Cours de bases de données – Livraison 1

Pascal Ostermann – pasccal@orange.fr

30 mars 2021

Avertissement

Ce texte ne se substitue au cours qu'en raison des circonstances. Écrit dans l'urgence, il n'a pas la rigueur d'un bon ouvrage sur le sujet ; sans intervention des étudiants, il n'a pas l'interactivité d'un cours ; et je souhaite voir ce document disparaitre après la crise présente. Il s'agit globalement de la transcription du cours tel qu'il aurait pu être donné cette année, et il est donc divisé en livraisons, correspondant grosso modo à une séance de cours. Dès lors, si vous lisez une de ces livraisons en beaucoup moins d'une heure quatre-cinq, c'est sans doute que vous l'avez trop vite lue. Si par contre, il vous y faut beaucoup plus de temps, c'est que j'ai merdé, et je vous prie de me le signaler au plus vite.

Introduction / niveaux ANSI

Définition

La **base de données** (BD – en anglais, c'est database donc DB) d'une application est l'ensemble de ses données *persistantes* : celles qui continuent d'exister lorsque l'application ne travaille pas.

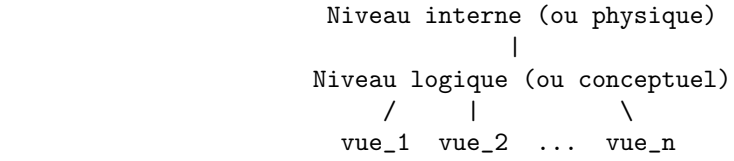

Figure 1 – Les niveaux ANSI (1975)

À l'opposé des données que vous avez pu faire évoluer dans un programme, les informations contenues dans une BD traitent du monde réel. Deux applications différentes peuvent alors partager des infos communes. Imaginons par exemple deux applications utilisées dans une même école, définissant d'une part les cours,

d'autre part les salaires des professeurs : il faut clairement que les deux listes de profs soient cohérentes. Dans ce genre de situation, on parlera de plusieurs **vues**, restrictions de la BD contenant les infos nécessaires à chacune des applications. Et la cohérence de ces informations sera assuré par la définition d'un unique **niveau logique**.

Par ailleurs ces informations sont précieuses, parfois parce que nous désirons les exploiter de manière directe (par exemple le fichier des clients d'une entreprise), au minimum pour ce que les recueillir nous a coûté. Il faut donc veiller à la *sécurité* de ces données, que nous ne voulons ni nous faire voler (confidentialité) ni laisser se déformer (conformité).

— Pour la *conformité*, la seule manière de l'obtenir – afin de lutter par exemple contre un rançongiciel, remplaçant les données de votre ordinateur par une version cryptée... et le pirate ne vous fournira la clef de cryptage qu'après paiement d'une rançon [1] – est de FAIRE DES SAUVEGARDES FRÉQUENTES, de préférence sur un support qui ne reste pas en permanence relié à l'ordinateur.

— La *confidentialité* ne peut s'obtenir qu'en chiffrant les données. C'est ce qui explique la différence entre **niveau interne**, où les valeurs cryptées sont visibles sur le système d'exploitation ; et **niveau logique** où elles sont lisible, mais auquel on ne peut accéder que de l'intérieur de la base de données.

Digression : comment ranger les informations en mémoire

Dans ce contexte de données cryptées dont un intrus malveillant pourrait observer l'évolution, on peut éliminer certaines structures de données.

— Les *listes chaînées* ne conviennent clairement pas. À chaque nouvelle donnée introduite, le système de gestion de bases de données (que j'abrègerai plus loin en SGBD) devrait y réserver un espace, et en avertir le système d'exploitation. Notre intrus risque d'apprendre assez vite que telle info est stockée à tel endroit. Même si cette donnée est cryptée, c'est une brèche de sécurité.

— Mieux d'user des *tableaux*, où la réservation d'espace se passe au tout début. Tout ce que saura le système d'exploitation est que telle donnée est quelque part dans le tableau, au milieu de milliers d'autres. Encore n'est-il pas très malin de ranger les données dans un ordre trop facilement prévisible, comme un ordre alphabétique. Cela permettrait là encore à l'intrus de relier une info à sa valeur cryptée.

— La meilleure solution est donc d'utiliser une *table de hachage* : un tableau où la valeur i est stockée à l'adresse h(i) – h étant une *fonction de hachage*. À noter que i n'est pas la valeur totale de la donnée, mais ce

1. Les amateurs de séries TV trouveront une discussion intéressante des effets d'un tel malware (Faut-il payer ou pas ?) sur un cabinet d'avocats dans The Good Wife, saison 6, épisode 5 "Shiny objets." L'actualité pourra vous en fournir des exemples réels, puisque depuis la crise de la Covid, les hôpitaux ont souvent fait l'object de telles attaques.

qu'on appellera plus tard un **identifiant** en entité-association, et la **clef primaire** dans le modèle relationnel.

Le modèle Entité-Association

Entity-relationship en anglais, ce qui fait que beaucoup traduisent par entité-relation. Mais en enseignement, je préfère employer un mot différent pour les *relationships* et pour les *relations* du modèle relationnel.

