouter de nouveaux livres dans le système ou en retirer, par exemple quand ces derniers ont été perdus ou sont en trop mauvais état. Cependant, tout un chacun ne doit pas pouvoir modifier la liste des livres du système. Ce dernier doit donc permettre différents niveaux d'accès aux données. Enfin, on souhaite que quelle que soit la borne utilisée, la procédure soit la même. Les données doivent donc être les mêmes. Il semble donc raisonnable de penser que ces données sont stockées dans un même endroit et que les bornes ne sont que des clients permettant de consulter et modifier ces données de manière contrôlée.

Ces caractéristiques, et bien d'autres, sont celles d'un *système de gestion base de données*. Ce dernier est le composant logiciel principal autour duquel est construit le système d'information. Nous reviendrons sur ces aspects tout au long des chapitres suivants. Attaquons nous d'abord au problème de la modélisation de données.

A. Le modèle relationnel

De manière générale, un modèle de données est une représentation (mathématique ou informatique) de concepts que l'on souhaite étudier. Les intérêts d'une telle modélisation sont nombreux. Un modèle permet d'énoncer des *propriétés* de ces données en termes *logiques*. Il permet aussi de *programmer* des

processus réels complexes (par exemple, emprunter un livre) sous forme d'opérations élémentaires sur les données.

L'un des modèles de données les plus populaires est le *modèle relationnel*. Ce dernier a été défini en 1970 par l'informaticien Américain Edgar F. Codd (1923–2003), alors qu'il était employé par IBM. Dans ce modèle, un objet modélisé (on parle d'*entité*) est représenté par un n-uplet de valeurs *scalaires* et les collections d'objets par des ensembles de n-uplets. Par exemple, le livre emprunté par Alice peut être représenté par le quintuplet suivant :

```
('La théorie de l'information', 'A. Bellanger', 'Folio', 2014, '978-2070456260')
```

Dans ce dernier, quatre composantes sont des chaînes de caractères (le titre, les auteurs, le nom de l'éditeur et l'ISBN) et une composante est un entier (l'année d'édition). L'ensemble des livres de la bibliothèque peut alors être représenté par un *ensemble* `Livre` de tels n-uplets :

```
Livre = {
  ('La théorie de l'information', 'A. Bellanger', 'Folio', 2014, '978-2070456260')
  ('Les affiinités électives, 'Goethe', 'Folio', 2011, '978-2070372379')
  ('Pantagruel', 'Rabelais', 'Folio', 2001, '978-2070363872')
  ('Lettres Persanes', 'Montesquieu', 'Folio', 2015, '978-2070429349')
  ('Discours sur la servitude volontaire', 'La Boétie', 'GF Flammarion', 1983,
   '978-2080703941')
  ...
  }
```

On appelle un tel ensemble une *relation*. Une autre relation est l'ensemble *Emprunt* des livres actuellement empruntés :

```
Emprunt = {
  ('Pantagruel', 'Alice', 01/04/2020),
  ('Le Bouclier Arverne', 'Charlie', 15/05/2020),
  ('Le Guide du routard galactique', 'Charlie', 08/04/2020),
  ...
  }
```

Chaque relation se conforme à un schéma. Ce dernier est une description qui indique pour chaque composante des n-uplets de la relation leur *nom* et leur *domaine*. On appelle une telle composante un *attribut*. Par exemple, les éléments de la relation Livre possèdent cinq attributs :

titre : le titre du livre, une chaîne de caractères
auteur : le ou les auteurs du livre, une chaîne de caractères
éditeur : l'éditeur du livre, une chaîne de caractères
année : l'année de publication du livre, un entier naturel
isbn : l'identifiant unique du livre, une chaîne de caractères

Une manière plus compacte de noter un schéma pour une relation est la suivante :

Livre(*titre* String, *auteur* String, *éditeur* String, *année* Int, *isbn* String)

Une base de données est un ensemble de relations. Par exemple, la base de données de la médiathèque peut être composée de trois relations, *Livre*, *Emprunt* et *Usager*. Cette dernière, que nous n'avons pas encore décrite, recense toutes les personnes inscrites à la médiathèque. Elle pourrait avoir le schéma suivant :

Usager(*nom* String, *prénom* String, *code_barre* String)

Dans une telle relation, on stocke les noms et prénoms des personnes ainsi que la suite de caractères que représente le code barre de leur carte (cette information est nécessaire pour retrouver la personne à partir de son code barre imprimé sur sa carte). Par extension, on appelle schéma d'une base de données l'ensemble des schémas des relations constituant la base.

