Requêtes floues et SGBD relationnels : vers un couplage renforcé

Fuzzy queries and RDBMSs : towards a tighter coupling

G. Smits [1] O. Pivert [1] T. Girault [2]
[1] IRISA-ENSSAT, Université de Rennes 1, [2] Ingénieur freelance

Technopole Anticipa, BP 80518, 22305 Lannion Cedex
{smits, pivert}@irisa.fr, toma.girault@gmail.com

Résumé :

Dans cet article, nous présentons une stratégie pour la mise en œuvre d'un système d'interrogation floue autour d'un SGBD classique. Le système considéré repose sur le langage SQLf qui permet d'exprimer une grande variété de requêtes floues. Les expérimentations menées montrent que la stratégie d'implémentation choisie apporte des gains de performances par rapport aux architectures existantes fondées sur un couplage plus lâche entre une couche dédiée à l'évaluation des requêtes floues et un SGBD, ce type de solution nécessitant une étape de post-traitement pour calculer la relation floue produite par une requête. Nous décrivons également une interface conviviale visant à aider les utilisateurs non-experts à exprimer leurs requêtes floues de façon intuitive.

Mots-clés :

Bases de données, requêtes floues, optimisation de requête, SQLf, PostgreSQL, interface.

Abstract:

In this paper, we present an implementation strategy for a fuzzy querying system embedded in a regular DBMS. This system relies on the language SQLf that makes it possible to express a great variety of fuzzy queries. Experiments show that this implementation strategy induces performance gains with respect to existing strategies based on a loose (or milder) coupling between a fuzzy querying layer and a DBMS, that necessitate an external postprocessing so as to compute the result in the form of a fuzzy relation. We also describe a user-friendly interface aimed at helping nonexpert users express their fuzzy queries in an intuitive manner.

Keywords:

Database fuzzy querying, query optimization, SQLf, PostgreSQL, interface.

1 Introduction

Durant la dernière décennie, on a pu constater un regain d'intérêt pour l'expression de préférences dans les requêtes de bases de données. En réalité, les premiers travaux de recherche sur ce sujet remontent aux années 80, voir par exemple [12]. Les motivations pour les requêtes à préférences sont multiples [10]. Tout d'abord, il est apparu souhaitable d'offrir des langages de requête plus expressifs, pouvant traduire plus fidèlement le besoin d'information d'un utilisateur. En deuxième lieu, l'introduction de préférences dans les requêtes fournit une base à l'ordonnancement des réponses, ce qui est particulièrement appréciable dans le cas d'ensemble de réponses volumineux. Enfin, une requête classique peut produire un résultat vide, alors qu'une version relaxée (et donc moins restrictive) de la requête pourra peut-être être satisfaite par quelques éléments.

Dans cet article, nous considérons l'approche de requêtes à préférences fondée sur la théorie des ensembles flous [4, 6], qui bénéficie de la grande expressivité de cette dernière quant à la modélisation de différents types de critères graduels et à leur combinaison. Le cadre langagier considéré est SQLf [4, 14], une extension floue de SQL initialement proposée dans les années 90 et complétée depuis par divers opérateurs additionnels. Ce langage incorpore de nombreux constructeurs flous et constitue un outil puissant pour exprimer des requêtes à préférences dans un cadre de bases de données. Cependant, deux questions n'ont pas trouvé jusqu'à présent de réponses définitives, qui concernent respectivement : i) la mise en œuvre efficace d'un système d'interrogation fondé sur SQLf (ce qui soulève la question de l'optimisation de requêtes floues), ii) la façon dont on peut aider un utilisateur non-expert à spécifier ses requêtes floues (qui sont intrinsèquement plus complexes que des requêtes classiques, ne

serait-ce que parce que des fonctions d'appartenance entrent en jeu). Dans cet article, nous montrons qu'une architecture bien choisie permet d'atteindre de bonnes performances quant à l'évaluation de requêtes, et nous définissons également une technique conviviale reposant sur une interface graphique pour la spécification des requêtes floues. Ces deux résultats devraient contribuer, du moins l'espérons-nous, à convaincre la communauté des bases de données de la pertinence et de la faisabilité de l'approche floue pour la modélisation de requêtes à préférences.

