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.

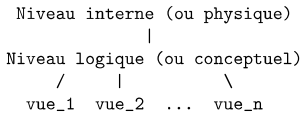

FIGURE 1 – Les niveaux ANSI (1975)

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.

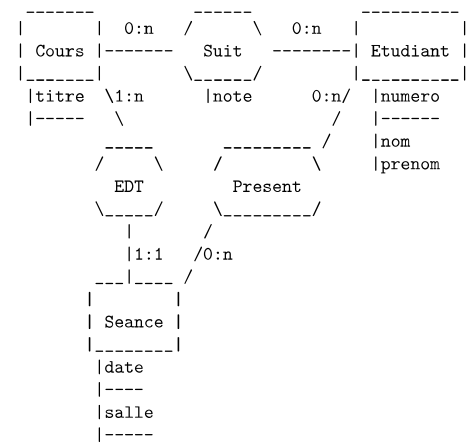

FIGURE 2 – Un exemple de schéma Entité-Association

— 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

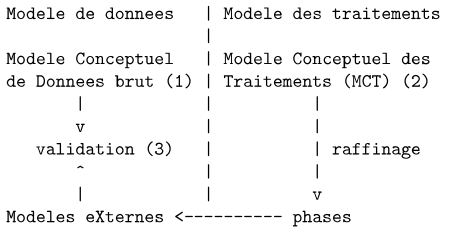

FIGURE 3 – La méthode Merise en un schéma

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

Livraison 2 – modèle relationnel

Pascal Ostermann – pasccal@orange.fr

5 avril 2021

Un schéma relationnel est constitué uniquement de **relations** de la forme

$$R(A_1 : D_1, \cdots, A_n : Dn)$$

où R est un nom de relation (n-aire – on dit aussi que R est d'*arité* n), les A_i n noms d'**attributs** et les D_i n domaines : des types élémentaires de données, nombres, chaîne de caractères, date. Une occurrence de R sera un ensemble de n-uplets $(a_1, \cdots, a_n)$ où $\forall i\ a_i \in D_i$. Remarquez que la définition des attributs correspond à celle vue pour l'entité-association : des valeurs atomiques, et JAMAIS DE LISTES NI D'ENSEMBLES.

Algèbre relationnelle

Il n'est sans doute pas très rigoureux de présenter l'algèbre relationnelle à partir du **calcul relationnel**. Mais à la manière de Monsieur Jourdain, vous connaissez déjà le *calcul* : c'est utiliser la logique formelle [1] dans le cadre du modèle relationnel. Et puis, après avoir beaucoup souffert à dessiner trois schémas avec de mauvais outils, j'ai l'intention de me faire plaisir, et d'écrire des formules mathématiques en LaTeX.

Opérateurs unaires

Projection

Si t est un n-uplet de la relation R et X un sous-ensemble des attributs de cette relation, on notera t.X la restriction de t aux attributs de X. Cette notation pemet de définir élégamment la projection de R sur X :

$$\prod_X(R) =_{def} \{t.X \mid R(t)\}$$

1. ou plus précisément, le calcul des prédicats du premier ordre : les prédicats y sont les relations, et on peut y utiliser les connecteurs $\vee$, $\wedge$, $\neg$, ainsi que les quantificateurs $\forall$ et $\exists$.

Sélection

Définissons d'abord les **conditions de sélection**.

— Si A et B sont des attributs, Cte une constante, et θ un comparateur de la liste $\{=, \neq, <, >, \leq, \geq\}$ alors $A\theta B$ et $A\theta Cte$ sont des conditions de sélection (atomiques).

— Si φ et ψ sont des conditions de sélection, alors $\neg\varphi$, $\varphi \vee \psi$, $\varphi \wedge \psi$ sont des conditions de sélection.

— Rien d'autre n'est une condition de sélection.