B. Modélisation relationnelle des données

Principes généraux

La théorie et la pratique des bases de données relationnelles est un domaine de recherche toujours actif, mais qui s'est fortement développé et complexifié depuis ses balbutiements dans les années 1970. Nous ne proposons pas d'étudier ici cette théorie, qui irait bien au-delà du programme de terminale et demande des connaissances en mathématiques et en logique, notamment en théorie des ensembles. Nous allons cependant essayer de donner les grands principes sous-jacents en les illustrant par des exemples.

Le fil conducteur de cette section sera l'exemple de la médiathèque que nous allons affiner au fur et à mesure. 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 en donnant leur schéma, 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.

Contraintes d'intégrité

Une contrainte d'intégrité est une propriété logique, préservée à tout instant par la base de données et qui garantit la cohérence des données. En d'autres termes, une contrainte d'intégrité est un invariant de la base de données. On distingue quatre grandes catégories de contraintes d'intégrité jouant des rôles complémentaires.

Contrainte de domaine

Considérons en premier lieu la relation des usagers, que l'on va rendre un peu plus réaliste. Normalement, lorsqu'une personne s'inscrit dans une médiathèque, elle donne non seulement son nom et son prénom, mais aussi son adresse. Cette dernière peut servir à déterminer son tarif d'abonnement (par exemple, les gens qui habitent dans la ville de la médiathèque ont un tarif préférentiel) ou à envoyer des courriers de relance. De nos jours, il est aussi commun de stocker une adresse e-mail. Lors de l'inscription, la personne obtiendra une carte avec un code barre et ce dernier doit donc également être associé à l'usager. Chacune des propriétés que l'on souhaite renseigner pour un usager correspond à un attribut pour la relation Usager. Il peut être compliqué de décider si une propriété complexe (par exemple l'adresse) doit être séparée en plusieurs attributs. Ce découpage est important car il va influencer la façon dont on va interroger les données stockées. Nous proposons la modélisation suivante :

Usager(*nom* String, *prénom* String, *adresse* String, *cp* String, *ville* String, *email* String, *code_barre* String)

On peut s'étonner ici que les attributs soient tous représentés par des chaînes de caractères. Le code postal (attribut *cp*) et le code barre (attribut *code_barre*) sont a priori des suites de chiffres. On pourrait donc vouloir les représenter par des entiers. Le problème est que ces suites de chiffres peuvent commencer par le chiffre 0, non significatif pour les entiers. Ainsi, le code postal de la ville d'Oyonnax est 01100. Le stocker comme un entier reviendrait à stocker l'entier 1100, ce qui peut être problématique par la suite.

Dans un premier temps, nous allons nous abstraire lors de la modélisation des aspects techniques propres aux systèmes de gestion de bases de données et au langage SQL. Nous proposons donc d'utiliser des domaines génériques, inspirés des types de données des langages de programmation :

String représente les chaînes de caractères ;
Int représente les entiers signés (qu'on suppose de taille arbitraire, comme les entiers de Python) ;
Boolean représente les valeurs booléennes True et False ;
Float représente les nombres flottants ;
Date représente des dates (jour/mois/année) ;
Time représente des instants (heure : minute : seconde) ;

Le choix des domaines est particulièrement important, car il doit répondre à deux impératifs :

- permettre de représenter exactement et sans perte d'information toutes les valeurs possibles pour un attribut ;
- limiter autant que possible la saisie de valeurs illégales ou mal formées.

Les contraintes de domaines sont toutes les propriétés que le domaine d'un attribut va permettre de garantir. Par exemple, si l'on souhaite représenter un choix binaire (abonné/non abonné, présent/absent, etc.) le domaine **Boolean** ne contenant que deux valeurs permet de garantir la contrainte que l'attribut ne peut pas avoir de troisième valeur. De même, si l'on souhaite stocker l'âge d'une personne, le domaine **Int** est plus approprié que **String**, ce dernier permettant de représenter des valeurs qui ne sont clairement pas des âges (par exemple 'Toto'). En revanche, l'utilisation seule du domaine **Int** ne permet pas de garantir l'absence de données incohérentes dans la base. Ainsi, des valeurs telles que -10 ou 45000 sont des entiers valides mais a priori pas des âges raisonnables pour des personnes.

