Réécriture et évaluation de requêtes flexibles

Ludovic Liétard, Olivier Pivert, Daniel Rocacher

Introduction

Au-delà de la définition d'un langage de requêtes flexibles comme SQLf, se pose la une question de l'étude des mécanismes d'évaluation de ces requêtes. L'évaluation optimale de requêtes exprimées de façon déclarative (cas de SQL) demeure un problème ouvert en raison de sa nature combinatoire, et les systèmes commerciaux se bornent à employer des heuristiques de bon sens qui, en général, conduisent à des solutions acceptables. Cet état de fait est rendu plus critique, dans le cas des requêtes floues, par l'augmentation des volumes manipulés (seuls les éléments de degré nul peuvent être éliminés) et par l'impossibilité d'utiliser des accélérateurs d'accès usuels tels que les index.

Deux stratégies sont cependant envisageables :

- la traduction d'une requête floue en une requête booléenne (dérivation), étant donné un niveau de satisfaction (seuil qualitatif) à atteindre,
- la compilation de la requête floue. Cette deuxième stratégie vise à obtenir de façon automatique une procédure d'évaluation incluant des heuristiques.

La section 1 de ce document est consacrée à la dérivation de requêtes booléennes et montre qu'elle est bien adaptée aux requêtes de type projection-sélection-jointure. La section 2 présente brièvement les principes liés à la compilation de requêtes floues. Une annexe contient les définitions de la théorie des ensembles flous qui sont utiles à la lecture de cette présentation.

Une présentation orale en a été faite lors de la réunion du groupe APMD du 14 mars 2006 à Lyon [7].

1. Dérivation de requêtes booléennes SQL et fonctions externes

Cette solution vise à limiter les développements logiciels par une utilisation maximale (pour autant qu'elle soit possible) des capacités des SGBD relationnels commerciaux (en

particulier leurs mécanismes d'optimisation). Pour ce faire, la requête graduelle initiale est *transformée* en une requête booléenne usuelle incluant des fonctions pour gérer les calculs de degré. Une telle requête booléenne délivre un ensemble flou qui correspond aux éléments du résultat dont le degré dépasse un seuil α fixé (par défaut 0) – ou un *sur-ensemble*, auquel cas il est nécessaire d'*éliminer les éléments indésirables* (opération appelée calibrage par la suite). Cette approche permet de traiter de façon efficace les requêtes SQLf du type sélection-projection-jointure lorsque les calculs de degré ne posent pas de problème.

1.1 Principe

Le principe consiste à voir le résultat recherché (ou plus exactement son support) comme une coupe de niveau (α-coupe) (*cf.* annexes) du résultat de la requête graduelle initiale, calculée au moyen d'une requête ne faisant intervenir que des opérations et expressions booléennes. Le problème est principalement de distribuer l'opération d'α-coupe portant sur une expression de sélection sur les différentes composantes de celle-ci. Il a été montré dans [1, 2] que cette transformation était possible pour la grande majorité des prédicats flous, y compris ceux faisant intervenir des modificateurs et des connecteurs. Selon les connecteurs présents dans la requête floue initiale, la condition booléenne obtenue peut retourner l'α-coupe exacte (*dérivation forte*) ou un sur-ensemble de celle-ci (*dérivation faible*), auquel cas les éléments indésirables doivent être supprimés au moyen d'une condition complémentaire portant sur le degré. L'architecture logicielle correspondant à cette méthode est décrite par la figure 1.