La sélection de R suivant la condition C est alors

$$\sigma_C(R) =_{def} \{t \,|\, R(t) \wedge \varphi\}$$

Opérateurs « ensemblistes » $\cup$, $\cap$ et $-$

Vous connaissez déjà ces opérateurs de la théorie des ensembles. Puisque les seuls ensembles qui importent ici sont les relations, nous ne les appliquons qu'à des relations **union-compatibles** :

— de même arité, et

— où les attributs correspondants sont de même domaine.

Dès lors,

$$R \cup S =_{def} \{t \,|\, R(t) \vee S(t)\}$$
$$R \cap S =_{def} \{t \,|\, R(t) \wedge S(t)\}$$
$$R - S =_{def} \{t \,|\, R(t) \wedge \neg S(t)\}$$

Produit cartésien et jointure

Autre opérateur ensembliste, le produit cartésien :

$$R \times S =_{def} \{s, t \,|\, R(s) \wedge S(t)\}$$

où « s,t » est un abus de langage pour le n+m-uplet contenant les valeurs du n-uplet s, suivies de celles du m-uplet t. Mais le produit cartésien est plus souvent utilisé sous forme de **jointure** $\bowtie$[2] qui suppose la donnée d'une condition de jointure :

$$R \bowtie_C S =_{def} \sigma_C(R \times S)$$

Optimisation d'une requête

La jointure est l'opérateur le plus coûteux de l'algèbre relationnelle. L'algorithme en est évident : parcourir l'une des relations, et chercher chaque fois tous les éléments qui correspondent dans la seconde, soit une complexité $n * m$, produit de la taille des deux relations. Mais on peut parfois la rendre plus efficace… par exemple quand il s'agit d'**équijointure** (la condition de jointure est un $\wedge$ d'égalités) ou de **jointure naturelle** (équijointure sur les attributs de même nom, notée sans condition sous le papillon). Dans ce cas, on peut définir un **index** sur l'une des deux relations (celle de taille m, disons), permettant d'accéder plus rapidement à la valeur, en $\log m$ (les arbres binaires de recherche que vous avez rencontrés l'année dernière), voire moins : la jointure se passe alors en $n * \log m$. Notez que la création d'index n'est pas une panacée. Quand on modifie des données sur lesquelles porte un index, il faut également modifier l'index : la création d'index améliore certaines requêtes, mais dégrade certaines mises-à-jour.

Je répète que la jointure est l'opérateur le plus coûteux de l'algèbre relationnelle. La suite du cours – ce que j'appellerai plus loin la **normalisation** – va donc consister à définir des schémas qui permettent de minimiser le nombre de jointures… mais évidemment sans trop dégrader les mises-à-jour.

2. À l'oral, je dis parfois « papillon » ; en LaTeX, j'ai trouvé le symbole sous le nom « bowtie » c'est-à dire nœud-pap.

Extensions de l'algèbre – SQL

À propos de mises-à-jour… L'algèbre relationnelle ne permet d'exprimer que les requêtes (et encore, seulement certaines de celles-ci, car il y manque celles concernant les fonctions d'agrégation : compter, faire des sommes, des moyennes, des variances, etc…). Il y manque la possibilité de modifier des données (**mises-à-jour**) et de définir les relations. Voici donc quel devrait être le plan de n'importe quel langage implémenté. (Le plus fréquent de nos jours est SQL, mais il y en a d'autres.)

```
Langage de Definition des Donnees
Langage de Manipulation des Donnees
  Mises-a-jour
    Inserer
    Supprimer
    Modifier
  Requetes
    Algebre relationnelle
    Fonctions d'agregation
Divers
  Index
  Clefs et autres contraintes d'integrite
  Autorisations d'acces
```

FIGURE 1 – Plan d'un quelconque langage BD

Notez l'intérêt de définir un langage BD en termes déclaratifs, et de le considérer comme une extension de l'algèbre relationnelle : d'une part, on n'y écrit pas directement un programme, mais le résultat que l'on veut obtenir ; et d'autre part les jointures y sont facilement repérables, ce qui permet au SGBD

d'**optimiser** une jointure R $\bowtie$ S – a priori en o(taille(R)*taille(S)) – en appliquant les conditions de sélection sur R et S avant de faire la jointure, et en utilisant les index à bon escient. Si vous utilisez donc une BD dans le cadre d'un intergiciel (par exemple via JDBC – Java DataBase Connectivity), il est clairement plus efficace de faire les jointures dans le langage BD plutôt que de les programmer en Java.

