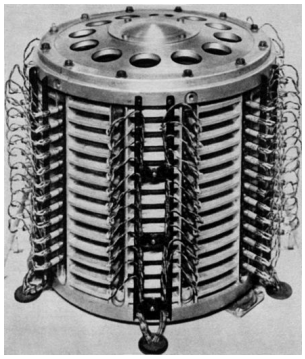
Chapitre 7 : Modèle relationnel

L'an dernier, nous avons travaillé sur le traitement de données grâce aux tableaux CSV et nous avons rapidement évoqué un autre format qu'est le JSON. Les tableaux CSV ou le JSON permettent l'échange de données via un simple fichier texte ce qui constitue un atout important. Toutefois, leur principal problème réside dans le fait qu'ils ne peuvent permettre le stockage de bases de données de grandes tailles : en effet, il parait impensable de dépasser les 100 000 entrées dans un tableur sans avoir des problèmes de vitesse d'exécution et de crash répétitifs.

Est-ce possible d'atteindre 100 000 entrées? Oui… IMDb compte par exemple 6,5 millions d'entrées à l'heure actuelle !

I. Introduction

1) Un peu d'histoire



Dans les premiers temps de l'informatique (1930 jusqu'en 1950), toutes données générées par des mesures ou des calculs informatiques étaient enregistrées sur un support extérieur à l'ordinateur. Il en existait alors de deux types : les cartes perforées ainsi que les bandes magnétiques (un tambour magnétique ci-contre).

Les **premiers disques durs modernes** apparaissent en 1960 et permettent de stocker directement de l'information dans un ordinateur au lieu de les stocker sur des bandes magnétiques externes. Cette invention, permettant une manipulation souple ou très rapide des données, pousse à l'apparition au milieu des années 60 du terme "**base de données**" (ou database en anglais). L'idée des bases de données a été de proposer un système de stockage, de classement et de manipulation efficace de l'information via un disque dur. Historiquement, les bases de données ont vu leur envol en 1960 dans le cadre du programme Apollo, visant à envoyer un homme sur la Lune.

Rem : l'organisation des bases de données actuelles découlent des travaux d'**Edgar F. Codd** qui en **1970**, fait un lien direct entre une structure mathématique appelé "**algèbre relationnelle**" et les opérations effectuées sur des tables (jointures, clés primaires et secondaires etc.).

Avec le développement d'internet, la quantité de données à stocker et à accéder a littéralement explosé. Aujourd'hui, la plupart des sites internet utilisent au moins une base de données : les frameworks Python Flask et Python Django utilisent par exemple des bases de données pour gérer un site web côté serveur. Les bases de données jouent un rôle

fondamental dans notre monde devenu numérique où il est extrêmement facile d'accéder à l'information. Voilà pourquoi nous allons cette année les étudier.

2) Système de gestion de bases de données

Jusqu'en 1960, les informations étaient enregistrées dans de de multiples fichiers manipulées par des logiciels dédiés (un base de données = un logiciel !).

Pour éviter les problèmes de compatibilité, d'encodage et de propriété, IBM (conjointement avec Rockwell) met sur le marché le logiciel Information Management System (IMS). Avec ce premier "système de gestion de bases de données" (ou **SGBD**), les informations sont enregistrées dans des bases de données organisées de manière hiérarchique (les données sont organisées sous forme de dictionnaires et les enregistrements sont reliés entre eux à l'aide de listes chainées ou d'arbres). Mais l'utilisation de ces logiciels est encore complexe en raison de l'absence d'un modèle simple de description de données.

Ce n'est qu'après la mise en place du modèle relationnel par Edgar F. Codd que les bases de données prennent une autre dimension : les SGBD permettent alors une mise en place et une utilisation facile de bases de données de grande taille.

System R d'IBM est le premier SGBD à proposer une implémentation du langage SQL, traduction en langage de description du modèle relationnel. Le premier SGBD commercial d'ampleur est **Oracle** (Oracle Corporation) qui pendant plus de 20 ans reste la seule option pour mettre en place facilement des bases de données relationnelles.

Avec le développement d'Internet à la fin des années 1990, le besoin de solutions moins onéreuses qu'Oracle pousse à l'apparition de nombreux concurrents OpenSource (MySQL en 1995, SQLite en 2000). Ces SGBD libres de droits promettent les mêmes performances qu'Oracle tout en se fondant sur le même modèle théorique.

Nous étudierons le fonctionnement de ces SGBD, applications du modèle relationnel, dans un prochain chapitre et nous nous concentrerons ici sur le modèle relationnel théorique.

II. Modélisation relationnelle des données

1) Modèle relationnel

Un modèle de données est une représentation (mathématique ou informatique) de concepts que l'on souhaite étudier. Un tel modèle théorique permet d'énoncer les propriétés de ces données en terme **logique**. Il permet également de formaliser des processus réels complexes (dans ce chapitre, l'emprunt d'un livre) sous forme d'opérations élémentaires simples sur des données.

Rem : une structure de données indique la manière dont on va organiser les données dans un langage de programmation données. À un modèle de données fixé correspond en général plusieurs structure de données.