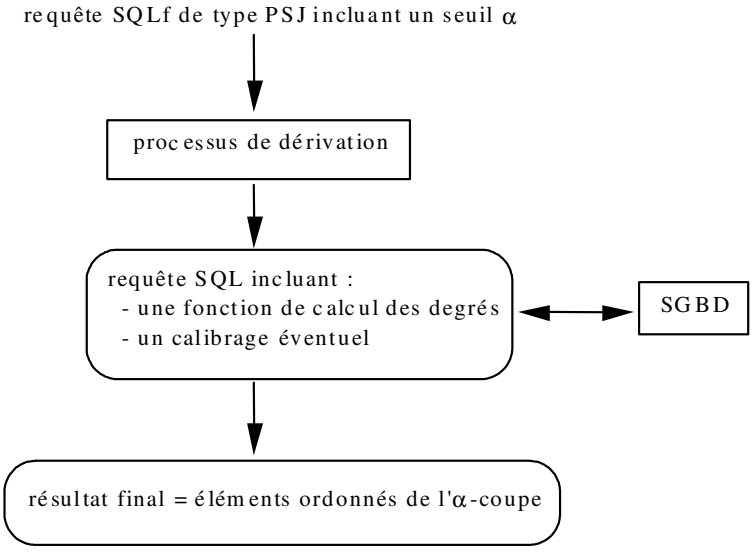

Figure 1. *Architecture logicielle d'évaluation de requête SQLf (dérivation)*

Exemple 1. Soit la requête visant à trouver les numéros des employés qui vérifient la condition : être bien payé et travailler dans un département à budget moyen, à un degré au moins égal à 0.8. L'évaluation de cette requête repose sur le calcul de la 0.8-coupe de la condition :

salaire = "bien payé" **et** budget = "moyen"

appliquée au résultat de l'équi-jointure des relations emp et dept. En supposant que les prédicats flous soient ceux spécifiés dans la figure 2, la condition booléenne dérivée (équivalente) est "salaire ≥ 15000 **et** budget ∈ [3.2, 3.8]", puisque :

(salaire = "bien payé" **et** budget = "moyen")(x) ≥ 0.8 ⇔

min((salaire = "bien payé")(x), (budget = "moyen")(x)) ≥ 0.8 ⇔

(salaire = "bien payé")(x) ≥ 0.8 **et** (budget = "moyen")(x) ≥ 0.8.

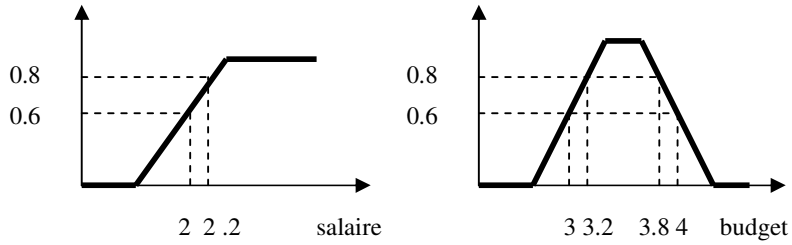

Figure 2. *Les prédicats "salaire = bien payé" et "budget = moyen"*

La requête SQL finale est donc :

select emp#, mu(salaire, budget) **from** emp E, dept D
where E.dep# = D.dep# **and** salaire ≥ 15000 **and** budget **between** 3.2 **and** 3.8

où mu est une fonction qui calcule le degré final de chaque n-uplet du résultat (ici, cette fonction calcule le minimum des degrés relatifs aux conditions bien-payé(salaire) et moyen(budget)).

En revanche, si le connecteur reliant les deux termes flous était une *moyenne arithmétique* (notée am), on aurait :

am(salaire = "bien payé", budget = "moyen")(x) ≥ 0.8 ⇔

((salaire = "bien payé")(x) + (budget = "moyen")(x)) / 2 ≥ 0.8 ⇒

((salaire = "bien payé")(x) + 1) / 2 ≥ 0.8 **et** ((budget = "moyen")(x) + 1) / 2 ≥ 0.8 **et**
max((salaire = "bien payé")(x), (budget = "moyen")(x)) ≥ 0.8

et la condition finale (qui, dans ce cas, n'est pas équivalente à l'α-coupe initiale puisque *des implications et non des équivalences* ont été utilisées) serait :

salaire ≥ 13000 **et** budget ∈ [3, 4] **et** (salaire ≥ 15000 **ou** budget ∈ [3.2, 3.8]).

Il serait alors nécessaire d'ajouter dans la clause "**where**" une condition portant sur les degrés pour écarter les éléments n'appartenant pas à l'α-coupe de la requête initiale.♦

Comme l'illustre l'exemple précédent, il se peut que l'on obtienne un sur-ensemble de l'α-coupe souhaitée, et donc qu'un certain nombre d'éléments indésirables fassent l'objet d'un accès, ce qui augmente bien évidemment le coût global de l'évaluation. Des mesures [3] expérimentales ont été effectuées pour évaluer la proportion de tels n-uplets en fonction des connecteurs flous présents dans la requête initiale. Il s'avère que cette méthode constitue, pour les requêtes simples, une technique d'évaluation efficace.