Si j'étais en cours magistral, je consacrerais le reste de la séance à détailler la syntaxe de SQL. L'exercice me semble de peu d'intérêt ici ; et je vous renvoie à vos cours de prépa, et à mon Petit Manuel ci-joint. Sachez pourtant que SQL n'est pas au programme de l'examen.

Exemples

Etudiant(<u>num_et</u>, nom, prenom)
Suit(<u>titre_cours, num_et</u>, note)
Seance(<u>date_seance, salle_seance</u>, titre_cours)
Present(<u>num_et, date_seance, salle_seance</u>)

FIGURE 2 – Même schéma en relationnel

Pour conclure, terminons par quelques requêtes écrites en algèbre relationnelle. Je pars du même schéma que celui défini en entité-association. (Les attributs soulignés y représentent la clef primaire – voir le cours suivant.)

1. Liste des inscrits en BD, avec leur note :

$$\prod_{num\_et,nom,note} (Etudiant \bowtie \sigma_{titre\_cours='BD'} Suit)$$

2. Liste des inscrits à au moins deux cours :

$$\prod_{num\_et} (Suit \bowtie_{num\_et=num\_et' \wedge titre\_cours \neq titre\_cours'} Suit')$$

3. Liste de ceux qui ne sont jamais venus en BD :

$$\prod_{num\_et} Etudiant - \prod_{num\_et} (Present \bowtie \sigma_{titre\_cours='BD'} Seance)$$

Remarques

Dans la première requête (1), j'ai voulu donner un ensemble significatif des attributs d'un Etudiant : une valeur intuitive (le nom) mais surtout un identifiant pour distinguer les éventuels homonymes. J'y ai également appliqué la sélection avant la jointure, de sorte que cet opérateur soit ici le moins coûteux que possible. Enfin la chaîne de caractères y est mise entre quotes (accents aigus, si vous préférez) comme elle le sera en SQL.

Dans les suivantes, je me suis contenté d'un identifiant de l'étudiant : si vous voulez y rajouter d'autres informations, comme les nom et prénom, il suffit de joindre la requête globale avec la relation Etudiant. (2) montre ce qui se produit lorsqu'une même relation apparait plusieurs fois dans une requête : nécessité d'un renommage – ici exprimé par l'usage de la prime – pour distinguer les deux occurrences de la relation. (3) exprime un « pour tout », qui ne peut s'exprimer en algèbre que par la négation (soit une différence ensembliste) de « il existe » (deux projections).

Livraison 3 – normalisation I

Pascal Ostermann – pasccal@orange.fr

12 avril 2021

Redondances

En modélisant une base de données, le principal écueil à éviter est la redondance. Supposons en effet qu'une même donnée ait été représentée plusieurs fois... ou pour nous fixer les idées, que dans le schéma d'école qui court depuis la première livraison, on dispose comme attributs de la note d'un étudiant à un cours, mais aussi de moyennes – calculables à partir des notes. Lorsqu'on modifie une note, il faut également modifier les moyennes. C'est au mieux relativement inefficace, puisqu'il s'agit de modifier plusieurs valeurs au lieu d'une seule. Mais un utilisateur ignorant des subtilités du schéma peut même omettre de modifier les moyennes, et rendre la base de données incohérente. Ce n'est pas trop grave ici : il suffit d'y calculer les nouvelles moyennes. Mais il est des redondances qui ne se gèrent pas aussi aisément.