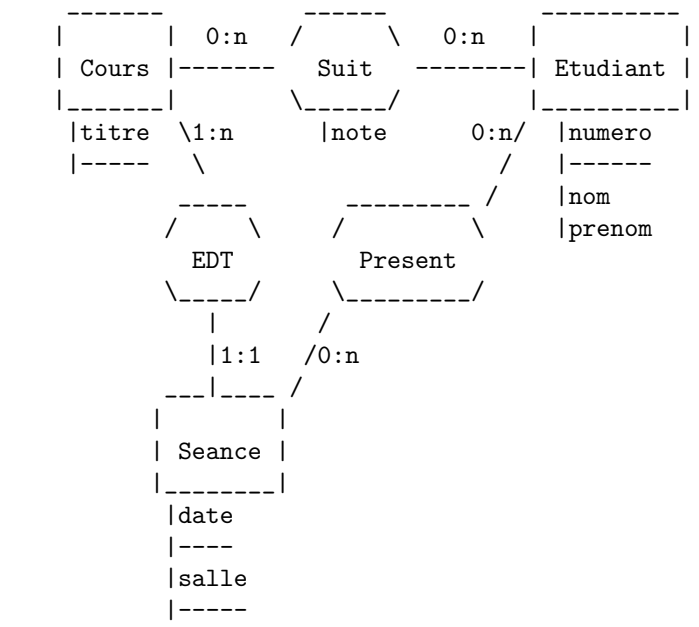

FIGURE 2 – Un exemple de schéma Entité-Association

— Pour commencer, nous parlerons d'**attributs** pour caractériser les objets informatiques élémentaires : nombres, chaînes de caractères, dates [2] – et c'est tout : un attribut ne peut pas être une liste, un ensemble, ou aucune autre structure multivaluée.

— Une **entité** (dans un rectangle) représente un objet du monde réel. Ce n'est donc pas un objet informatique, et *il ne peut être connu que par ses attributs.* Car on lui associe divers attributs, parmi lesquels il devra y avoir un **identifiant**. L'identifiant (souligné) peut être constitué de plusieurs attributs : ainsi la Seance (de cours) est identifiée par sa date et sa salle.

2. Une bonne fois pour toutes : on supposera défini un format date, comprenant jour et heure.

— Une **association** permet de relier un nombre fixe d'entités. Mathématiquement, c'est un ensemble de n-uplets $(a_1, a_2, \cdots, a_n)$ chacun des a_i étant une occurrence d'entité – ou plus précisément de l'identifiant de celle-ci. Quelques remarques :
  — Si une association relie au moins deux entités, elle peut également relier deux fois la même. Ainsi dans une hiérarchie, on pourra avoir l'association Est_chef reliée à l'entité Personne par les deux **rôles** *chef* et *subalterne*.
  — Une association peut également avoir des attributs, par exemple la note d'un Etudiant à un Cours. Elle sera pourtant identifiée par les entités qu'elle lie. Ainsi un appel téléphonique ne peut être une association sans quoi ... une personne ne peut jamais en rappeler une autre !

  Pour les **connectivités** attachées aux rôles dans une association, prenons le cas de la relation EDT (pour Emploi Du Temps). Une occurrence de cette relation est une paire, comprend un et un seul Cours, une et une seule Seance : ce n'est donc pas ce que signifient ces connectivités. En fait, la connectivité entre Cours et EDT signale le nombre de ces paires EDT où figure un Cours – soit en français le nombre de séances associées à un cours : au moins une, au plus... disons "n" pour ne pas parler de l'infini. Par contre, une séance correspond à un et un seul « EDT » donc un et un seul Cours.

Conclusion

Le modèle entité-association est simple et graphique, ce qui le rend très pratique à la fois pour expliquer une BD à un utilisateur lambda, et pour concevoir une base de données. Exemple de la méthode Merise qui peut se résumer en un schéma. L'étude initiale des données [en entité-association] (1) et des traitements [je vous épargne le modèle de traitements de Merise, très peu rigoureux et probablement inutilisable] (2) se fait indépendamment ; les traitements sont raffinés jusqu'à obtenir des procédures élémentaires, les phases ; chacune de ces phases permet alors de déduire un modèle externe ; et ces MXs sont enfin confrontés au MCD brut, permettant d'obtenir un MCD validé (3).

J'insiste sur la simplicité du modèle. Certains (par exemple Merise II, alias Euromerise) veulent y ajouter des constructeurs orientés-objet, où permettre qu'une entité n'ait pas d'identifiant. Cela complique les schémas, et les rend moins aisément compréhensibles, sans permettre d'exprimer quoi que ce soit de plus que le modèle original : je ne recommande pas.

Enfin, le modèle est très proche de l'implémentation : chaque entité va se transformer en table de hachage (c'est-à-dire en *relations* de la suite du cours), et on y intégera les associations reliées en 0 : 1 ou 1 : 1 ; enfin les associations qui n'ont pas de tel rôle donneront une relation séparée. Du point de vue implémentation, le seul défaut du modèle est le choix des identifiants (*clefs primaires*) que les contraintes E-A imposent : les entités doivent y être identifiés par des attributs propres, et les associations par les entités qu'elles relient. Cela peut

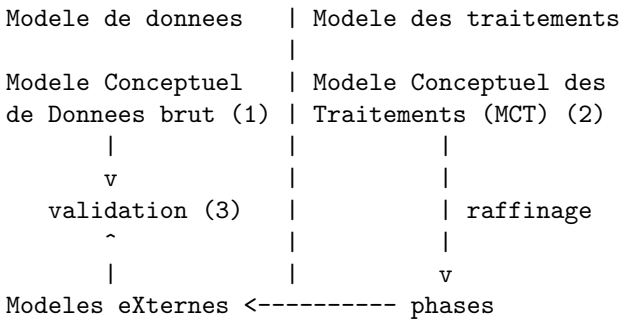

FIGURE 3 – La méthode Merise en un schéma

conduire à des choix d'identifiants assez malheureux, ou à la création d'inutiles identifiants artificiels.