Contrainte d'entité

La contrainte d'entité permet de s'assurer que chaque élément d'une relation est unique et identifie une entité de manière non ambiguë. Considérons une relation Usager où l'on n'aurait stocké que les noms et prénoms des personnes. Il est relativement courant que deux personnes portent le même nom de famille et le même prénom, surtout dans des villes assez peuplées. Si ces deux personnes s'inscrivent toutes les deux à la médiathèque, la relation Usager contiendra deux couples identiques, ce qui poserait problème. Pour éviter cette situation, il faut s'assurer, lors de la modélisation, que pour chaque entité d'une relation, un attribut ou en ensembled'attributs permet d'identifier cette entité de manière unique. La relation Usager définie à la section précédente respecte bien cette contrainte. En effet. le code barre est unique (car la carte associée est donnée à la personne au moment où elle s'inscrit et on peut supposer que le système ne réutilise pas un numéro déjà attribué). On appelle un tel ensemble d'attributs la clé primaire de la relation. On indique cela en les soulignant dans le schéma :

Usager(*nom* String, *prénom* String, *adresse* String, *cp* String, *ville* String, *email* String, *code_barre* String)

Notons que les attributs pouvant jouer le rôle de clé primaire peuvent ne pas être uniques. Dans notre cas, on pourrait imaginer que l'adresse *email* puisse aussi jouer le rôle de clé primaire. Ou encore, la combinaison du *nom*, du *prénom*, de *l'adresse*, du code postal (attribut *cp*) et de la *ville* ensembles jouent le rôle de clé primaire. Mais attention, le choix de la clé primaire est important, car le système garantira son unicité (pour préserver la contrainte d'entité). En fonction des choix, certains cas d'utilisation seront impossibles.

Par exemple, si l'on choisit l'adresse *email*, alors il ne sera pas possible pour un parent de renseigner son adresse e-mail dans le compte de ses enfants, l'adresse devant être unique pour chaque entité de la relation. De même, si on choisit la combinaison des noms, prénoms, adresses, code postal et ville, il faut s'assurer que l'adresse est suffisamment précise pour distinguer deux personnes homonymes qui habiteraient le même immeuble de la même ville.

Dans le cadre de la relation *Livre*, on peut redonner le schéma initial en utilisant l'ISBN comme clé primaire :

Livre(*titre* String, *auteur* String, *éditeur* String, *année* Int, *isbn* String)

Encore une fois, cette modélisation est raisonnable. Utiliser uniquement le dire ou le couple titre et auteurs comme clé primaire aurait posé problème si la bibliothèque avait en stock le même livre dans deux versions différentes (par exemple en livre de poche et en grand format illustré). *L'isbn* lui-même peut ne pas être suffisant si la médiathèque dispose de plusieurs copies du même ouvrage. Il faudrait alors distinguer ces différentes copies dans la base de données, par exemple en ajoutant un identifiant unique sur chaque exemplaire pour servir de clé (par exemple un code barre unique ajouté par la médiathèque).

Contrainte de référence

Les clés primaires ne permettent pas seulement de distinguer les entités de manière unique. Elles permettent aussi de servir de référence dans une autre relation. Intéressons nous maintenant à la relation Emprunt. L'exemple que l'on a donné en introduction est naïf, car il stocke dans cette table le prénom

de l'utilisateur, le titre du livre et la date de rendu. On pourra plutôt définir la relation Emprunt avec le schéma suivant :

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

Ici, *code_barre* et *isbn* sont des clés étrangères. Cela signifie que la valeur de l'attribut *code_barre* dans la relation *Emprunt* doit être l'une des valeurs existantes pour cet attribut dans la relation *Usager*. De même, *isbn* doit correspondre à un isbn existant dans la relation *Livre*. Cette contrainte permet de garantir que la relation *Emprunt* ne mentionne que des livres et des usagers connus de la base de données. Elle permet d'éviter de rajouter des valeurs fictives ne correspondant à aucun utilisateur ou à aucun livre. De manière plus importante, elle empêche aussi de supprimer des entités des relations *Livre* et *Usager*. En particulier, si la relation *Emprunt* contient l'entité `('123456789', '978-2340033641', 07/03/2020)`, alors retirer l'utilisateur dont le code barre est `'123456789'` de la relation *Usager* est une violation de la contrainte de référence, car la valeur `'123456789'` ne serait plus celle d'une clé primaire dans la relation *Usager* (et d'un point de vue pratique, cela permet de s'assurer qu'avant de désinscrire une personne, cette dernière a rendu tous ses livres).