Réécrivons par exemple le schéma total de cette même école sous la forme d'une unique relation :

$$Ecole(num\_et, nom\_et, prenom\_et, titre\_cours, note, date\_seance, salle\_seance)$$

FIGURE 1 – Encore le même schéma, mal normalisé

Quoique techniquement possible, une telle relation « globale » n'est généralement pas acceptable, et on en voit la raison. Par exemple la note d'un étudiant à un cours est reproduite autant de fois qu'il en a suivi de séances... et s'il y a incohérence entre ces notes, impossible de déterminer laquelle est la bonne. On dit formellement que cette relation n'est pas en deuxième forme normale ; mais il me faut d'abord quelques définitions.

Dépendances fonctionnelles

Soit X et Y deux ensembles d'attributs d'une relation R. La **dépendance fonctionnelle** (DF) X → Y est vraie dans R si et seulement si

$$\forall s, t \in R (s.X = t.X \supset s.Y = t.Y)$$

Propriétés

— si Y ⊂ X, alors X → Y (DF **triviale**)
— si X → Y et Y → Z, alors X → Z
— X → YZ si et seulement si X → Y et X → Z

Clefs et superclefs

Un ensemble d'attributs X est dit **superclef** de la relation R ssi X → R (R abus de langage pour l'ensemble des attributs de R)

X est **clef** de la relation R ssi c'est une superclef minimale pour l'inclusion

Clef primaire, foreign key

Il est évident que toute relation a au moins une clef. Chaque fois que l'on définira une relation – qui deviendra informatiquement une table de hachage –, on en choisira une pour servir de clef d'accès, la **clef primaire**. À partir de maintenant, la clef primaire sera systématiquement soulignée dans nos relations. Et j'exige que vous fassiez de même en TD et à l'examen.

Lorsqu'on fera référence au contenu d'une relation R dans une autre relation S, ce sera via la clef primaire K de R. À l'intérieur de S, K est parfois appelée une **foreign key**, expression que je refuse de traduire car trompeuse : une foreign key dans S n'est en rien une clef de S. Notez bien que lorsque qu'on modifie une clef primaire, il faut également la modifier partout où elle apparaît en tant que foreign key, et modifier également tous les n-uplets concernés. Il convient donc de bien choisir la clef primaire afin de ne jamais avoir à faire cette manipulation. [1]

Décomposition sans perte d'information (SPI)

La décomposition d'une relation R en S et T est dite **sans perte d'information** ssi

$$R = S \bowtie T$$

Théorème de décomposition

Soient X et Y deux ensembles *disjoints* d'attributs d'une relation R. Si X → Y est vraie dans R, alors R se décompose SPI en $\prod_{XY}(R)$ et $\prod_{\complement Y}(R)$.

1. En particulier, dès qu'il s'agit d'êtres humains, il convient de les identifier par un numéro de client, d'employé ou d'étudiant AVANT de découvrir qu'il existe des homonymes. Ce numéro ne peut en France être le numéro de sécu, qui ne peut légalement (lois informatique et liberté) être employé que par l'INSEE et la Sécurité Sociale.

Remarque importante

$\complement Y$ est ici une notation pour le complémentaire de Y dans l'ensemble des attributs de R. Et j'insiste sur le fait qu'il s'agit du complémentaire de Y, la seule *partie droite de la DF !* La partie gauche, commune aux deux résultats de la décomposition, est ce qui nous permet de faire la jointure naturelle entre ces deux relations, et donc de démontrer le théorème de décomposition.

Forme normale de Boyce-Codd (FNBC)

La relation R est en forme normale de Boyce-Codd si et seulement si les seules DFs $X \to Y$ qui y sont vraies sont d'une des formes
— $Y \subset X$ (DF triviale) ou
— X est superclef

Propriété

Donné un ensemble de dépendances fonctionnelles, toute relation peut se décomposer SPI en un ensemble de relations en FNBC.