Pour les bases de données, un modèle de données très populaire est le **modèle relationnel**.

Définitions :
- Dans le modèle relationnel, un objet (appelé aussi entité) est représenté par un n-uplet de valeurs scalaires et les collections d'objets par des ensembles de n-uplets.
- Ces ensembles de n-uplets sont appelés **relations**.
- Une base de données est un ensemble de **relations**.

Exemple :
Erwan emprunte un livre à la bibliothèque.
Ce livre peut être représenté par un 5-uplet (appelé aussi quintuplet) suivant :
```
('Into the Wild', 'J. Krakauer', 'Villard', 1996, '9780679450252')
```
Les valeurs sont dites scalaires car elles ne sont pas constituées d'autres sous-éléments (tableau par exemple).

Ce livre appartient à la bibliothèque qui a une collection de livres. On peut donc créer la **relation Livre** qui s'écrirait ainsi ;
Livre = {
```
('Into the Wild', 'J. Krakauer', 'Villard', 1996, '9780679450252'),
('The push', 'T. Caldwell', 'Penguin Books', 2017, '9781405924740'),
('Python pour les nuls', 'J.-P. Mueller', 2016, '9782754083218'),
…
}
```

Exercice :
De nombreux livres ont été empruntés à la bibliothèque.
- Erwann a emprunté "Into the Wild" le 12 mai 2020,
- Antoine a emprunté "The push" le 20 juin 2020,
- Victoire a emprunté "Le guide du routard galactique" le 23 juin 2020.

Ecrire une **relation Emprunt** qui donne l'ensemble des livres actuellement empruntés. On fera figurer les informations nécessaires dans la relation **Emprunt**.

Vous avez sans doute remarqué que les informations du n-uplet sont toujours organisées de la même manière et que les types sont soit des entiers, soit des chaines de caractères. Cette organisation rigoureuse est appelée un schéma.

Définitions :

- Un **schéma** est une description qui indique pour chaque composante des n-uplets de la relation leur nom et leur domaine. Chaque relation se conforme à un schéma.
- Une telle composante (nom et domaine) est appelée un **attribut**.

Exemple :

Pour les livres de notre bibliothèque, la relation Livre possède cinq attributs :

- titre : titre de l'ouvrage, chaine de caractères
- auteur : auteur de l'ouvrage, chaine de caractères
- éditeur : éditeur du livre, chaine de caractères
- année : année de publication, entier naturel
- isbn : identifiant unique du livre, chaine de caractères

Un schéma se note de manière plus compacte :
Livre(*titre* `String`, *auteur* `String`, *éditeur* `String`, *année* `Int`, *isbn* `String`)

Exercice :

Donnez le schéma de la relation **Emprunt** définie ci-dessus.
Donnez le schéma de la relation **Usager**. Cette relation aura pour attributs le nom de l'usager, le prénom de l'usage ainsi qu'un code barre. Les types sont à déterminer.

Les relations **Usage**, **Emprunt** et **Livre** forment la base de données de la bibliothèque.

Rem : il existe d'autres modèles que le modèle relationnel qui est assez rigide. Par exemple, un modèle de données basé sur les graphes semble plus approprié pour représenter un réseau routier ou social. La difficulté actuelle dans le traitement et le stockage des données est de faire cohabiter différents types de modèles afin de s'adapter à nos questionnements.

2) Contraintes d'intégrité

Au travers de l'exemple de la bibliothèque que nous allons peu à peu raffiner, nous allons dans cette section les grands principes de la modélisation relationnelle des données.

Propriété :

La modélisation de données se décompose en trois grandes étapes :

- déterminer les **entités** que l'on souhaite manipuler ;
- modéliser les ensembles d'entités comme des **relations** en donnant leur **schéma** et en choisissant correctement les domaines de chaque **attribut** ;
- définir les **contraintes** de la base de données, c'est à dire l'ensemble des propriétés logiques que nos données doivent toujours vérifier pour rester cohérente.

Rem : il est important de passer du temps à concevoir ces étapes. En effet, il est **très couteux** en temps (et en énergie) de modifier une base de données déjà existante (attributs manquants, relations à modifier, nouvelles contraintes non remplies etc.).

a. Contrainte de domaine :

Définition : Les contraintes de domaine restreignent les valeurs d'un attribut à celles d'un domaine donné et évitent que l'on puisse donner des valeurs illégales.

Types génériques :
Pour décrire les contraintes de domaine, on définit quelques types classiques :

- `String` : chaines de caractères
- `Int` : entier
- `Boolean` : valeurs booléennes
- `Float` : nombres flottants
- `Date` : date au format jour/mois/année
- `Time` : instants au format heure : minute : seconde

Exemple :
Dans notre exemple de la bibliothèque, en plus du prénom, du nom et d'un code, on peut souhaiter enregistrer l'adresse physique, l'adresse email et la date de naissance pour la relation **Usager**.
La contrainte de domaine sur l'adresse email sera "Chaine de caractères" : String
La date de naissance sera contrainte à être une "Date" : Date

Exemple :
La contrainte de domaine sur l'adresse physique est plus problématique.
Comment doit-on stocker l'adresse physique ?
Quels sont les contraintes de domaine sur ces quatre attributs ?