1.2. Limites de cette approche

Cette approche est bien adaptée au cas des requêtes floues du type projection-sélection-jointure dans la mesure où :

1) il est possible, pour ces requêtes, de dériver une condition booléenne,
2) le calcul du degré final associé à un n-uplet repose uniquement sur les valeurs d'attribut de ce n-uplet.

En ce qui concerne les requêtes imbriquées, cette approche est envisageable si la requête peut se réécrire en une requête floue de type projection-sélection-jointure (cas de l'opérateur *in* par exemple). Elle ne peut être utilisée dans les autres cas (opérateur *not in*, partitionnement par exemple), où la stratégie basée sur une traduction procédurale s'impose donc.

2. Traduction procédurale et utilisation de conditions d'arrêt

Ici encore, l'idée est de mettre en oeuvre une interface au dessus d'un SGBD usuel, mais il s'agit cette fois d'une *interface de plus bas niveau* dans la mesure où l'objectif n'est plus d'obtenir une requête SQL mais un *programme faisant appel à des requêtes SQL* (dont le rôle se limite à accéder aux données), le calcul des degrés étant effectué dans le programme lui-même.

Des schémas de traduction pour les diverses composantes des requêtes SQLf doivent donc être définis, d'une façon similaire à ce qui est fait dans un SGBD usuel (pour les requêtes booléennes). Considérons, à titre d'exemple typique, la requête :

select ... **from** r **where** c1 **and** A **not in**
(**select** B **from** s **where** c2).

Un n-uplet t de r est sélectionné (ou non) selon la satisfaction de la condition c1 et l'absence dans s d'un n-uplet possédant la valeur t.A et satisfaisant la condition c2. Dans ce contexte, un n-uplet t de r est satisfaisant *si et seulement si* :

$$\min(\mu_{c1}(t),\ 1 - \sup_{v \in s} \min(\mu_{c2}(v),\ \mu_{=}(t.A,\ v.B))) \geq \alpha \qquad (1).$$

Un algorithme naif pour évaluer cette requête se compose de deux boucles imbriquées. La première pour accéder aux n-uplets t de la relation r, une deuxième pour accéder aux n-uplets v de la relation s pour calculer $\sup_{v \in s} \min(\mu_{c2}(v), \mu_{=}(t.A, v.B))$ pour un n-uplet t.

Il a été montré dans [4] que, de la formule (1), peuvent être inférées deux conditions d'arrêt par échec permettant (éventuellement) d'*éviter de parcourir exhaustivement* la relation s dans la boucle imbriquée, et donc de réduire la complexité de la procédure. Ces conditions sont :

H1 : $\mu_{c1}(u) < \alpha$ d'une part,

H2 : $(\exists v \mid \mu_{c2}(v) > (1 - \alpha)$ **et** u.A = v.B) d'autre part,

qui toutes deux rendent fausse la formule (1). D'un point de vue pratique, l'algorithme d'évaluation consiste en un programme écrit en un langage tel que C/SQL ou PL/SQL, incluant des requêtes SQL pour accéder aux données. Pour une requête floue imbriquée du type considéré ici, deux requêtes booléennes sont nécessaires (l'une sur la relation r, l'autre sur la relation s), et il est intéressant de noter que la condition H1 peut se traduire par un critère de sélection (obtenu par dérivation) dans la première de ces requêtes. L'algorithme d'évaluation correspondant est donné ci-dessous :

```
declare crs1 cursor for
      select * from r where DRV(c1, ≥, λ);          \\ condition dérivée issue de H1
declare crs2 cursor for
      select * from s where s.B = a;
result := Ø;
for each u in crs1 do
   a := u.A;
   if μc1(u) ≥ λ then                               \\ test utile si dérivation faible
      μ' := 0;
      failure := false;
```

```
      for each v in crs2 and not failure do
         if μc2(v) > 1 − λ then failure := true          \\ heuristique H2
         else μ' := max(μ', μc2(v)) endif
      enddo;
      if not failure then
         μ := min(μc1(u), 1 − μ');                      \\ calcul du degré final μ associé à r
         result := result + {<μ/u>}
      endif
   endif
enddo
```

Exemple 2. Considérons les extensions des relations emp de schéma (emp#, âge, dep#) et dept de schéma (dep#, budget) des tables 1 et 2 et la requête visant à trouver les départements à moyen budget où ne travaille aucun employé jeune, avec un seuil de 0.6. Cette requête floue initiale s'exprime, sous une forme imbriquée :

select 0.6 dep# **from** dept **where** budget = "moyen" **and** dep# **not in**
(**select** dep# **from** emp **where** âge = "jeune").