Méthode de normalisation

La propriété précédente est facile à démontrer : il suffit d'appliquer le théorème de décomposition chaque fois qu'on rencontre une DF qui viole la condition de Boyce-Codd. C'est la logique qui sous-tend la **méthode de normalisation** qui se déroule en trois temps :

1. Établir une liste d'attributs permettant de rendre compte de l'univers du discours. Elle sera plus tard considérée comme une relation, dite « globale ».
2. Établir la liste des dépendances fonctionnelles valides sur ces attributs.
3. Décomposer la relation globale à l'aide du théorème de décomposition, afin d'obtenir des relations normalisées.

Un exemple : centre de recherche

Un centre de recherche, divisé en laboratoires, est réparti dans plusieurs bâtiments d'un campus.

Un bâtiment est identifié par son nom, et une salle par le nom de son bâtiment et un numéro. Une salle est affectée à un unique laboratoire, en tant que lieu collectif (salle machines, local café, etc...) ou en tant que bureau d'un ou plusieurs employés. Dans ce dernier cas, il lui est associé un unique numéro de téléphone. Toutes les salles d'un même laboratoire sont dans le même bâtiment, mais un bâtiment peut accueillir plusieurs laboratoires.

À chaque employé est associé un numéro, ses nom et prénom, une adresse E-mail, le laboratoire auquel il est affecté, son numéro de téléphone au travail, et – pour ceux qui l'acceptent – un numéro de téléphone personnel.

On conservera également les différents articles signés par les chercheurs. Un article sera identifié par son titre, sa date de publication, et le nom de la revue où il a été publié – ou de la conférence où il a été présenté. Un chercheur a pu signer (ou co-signer) un nombre quelconque d'articles. À chaque article est associé un unique domaine (qui n'a rien à voir avec un nom de laboratoire...), un résumé, et un ensemble de mots-clefs. Dans un proche avenir, on prévoit de stocker le texte même de l'article, sous forme de fichier LaTeX.

Liste des attributs

nom_bât, num_salle, nom_labo, num_tél, num_emp, nom_emp, prénom_emp, tél_perso, titre_art, date_art, nom_revue, texte_art, mot_clef[2]

Dépendances fonctionnelles

1. nom_bât, num_salle → nom_labo, num_tél
2. nom_labo → nom_bât
3. num_emp → nom_emp, prénom_emp, tél_perso, nom_bât, num_salle
4. titre_art, date_art, nom_revue → texte_art

Décomposition en FNBC

Le seul résultat indiscutable de cette décomposition est d'obtenir les relations
Salle(nom_bât, num_salle, nom_labo, num_tél)
Article(titre_art, date_art, nom_revue, texte_art)
Emp(num, nom, prénom, tél_perso, bât_bur, num_bur)
et un « reste » R(num_emp, titre_art, date_art, nom_revue, mot_clef)
Les relations Emp et Article sont parfaites, mais R devra être discutée au cours suivant – elle fait intervenir une dépendance multivaluée –; et Salle n'est pas en forme normale de Boyce-Codd à cause de la DF 2 : elle peut se décomposer SPI en
Labo(nom, bâtiment)
Attribution(num_salle, nom_labo, num_tél)
La relation Attribution n'est pas satisfaisante : la clef primaire ne peut y être le téléphone (ne serait-ce que parce que toutes les salles n'ont pas le téléphone) et ne peut être que la paire très peu intuitive num_salle, nom_labo. Par ailleurs, la dépendance fonctionnelle qui permettait d'exprimer la « bonne » clef nom_bât num_salle de la relation Salle ne peut plus s'exprimer dans Labo et Attribution :

2. Chaque fois que je pose cet énoncé ou que j'use de la formulation « ensemble de bidules » certains étudiants créent un attribut « ensemble de bidules. » Je répète qu'un « ensemble » ou une « liste » ne peuvent être des attributs.

on dit qu'il y a perte de dépendances. Mieux vaut ici s'abstenir de cette dernière décomposition, en garder Salle, qui est en troisième forme normale. [3]

Troisième forme normale (3FN)

La relation R est en troisième forme normale si et seulement si les seules DFs $X \to A$ (A est un attribut !) qui y sont vraies sont d'une des formes
— $A \in X$ (DF triviale) ou
— X est superclef ou
— A fait partie d'une clef

Propriétés

— Si R est en FNBC, alors elle est en 3FN.
— Toute relation peut se décomposer SPI et sans perte de dépendances en des relations en troisième forme normale.

