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

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

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,

$$R \cup S =_{def} \{t \mid R(t) \vee S(t)\}$$

$$R \cap S =_{def} \{t \mid R(t) \wedge S(t)\}$$

$$R - S =_{def} \{t \mid R(t) \wedge \neg S(t)\}$$

Produit cartésien et jointure

Autre opérateur ensembliste, le produit cartésien :

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

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

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.

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.

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).