Soit Cder la condition dérivée obtenue à partir de la condition floue : "budget = moyen" associée au seuil minimal 0.6. Supposons que la fonction d'appartenance associée au prédicat flou "moyen" soit celle de la figure 2. La condition dérivée Cder, obtenue ici grâce à une dérivation forte, s'exprime : "budget ∈ [3, 4]". Le processus d'évaluation est décrit ci-après :

- accès au premier n-uplet r de dept satisfaisant Cder : r = <4, 3.8>; le premier n-uplet s de emp tel que s.B = r.A est lu : s = <82, 34, 4>; la condition c2(s) > (1 − α) est vérifiée (ici, 0.6 > 0.4) et aucun autre accès à emp n'est effectué;
- le n-uplet suivant de dept ne satisfait pas Cder et n'est donc pas retourné; aucun accès à emp n'est nécessaire. ♦

emp

emp#	âge	μ_{jeune}	dep#
82	34	0.6	4
6	39	0.1	4
37	28	1	2
21	30	0.8	4

Table 1 *Extension de la relation emp*

dept

dep#	budget	μ_{moyen}
4	3.8	0.8
2	2.9	0.5

Table 2 *Extension de la relation dept*

D'une façon similaire, des conditions d'arrêt "anticipé" ont été proposées pour l'évaluation de propositions quantifiées dans [5]. Dans ce contexte, des conditions d'arrêt par échec basées sur certaines propriétés des opérateurs utilisés (notamment la monotonie) peuvent être mises en évidence, et il est raisonnable d'envisager d'évaluer de telles requêtes à l'aide d'algorithmes analogues à celui présenté plus haut.

3. Conclusion

Nous avons mis en évidence plusieurs méthodes permettant d'évaluer des requêtes floues. L'approche consistant à développer un SGBD spécifique dont le noyau serait pourvu d'algorithmes d'évaluation adaptés à chaque opérateur de base du langage, ainsi que de mécanismes d'indexation ad hoc, ne peut certes être écartée a priori, mais elle induit un coût de développement important et l'accent a été plutôt mis sur une deuxième voie fondée sur l'interfaçage d'un SGBD classique avec un module de pré-traitement des requêtes graduelles. Deux types d'interfaçage peuvent être envisagés :

- le premier repose sur le principe dit "de dérivation", qui consiste à traduire une requête SQLf en une requête SQL, le calcul des degrés étant effectué au moyen de fonctions incluses dans la requête obtenue. Dans le cas général, la requête dérivée retourne un sur-ensemble des éléments souhaités, et un post-traitement est alors nécessaire pour éliminer les éléments indésirables. Cette technique permet de traiter efficacement des requêtes floues "simples", c'est-à-dire des requêtes monobloc de type projection-sélection-jointure,
- le second fait appel à la compilation de la requête SQLf initiale. Ce que l'on dérive n'est plus une requête SQL mais une procédure comportant des curseurs pour accéder aux données et des heuristiques pour optimiser ces accès. Dans ce cadre, aucun post-traitement n'est nécessaire, les degrés étant calculés par le programme lui-même. Cette approche permet d'évaluer efficacement des requêtes floues complexes, notamment des requêtes imbriquées.

Un prototype d'interrogation flexible d'une base de donnée relationnelle au travers d'une interface web a été mis en œuvre [6]. Ce prototype est fondé sur le principe de la dérivation et permet de traiter des questions du type selection-projection-jointure. L'interface permet de définir un profil utilisateur caractérisé par ses propres définitions de concepts graduels comme cher, loin, grand, … A l'aide de ces concepts l'utilisateur construit les différents composants de sa requête. Celles-ci est ensuite traitée pour en extraire une requête applicable à la base de données. Les résultats retournées sont ensuite affichés avec une évaluation de leur niveau d'adéquation.

Annexes.

Ensemble flou (définition).