Première et deuxième formes normales

Puisqu'il y a une troisième forme normale, vous devinez qu'il y en a aussi une première et une deuxième. Ces définitions sont plus ou moins obsolètes.

À l'origine du modèle relationnel, un attribut pouvait prendre un ensemble de valeurs : la première forme normale l'interdit et exige que ces valeurs soient atomiques : on retombe donc sur notre définition moderne du relationnel. On peut cependant trouver de la littérature sur le relationnel *non first normal form*, que je ne traduis car j'aime trop les acronymes anglais, NFNF ou NF2. Ces modèles permettent d'écrire dans une « case » du tableau un ensemble de valeurs, voire toute une relation. Ils sont plus ou moins équivalents aux modèles orientés-objets…

Quand une DF $X \to A$ viole la condition de la troisième forme normale, il y a deux cas : soit X fait partie d'une clef K, soit il ne fait partie d'aucune clef, ce qui donne un des schémas suivants

$$\begin{array}{ccc} X & \to & A \\ \subset & & \subset \\ K & \to & R \end{array} \qquad\qquad K \to X \to A$$

Dans le premier cas, une partie de la clef détermine quelque chose : on dit qu'il y a DF *partielle* ; dans le second, il y a une DF *transitive*. Une relation est dite en deuxième forme normale ssi elle ne contient pas de DF partielles.

3. Une autre raison de garder Salle est le peu de sens de la redondance que l'on supprime en prenant Labo et Attribution. De fait, dans Salle, la seule redondance est celle de la répétition de l'information « tel labo est dans tel bâtiment » de sorte qu'il n'y a problème que lorsque qu'on veut modifier cette info : il faut alors la modifier pour toutes les salles. Mais dans le monde réel, une telle mise-à-jour correspond à un déménagement du labo : il ne va pas garder la même répartition de salles dans son nouveau bâtiment. Ce problème n'existe pas vraiment.

Livraison 4 – normalisation II

Pascal Ostermann – pasccal@orange.fr

26 avril 2020

Au cours précédent, j'ai arrêté la normalisation du laboratoire de recherche sur la relation

R(num_emp, titre_art, date_art, nom_revue, mot_clef)

Les seules DFs vraies dans R sont triviales : par exemple, un article a plusieurs auteurs ; il lui est associé plusieurs mots-clefs, etc… R est donc en FNBC. Elle n'est pourtant pas satisfaisante, par exemple parce qu'un mot-clef associé à un article est répété autant de fois que cet article a de co-auteurs. En fait on désirerait la décomposer en les deux relations suivantes

Co-auteur(num_emp, titre_art, date_art, nom_revue)

Mot_clef(titre_art, date_art, nom_revue, mot_clef)

Mais pour justifier cette dernière décomposition, il me faut introduire un nouveau type de dépendances, avec un théorème de décomposition et une forme normale.

Dépendances multivaluées

Soit X, Y et Z trois ensembles d'attributs d'une relation R, disjoints deux-à-deux. La dépendance multivaluée (DMV) $X \twoheadrightarrow Y \mid Z$ est vraie dans R si et seulement si

$$\begin{aligned} \forall s,t \in R\,(s.X = t.X \supset \\ \exists u \in R\,(u.X = s.X \wedge \\ u.Y = s.Y \wedge \\ u.Z = t.Z)) \end{aligned}$$