La suite de l'article est structurée comme suit. La section 2 fournit quelques rappels sur l'interrogation floue de bases de données et présente les caractéristiques principales du langage SQLf. La section 3 discute différents types d'architecture de systèmes d'interrogation floue et expose le principe qui a été retenu dans le prototype que nous avons développé. La section 4 porte sur l'expressivité du fragment de SQLf qui a été implémenté dans ce prototype. En section 5, des résultats expérimentaux sont reportés, qui montrent que la stratégie de mise en œuvre choisie conduit à de bonnes performances relativement à d'autres types d'architecture. La section 6 présente une interface conviviale qui peut être utilisée pour spécifier une requête floue pas à pas. Enfin, la section 7 conclut l'article et esquisse quelques perspectives de futurs travaux.

2 Rappels sur SQLf

Les opérateurs de l'algèbre relationnelle peuvent être étendus de façon assez directe aux relations floues en considérant d'une part ces dernières comme des ensembles flous, et en introduisant des prédicats graduels dans les opérations appropriées d'autre part. En guise d'illustration, nous donnons ci-après la définition de la sélection floue, où r désigne une relation (floue ou classique), ψ une condition floue, et $\top$ une norme triangulaire (généralement $\min$) :

$$\mu_{\sigma_\psi(r)}(t) = \top(\mu_r(t),\ \mu_\psi(t)).$$

Une extension de SQL s'appuyant sur cette algèbre relationnelle étendue est présentée dans [4, 14]. Nous nous limitons ici à la description du bloc de base. Par rapport à SQL, les différences majeures concernent :

- le calibrage du résultat, qui peut être réalisé à l'aide d'un seuil quantitatif (nombre souhaité de réponses, noté k), d'un seuil qualitatif (degré minimal de satisfaction exigé, noté α), ou les deux,
- la nature des conditions autorisées, qui peuvent être floues.

En conséquence, le bloc de base devient :

select [**distinct**] [$k \mid \alpha \mid k, \alpha$] *attributs*
from *relations* **where** *cond-floue*

où *cond-floue* peut faire intervenir à la fois des prédicats flous et des critères booléens.

Par ailleurs, SQLf préserve (et étend) les constructeurs spécifiques à SQL, par exemple les opérateurs d'imbrication, le partitionnement de relation, etc.

3 Architectures de systèmes d'interrogation floue

L'évaluation de requêtes floues [7, 11, 9, 3] soulève différentes difficultés, parmi lesquelles les principales sont énumérées ci-dessous :

- les SGBD commerciaux n'offrent pas d'outils permettant de définir de façon simple des fonctions d'appartenance, des connecteurs flous, etc.
- il n'est pas possible d'utiliser directement les index existants lors de l'évaluation d'une condition de sélection (ou de jointure) floue ;
- une étape additionnelle vouée au calcul des degrés de satisfaction et au calibrage du résultat (top-k ou seuillage qualitatif) est nécessaire, ce qui induit un coût supplémentaire lors de l'évaluation.

En conséquence, la mise en œuvre d'un système d'interrogation floue peut être abordée selon trois types d'architecture [16] :

– *couplage lâche* : les nouvelles fonctionnalités sont intégrées au travers d'une couche logicielle au-dessus du SGBD. Le principal avantage de ce type d'architecture réside dans sa portabilité, puisque n'importe quel SGBD peut être utilisé comme moteur de requête sous-jacent. Son principal point faible tient à ses relativement médiocres performances, qui rendent difficiles le passage à l'échelle.
– *couplage moyen* : les nouvelles fonctionnalités peuvent être intégrées par l'intermédiaire de procédures stockées, à l'aide d'un langage procédural approprié aux bases de données tel que PL/SQL (dans le cas d'Oracle). Une alternative consiste à recourir à des fonctions externes. Avec ce type de solution, les données sont directement gérées par le noyau du SGBD, ce qui conduit à de meilleures performances.
– *couplage fort* : les nouvelles fonctionnalités sont incorporées dans le noyau même du SGBD. Cette solution, qui est évidemment la plus efficace en termes d'évaluation de requête, implique de réécrire entièrement le moteur d'évaluation, y compris l'analyseur syntaxique et l'optimiseur du SGBD, ce qui est une tâche très lourde.

Le type de mise en œuvre que nous présentons dans cet article est à la jonction entre le couplage moyen et le couplage fort dans la mesure où i) les fonctions d'appartenance correspondant aux prédicats flous spécifiés par l'utilisateur sont définies comme des procédures stockées et ii) les extensions graduelles des opérateurs (norme triangulaire pour la conjonction, conorme triangulaire pour la disjonction, quantificateurs flous) sont implémentées en langage C et intégrées dans le moteur de requêtes du SGBD relationnel PostgreSQL.