Un ensemble ordinaire A se voit associer sa *fonction caractéristique* f_A à valeur dans le doublet {0, 1}. Pour tout élément x du domaine de référence X (appelé aussi univers ou référentiel), f_A (x) spécifie si x *appartient ou non* à A. Cette démarche est étendue aux ensembles flous auxquels est associée une fonction (d'appartenance) notée μ, à valeur dans l'intervalle [0, 1] de $\mathbb{R}$, appelé *intervalle unité*). La valeur $\mu_E(x)$ exprime dans quelle mesure (à quel point) l'élément x de X appartient à l'ensemble flou E :

$$\begin{aligned} \mu_E : X &\rightarrow [0, 1] \\ x &\rightarrow \mu_E(x). \end{aligned}$$

Quand $\mu_E(x)$ est nul, x n'appartient pas du tout à E et quand il vaut 1, x est complètement dans E (x est aussi dit élément typique de E). Plus μ_E (x) est proche de 1 (resp. 0), plus (resp. moins) x appartient à E. Un ensemble usuel peut être vu comme un cas particulier d'ensemble flou dont la fonction d'appartenance ne prend que les valeurs 0 et 1.

Intersection, union et complémentation (définitions).

L'intersection (resp. l'union) de deux ensembles flous et un ensemble flou dont la fonction d'appartenance est définie par une norme (resp. co-norme) triangulaire. Le minimum (resp. maximum) est le plus souvent utilisé comme norme (resp. co-norme) et l'on obtient : $\forall x \in X : \mu_{E \cap F}(x) = \min(\mu_E(x), \mu_F(x))$ et $\mu_{E \cup F}(x) = \max(\mu_E(x), \mu_F(x))$. Le complément d'un ensemble flou est défini par la complémentation à 1 ($\mu_{\bar{F}}(x)) = 1 - \mu_F(x)$), ce qui permet la conservation des lois de De Morgan. Les opérateurs de logique étendus (et, ou, not) sont également définis de la même manière.

Support et noyau d'un ensemble flou (définitions).

Le *support* d'un ensemble flou E, noté supp(E), correspond à l'ensemble usuel des éléments appartenant quelque peu à E, soit :

$$\text{supp}(E) = \{x \mid x \in X \textbf{ et } \mu_E(x) > 0\}.$$

Le *noyau* d'un ensemble flou E, noté noy(E), se définit comme l'ensemble usuel des éléments appartenant complètement à E, donc :

$$\text{noy}(E) = \{x \mid x \in X \textbf{ et } \mu_E(x) = 1\}.$$

Les notions de support et de noyau sont confondues dans le cas d'un ensemble usuel A puisque si x appartient quelque peu à A, il est (complètement) dans A.

Référence

[1] P. Bosc, O. Pivert, On the evaluation of simple fuzzy relational queries: principles and measures, in: *Fuzzy Logic: State of the Art*, (R. Lowen & M. Roubens, eds.), Kluwer Academic Publishers, pp. 355-364, 1993.

[2] P. Bosc, H. Prade, An introduction to fuzzy set and possibility theory-based approaches to the treatment of uncertainty and imprecision in data base management systems, in : *Uncertainty Management in Information Systems – From Needs to Solutions*, (A. Motro & P. Smets, eds.), Kluwer Academic Publishers, pp. 285-324, 1997.

[3] P. Bosc, O. Pivert, On the efficiency of the alpha-cut distribution method to evaluate simple fuzzy relational queries, in: *Advances in Fuzzy Systems – Applications and Theory Vol. 4: Fuzzy Logic and Soft Computing* (B. Bouchon-Meunier, R.R. Yager & L.A. Zadeh, eds.), World Scientific, 1995, pp. 251-260, 1995.

[4] P. Bosc, O. Pivert, Requêtes flexibles et division relationnelle, *Ingénierie des Systèmes d'Information*, vol. 5, pp. 661-69, 1997.

[5] P. Bosc, L. Liétard, On the comparison of the Sugeno and the Choquet fuzzy integrals for the evaluation of quantified statements, *3rd European Congress on Fuzzy and Intelligent Technologies (EUFIT'95)*, Aachen (Germany), pp. 709-716, 1995.

[6] S. Bartel, *Interrogation floue de bases de données : extension de iSQLf.* Rapport de projet de 3ème année ENSSAT, avril 2006.

[7] L. Liétard, *Réécriture de requêtes*, Transparents présentés à la réunion APMD de mars 2006, http://www.prism.uvsq.fr/bscw/bscw.cgi/d5516/R%c3%a9%c3%a9criture.ppt