Propriétés

— si X et Y sont disjoints, alors $X \twoheadrightarrow Y|\emptyset$ et $X \twoheadrightarrow \emptyset|Y$ (DMVs triviales)
— si $X \twoheadrightarrow Y|Z$ alors $X \twoheadrightarrow Z|Y$
— si X, Y et Z sont disjoints, et si $X \to Y$, alors $X \twoheadrightarrow Y|Z$
— Certains notent $X \twoheadrightarrow Y$ (DMV globale) pour $X \twoheadrightarrow Y|\complement_R XY$
mais je n'aime pas cette notation à cause du R implicite : $X \twoheadrightarrow Y$ peut être vraie dans une relation R, et fausse dans une relation plus grande…

Exemple

Dans la relation R ci-dessus, la DMV suivante est vraie.

$$titre\_art, date\_art, nom\_revue \twoheadrightarrow num\_emp | mot\_clef$$

En effet, on a vu au cours précédent qu'un article était identifié par son titre, sa date et sa revue de publication. Donc si nous avons deux n-uplets concernant le même article, genre (je choisis un exemple plus parlant qu'un article scientifique)

Fantômas, Souvestre, crime
Fantômas, Allain, Fandor

Nous avons nécessairement également le n-uplet

Fantômas, Souvestre, Fandor

Puisque Souvestre est un des co-auteurs de Fantômas, il est également un des inventeurs du journaliste Jérôme Fandor, quand bien même il n'aurait jamais écrit une ligne sur ce personnage. On peut également dire que les mots_clefs associés à un article – qui correspondent grosso modo à son contenu – sont indépendants de ses auteurs : il ne change pas de contenu selon qu'on le considère comme écrit par untel ou par tel autre.

Théorème de décomposition

Soit X, Y, Z une *partition* de l'ensemble des attributs de R,

$$\text{R se décompose SPI en } \prod_{XY}(R) \text{ et } \prod_{XZ}(R)$$
$$\text{si et seulement la DMV } X \twoheadrightarrow Y|Z \text{ est vraie dans R.}$$

J'insiste sur le fait que X, Y, Z doivent former une partition : disjoints deux à deux comme dans toute DMV, et ils recouvrent R.

Quatrième forme normale (4FN)

Une relation R sera dite en quatrième forme normale si et seulement si toute DMV $X \twoheadrightarrow Y|Z$ (où X, Y, Z partition de R) est telle que

— Y ou Z est vide (DMV triviale) ou
— X est superclef de la relation R

Propriétés

— Le théorème de décomposition pour les DFs est un cas particulier de celui pour les DMVs.
— La 4FN est plus forte que la FNBC (elle-même plus forte que la 3FN, etc...) ; et si une relation n'admet pour DMVs que celles qu'on peut déduire des DFs, les deux formes normales sont équivalentes.
— Toute relation peut se décomposer SPI en des relations en 4FN.

Dépendances de jointure et cinquième forme normale

À lire le théorème de décomposition pour les DMVs et son « si et seulement », on pourrait penser que la quatrième forme normale est aussi la forme terminale. Ce n'est pas tout à fait vrai. Soit par exemple la relation suivante, tirée de Ullman [1] :

Buveurs(buveur, bar, bière)

On considère que cette relation signifie qu'un buveur fréquente un bar, que ce bar sert une bière, et que cette bière est aimée par le buveur. En d'autres termes, elle est par construction équivalente à

Fréquente(buveur, bar) ⋈ Sert(bar, bière) ⋈ Aime(buveur, bière)

Il n'y a pas de DMV dans la relation Buveurs, sans quoi elle serait décomposable en seulement deux relations. On dit qu'il y a ici une **dépendance de jointure** (DJ). Je n'en donne pas de formalisme, chacun des ouvrages que j'ai consultés en donnant une version différente, mais on peut aisément étendre les définitions liées aux DMVs par une cinquième forme normale : les DJs y seraient triviales ou déductibles des clefs. Et énoncer un théorème de décomposition. Mais c'est là que le bât blesse, le dit théorème signifiant qu'on peut décomposer ssi il y a DJ, et la DJ étant définie comme la possibilité de décomposer : on se retrouve à dire « on peut décomposer si et seulement si on peut décomposer. »