En l'absence de SGBD commerciaux capables d'interpréter des requêtes floues, certaines approches de la littérature proposent de passer par une étape de dérivation *derivation step* [5] de façon à générer une requête booléenne utilisée pour préfiltrer la relation (ou le produit cartésien des relations) concernée. L'idée est de restreindre le calcul des degrés de satisfaction aux seuls n-uplets qui satisfont à un niveau suffisant la condition de sélection (ou de jointure). Dans ce type d'approche, l'évaluation de requêtes floues comporte trois étapes :

1. dérivation d'une requête booléenne à partir de la condition floue apparaissant dans la requête utilisateur,
2. évaluation de cette requête booléenne et production de la relation résultante,
3. calcul des degrés de satisfaction attachés aux n-uplets de cette relation (et élimination éventuelle des n-uplets non suffisamment satisfaisants), ce qui produit la relation floue constituant le résultat final.

En termes de performances, l'intérêt d'utiliser une architecture de type "couplage moyen" réside dans le fait que la relation floue résultante est calculée durant la phase de sélection des n-uplets (aucun programme externe n'a besoin d'être appelé pour effectuer l'étape 3).

De manière à faciliter la maintenance et la distribution du code, les fonctionnalités nécessaires à l'évaluation de requêtes floues ont été mises en œuvre sous la forme d'une extension (PGXS) du SGBD open source PostgreSQL. Les fonctionnalités décrites dans la suite en section 4 ont été implémentées par des fonctions écrites en C ou des procédures en PL/PYTHON ou PL/PGSQL. La mise en œuvre tire parti du fait que PostgreSQL autorise l'appel à des modules externes au moment de l'exécution.

4 Expressivité du langage

Dans un premier temps, nous avons implémenté les principales fonctionnalités du langage SQLf, en modifiant légèrement la syntaxe définie dans [4, 14] de façon à éviter tout conflit avec la syntaxe SQL et le moteur de requête de PostgreSQL.

Prédicats flous : Des fonctions d'appartenance trapézoïdales peuvent être définies sur des attributs numériques en utilisant la procédure

suivante, où le premier paramètre définit l'étiquette linguistique attachée au prédicat et les autres correspondent aux bornes de la fonction :

select newTrapezoidalFuzzySet('autour_de_2007', 2004, 2006, 2008, 2010).

La définition de prédicats flous sur des attributs catégoriels peut se faire à l'aide de l'instruction suivante, où le premier paramètre définit l'étiquette linguistique attachée au prédicat, le deuxième énumère les valeurs un tant soit peut satisfaisantes, et le dernier spécifie les degrés de satisfaction associés :

select newDiscreteFuzzySet('French', ['Peugeot', 'Citroen', 'Dacia'], [1.0, 1.0, 0.6]).

Conditions floues. Des prédicats flous peuvent être introduits dans la clause de sélection (*where*), où $\sim=$ correspond à l'opérateur noté *is* dans [5] :

select * **from** *cars* **where** *year* $\sim=$ 'recent'.

Modificateurs. Des modificateurs flous peuvent être utilisés pour altérer le sens de certains prédicats. Les modificateurs disponibles par défaut (et définis *a priori* dans le système) sont *very* (modificateur renforçant) et *rather* (modificateur relaxant), définis comme suit : $\mu_{mod\,P}(x) = (\mu_P(x))^n$ où $n = 2$ (resp. 0.5) si *mod* est *very* (resp. *rather*)

select * **from** *cars* **where** year $\sim=$ 'very recent'.

Conjonctions et disjonctions. Comme dans le cas classique, une condition de sélection peut être définie comme une conjonction ou une disjonction de prédicats flous ou booléens. Les connecteurs SQL **and** et **or** ont été étendus respectivement par les opérateurs && et || . La paire t-norme/t-conorme sousjacente à ces opérateurs peut être choisie au moyen de l'instruction :

select set_norm('*norm*') ;

où *norm* peut prendre les valeurs : *Zadeh*, *probabiliste*, *Łukasiewicz* ou *Weber*. Une fonction qui convertit les booléens en nombres réels est utilisée pour combiner des prédicats flous et booléens. Un exemple de requête floue conjonctive dans ce formalisme est :

select * **from** *cars*
where *year* $\sim=$ 'very recent' && km $\sim=$ 'low'.

Seuils : Un seuil qualitatif (α) ou quantitatif (k) peut être défini pour contrôler le niveau de satisfaction ou la cardinalité du résultat. Leur spécification s'effectue comme suit :

select set_alpha(0.4) ; **select** set_k(20).