Rem : Les contraintes de domaine évitent les valeurs illégales. Par exemple, on ne peut pas attribuer la valeur 'toto' à une date de naissance car 'toto' est de type `String`. Par contre, rien n'empêche de donner -100 ou 40000 à l'âge d'une personne d'un type `Int`.

b. Contrainte d'entité :

Définition : Les contraintes d'entité garantissent l'unicité de chaque entité d'une relation. La **clé primaire**, garante de ces contraintes d'entité, est un ensemble d'attributs identifiant chaque entité de la relation de manière unique.

Rem : souvent la clé primaire est un unique attribut créé pour l'occasion. Plus rarement, il est composé de plusieurs attributs.

Exemple :
Dans notre bibliothèque, les usagers ont un nom, un prénom et un code barre.

- Le nom (ou le prénom) ne peut pas être une clé primaire car deux personnes d'une même ville peut avoir le même nom (ou pire le même prénom!) ;
- Pourquoi le nom et le prénom ensemble ne peuvent pas constituer une clé primaire ?
- Expliquez pourquoi seul un code barre unique peut constituer une bonne clé primaire et garantir les contraintes d'entité.
- Expliquez pourquoi nous ne pouvons pas nous passer du code barre unique, même si nous rajoutons des informations sur l'usager comme dans l'exemple du paragraphe a). Dans chacun des cas, expliquez la limitation liée aux contraintes d'entité.

Notation : La clé primaire est soulignée d'un trait plein. Par exemple :
Usager(*nom* **`String`**, *prénom* **`String`**, *adresse* **`String`**, *cp* **`String`**, *ville* **`String`**,
email **`String`**, *code_barre* **`String`**)

Exercice :

a) Quelle est la clé primaire du schéma **Livre** ?
b) Quelle pourrait être la clé primaire du schéma **Emprunt** défini précédemment ? On se réfèrera à des situations concrètes pour trouver la clé primaire de ce schéma.

c. Contrainte de référence :

Définition : Les contraintes de référence assurent la cohérence du lien entre deux relations A et B. La **clé étrangère**, garante de ces contraintes de référence, est un ensemble d'attributs d'une table A qui sont aussi une **clé primaire** d'une autre table B.

Exemple :
Comme vous l'avez sans doute compris dans l'exercice précédent, notre première approche de la relation **Emprunt** était très naïve. Il n'existe pas de bonne clé primaire. On va donc redéfinir cette relation en proposant le schéma suivant :

Emprunt(*code_barre* **`String`**, *isbn* **`String`**, *retour* **`Date`**)

code_barre et *isbn* sont définies comme des clés étrangères : *code_barre* doit être une **valeur valide** de la relation Usager.
Cela garantit deux propriétés :

a) la relation Emprunt ne mentionne que des usagers présents dans la base de données.
b) on ne peut pas supprimer une entité de la relation Usager si celle-ci n'a pas rendu tous ses livres.

L'usager est défini comme une clé secondaire et non primaire. Un usager va donc pouvoir emprunter une liste de plusieurs livres représentés par leurs isbn.

Exercice 1 :

a) Expliquez pourquoi l'isbn est défini comme la clé primaire de la relation Emprunt.
b) Avec vos propres mots, expliquez précisément la propriété b) de l'exemple ci-dessus.

Exercice 2 :

La relation Livre écrite comme Livre(*titre* **`String`**, *auteur* **`String`**, *éditeur* **`String`**, *année* **`Int`**, *isbn* **`String`**) est très mal définie.

En effet, si un livre a plusieurs auteurs, on peut se retrouver avec une liste d'auteurs dans une seule chaine de caractères. Par exemple, deux livres d'Axel Kahn :

Livre = {

```
('Copies conformes, le clonage en question', 'A. Kahn et F.
Papillon', 'Nil', 1998, '2702816738'),
('Entre deux mers', 'A. Kahn', 'Stock', 2015, ' 9782234079168'),
}
```

On va donc créer deux nouvelles relations **Auteur** et **Auteur_de**. La relation **Auteur_de** relie un auteur au livre dont celui-ci est un des auteurs.

- Donnez le schéma de la relation **Auteur**. On soulignera en trait plein la clé primaire.
- Expliquez pourquoi la relation **Auteur_de** a pour schéma :

Auteur_de(*auteur_id* **`Int`**, *isbn* **`String`**)

d. Contraintes utilisateurs (ou métier) :

> Définition : Les contraintes utilisateurs sont toutes les contraintes que l'on ne peut exprimer par les trois autres contraintes.

Exemple :

Pour une adresse email, une contrainte utilisateur serait que la chaine de caractères contienne un @ .

Exercice :

Quelle serait la contrainte utilisateur associée à l'âge d'un usager ?
Quelle serait la contrainte utilisateur associée à un code postal ?