En fait, on peut éviter de faire apparaître des DJs avec un petit peu d'attention lors de l'écriture de la liste des attributs. Lorsqu'on définit le troisième attribut (disons que c'est la bière), on découvre qu'il a deux significations, et on décide de le dupliquer, en obtenant cette liste d'attributs :

buveur, bar, bière_servie, bière_aimée

après quoi le résultat désiré s'obtient via les seules DMVs

1. buveur ↠ bar, bière_servie | bière_aimée
2. bar ↠ buveur | bière_servie

DMVs à partie gauche vide

Une autre maladresse que l'on peut commettre en établissant la liste des attributs est d'ajouter un ensemble d'attributs X qui n'ont aucun lien avec les autres Y. Cela peut notamment se produire lorsqu'on duplique des attributs comme ci-dessus, mais inutilement. Dans ce cas la relation globale pourra quand même se décomposer via le produit cartésien :

$$R = \prod_X R \times \prod_Y R$$

En d'autres termes, on a ici la DMV $\emptyset \twoheadrightarrow X|Y$.

1. Principles of database systems, à mon avis l'ouvrage de référence sur le sujet.

Conclusion : ce qu'il faut retenir

La méthode de normalisation consiste donc à

— établir une liste d'attributs – Cette liste doit être plate, sans structuration, afin de donner à chaque attribut un nom sans ambiguïté ; ces attributs doivent vraiment en être : des VALEURS ATOMIQUES, dont vous devez pouvoir signifier le domaine ; par ailleurs, il est plus prudent d'éviter les attributs qui n'ont pas de lien avec les précédents (afin d'éviter les DMVs de partie gauche vide), ou ceux qui auraient plusieurs liens avec ceux-là (afin d'éviter les DJs).

— établir la liste des dépendances – Elles devront être les plus fortes et les plus concises qu'il est possible : par exemple pas de XY $\rightarrow$ Z si X $\rightarrow$ Z est vrai ; pas non plus de X $\rightarrow$ Z si X $\rightarrow$ Y et Y $\rightarrow$ Z sont vraies. [2]

— Décomposer la relation globale constituée de la liste des attributs à l'aide des dépendances, pour obtenir un schéma normalisé – Les seules formes normales à retenir devraient être les troisième, Boyce-Codd et quatrième.

Exemple

Pour finir, le traitement complet de l'exemple du centre de recherche

Liste des attributs

nom_bât, num_salle, nom_labo, num_tél, num_emp, nom_emp, prénom_emp, tél_perso, titre_art, date_art, nom_revue, texte_art, mot_clef

Dépendances

1. nom_bât, num_salle $\rightarrow$ nom_labo, num_tél
2. nom_labo $\rightarrow$ nom_bât
3. num_emp $\rightarrow$ nom_emp, prénom_emp, tél_perso, nom_bât, num_salle
4. titre_art, date_art, nom_revue $\rightarrow$ texte_art
5. titre_art, date_art, nom_revue $\twoheadrightarrow$ num_emp | mot_clef

Résultat

Salle(<u>nom_bât, num_salle</u>, nom_labo, num_tél) – en 3FN
Emp(<u>num</u>, nom, prénom, tél_perso, bât_bur, num_bur)
Article(<u>titre_art, date_art, nom_revue</u>, texte_art)
Co-auteur(<u>num_emp, titre_art, date_art, nom_revue</u>)
Mot_clef(<u>titre_art, date_art, nom_revue, mot_clef</u>)
Sauf Salle, toutes les relations sont en 4FN.

2. Dans des cours plus théoriques et peut-être plus rigoureux – en tous cas plus longs –, on parle ici de la *couverture minimale* d'un ensemble de DFs, en on propose des méthodes pour la calculer, mais... aux dernières nouvelles, cela ne marchait pas vraiment pour les DMVs ; et la couverture minimale est généralement celle qui vient le plus naturellement à l'esprit.