Une autre remarque que l'on peut faire sur le schéma *Emprunt* est que *l'isbn* est déclaré comme clé primaire de la relation. Cela implique qu'un même livre ne peut apparaître dans deux entités distinctes. Là encore, on voit que la contrainte nous permet de garantir la cohérence des données : il n'est pas possible qu'un même livre soit emprunté par deux usagers en même temps.

Une dernière observation importante sur l'utilisation des clés étrangères est qu'elle permet de créer des associations multiples entre entités de différentes relations. Ici, on peut associer au même utilisateur plusieurs livres en cours d'emprunt. En effet, *code_barre* n'étant qu'une clé étrangère, il n'a pas à être unique dans la relation Emprunt : un même utilisateur a le droit d'emprunter plusieurs livres. En d'autres termes, nous pouvons par ce moyen associer à un utilisateur une liste de livres. Nous pouvons utiliser cette technique pour pallier un défaut de conception de la relation Livre. En effet, dans cette dernière, nous avons représenté les auteurs du livre comme un chaîne de caractères dans laquelle est écrite la liste des auteurs. Bien que cette modélisation fonctionne, elle ne permet pas d'exprimer certaines contraintes sur les données, par exemple qu'un même auteur n'apparaît pas deux fois pour le même livre. En effet, les chaînes de caractères étant arbitraires, on peut y saisir n'importe quoi. L'utilisation de clés étrangères va nous permettre de remédier à ce problème. On simplifie dans un premier temps le schéma de la relation Livre pour ne plus mentionner les auteurs :

$$\textit{Livre}(\textit{titre}\ \text{String}, \textit{éditeur}\ \text{String}, \textit{année}\ \text{Int}, \underline{\textit{isbn}}\ \text{String})$$

On peut ensuite créer une nouvelle relation Auteur ayant le schéma suivant :

$$\textit{Auteur}(\underline{\textit{a\_id}}\ \text{Int}, \textit{nom}\ \text{String}, \textit{prénom}\ \text{String})$$

Ici, l'attribut *a_id* est un identifiant d'auteur unique, associé à l'auteur lorsque l'employé de la médiathèque rajoute un nouvel auteur dans la base. La relation *Auteur* pourrait par exemple être la suivante :

```
{ (0, 'Goscinny', 'René'), (10, 'Rabelais', 'François'),
  (2, 'Bellanger', 'Aurélien'), (42, 'Montesquieu', 'Charles Louis'),
  (23, 'Uderzo', 'Albert'), ... }
```

La seule contrainte sur les identifiants est qu'ils doivent être uniques. Il n'y a en particulier aucune notion d'ordre. Munis des relations *Livre* et *Auteur*, nous pouvons associer des auteurs à des livres au moyen de la relation *Auteur_de* :

$$\textit{Auteur\_de}(\underline{\textit{a\_id}}\ \text{Int}, \underline{\textit{isbn}}\ \text{String})$$

Cette relation associe des auteurs à des livres. Les attributs *a_id* et *isbn* sont des clés étrangères faisant référence aux relations *Auteur* et *Livre* respectivement. Le couple de ces deux attributs forment la clé primaire de la table *Auteur_de*. Cela empêche qu'un même auteur et un même livre apparaissent deux fois dans la relation et donc qu'un même auteur soit mentionné deux fois pour le même ouvrage. En revanche, rien n'empêche qu'un même auteur apparaisse plusieurs fois pour des ouvrages différents, ou que différents auteurs apparaissent pour le même ouvrage. Attention cependant, si un isbn n'apparaît pas dans cette relation, c'est que le livre correspondant n'a pas d'auteur. Les contraintes de clé primaires et étrangères ne permettent pas d'empêcher ce cas de figure.

Contraintes utilisateurs

Nous terminons ce tour d'horizon des contraintes d'intégrité par les contraintes utilisateurs (parfois appelées contraintes métier). Ces dernières sont toutes les contraintes d'une relation qu'on ne peut exprimer par les trois précédentes. Un exemple de contrainte utilisateur est qu'un âge de personne doit être positif et inférieur à 200. Pour notre médiathèque, une autre contrainte utilisateur pourrait être que la chaîne de caractères représentant l'email contienne un et un seul caractère « @ ». Ces contraintes, liées à l'utilisation que l'on veut faire de la base de données, sont importantes mais difficilement exprimables dans la syntaxe très simple des schémas. On veillera donc à en faire une description précise en français. Nous verrons dans le chapitre suivant que les systèmes de gestion de bases de données proposent une syntaxe pour écrire certaines de ces contraintes.