Quantificateurs flous. Outre la conjonction et la disjonction, un quantificateur flou peut être utilisé pour combiner des prédicats booléens ou flous. Trois interprétations sont disponibles dans le prototype : celle de Zadeh (fondée sur un Sigma-count de l'ensemble flou impliqué), CTA (*Competitive Type Aggregation*, qui correspond à l'utilisation d'une intégrale de Sugeno) et OWA (*Ordered Weighted Aggregation*, fondée sur une intégrale de Choquet) [2]. Le choix s'effectue au moyen de l'instruction :

select set_quantifier('*quantifier*') ;

où *quantifier* prend l'une des valeurs *zadeh*, *cta* ou *owa*. Un exemple de requête est :

select * **from** *cars* **where** most(*year* $\sim=$ 'very recent', km $\sim=$ 'low, *brand* = 'Peugeot', *consumption* $\sim=$ 'low', price $\sim=$ 'reasonable').

Opérateurs graduels. Dès lors qu'une fonction de distance a été définie sur un domaine d'attribut, des opérateurs graduels peuvent être employés pour effectuer des comparaisons avec des scalaires (opérateur $\sim$) ou tester une inclusion graduelle (opérateur $in\ \sim$). Une variante de ces opérateurs a aussi été définie de façon à exprimer des comparaisons/inclusions graduelles vis-à-vis du résultat d'une sous-requête (voir [14] pour plus de détails). Un exemple dans le cas non-imbriqué est :

select * **from** *cars* **where** *year* $\sim$ 2008 && *brand* $in\ \sim$ ('Peugeot', 'Renault').

5 Évaluation de requête

Afin d'estimer le gain en termes de performances induit par la stratégie d'implémentation

que nous proposons, par rapport à des systèmes fondés sur un couplage lâche [13] avec et sans étape de dérivation, nous avons utilisé une relation appelée *randomtable* de cardinalité 100,000. Les n-uplets de *randomtable* ont été générés de façon pseudo-aléatoire sur trois attributs, à savoir *id*, *attnum*, et *atttext*. Le premier est la clé primaire, le deuxième un attribut numérique dont le domaine est l'intervalle [0, 200], et le troisième un attribut catégoriel dont le domaine est l'ensemble des vingt chaînes de caractères correspondant aux vingt premiers nombres entiers ("un", "deux", "trois", etc). Nous avons ensuite défini deux prédicats au moyen des instructions :

select newTrapezoidalFuzzySet('low', 0, 0, 10, 15) ;
select newDiscreteFuzzySet('around_10', ['huit', 'neuf', 'dix', 'onze', 'douze'], [0.6, 0.8, 1.0, 0.8, 0.6]) ;

L'objectif de cette expérimentation préliminaire était de comparer, à l'aide du prototype PostgreSQLf, le temps d'exécution associé à trois stratégies d'implémentation différentes : couplage lâche sans dérivation (LCND), couplage lâche avec dérivation (LCD), et couplage moyen avec dérivation (MCD). Le temps mesuré intègre la construction de la relation floue résultante. Pour réaliser ces mesures, trois requêtes ont été soumises aux trois moteurs d'évaluation correspondants. Cet échantillon, quoique limité, est représentatif des différents types de requêtes floues pouvant être traitées par PostgreSQLf. La requête 1 est composée d'un seul prédicat de sélection sur un attribut numérique, la requête 2 d'une conjonction de deux prédicats flous, et la requête 3 utilise un quantificateur flou pour agréger deux prédicats :

– Query 1 :
  select * **from** *randomtable* **where** *attnum* ∼= 'low' ;
  qui se dérive en
  select * **from** *randomtable* **where** *attnum* **between** 0 **and** 15 ;
– Query 2 :
  select * **from** *randomtable*
  where *attnum* ∼= 'low' && *atttext* ∼= 'around_10' ;
  qui se dérive en
  select * **from** *randomtable* **where** *attnum* **between** 0 **and** 15 **and** *atttext* **in** ('huit', 'neuf', 'dix', 'onze', 'douze')) ;
– Query 3 :
  select * **from** *randomtable*
  where most(*attnum* ∼= 'low', *atttext* ∼= 'around_10') ;
  qui se dérive en
  select * **from** *randomtable* **where** *attnum* **between** 0 **and** 15 **or** *atttext* **in** ('huit', 'neuf', 'dix', 'onze', 'douze')).

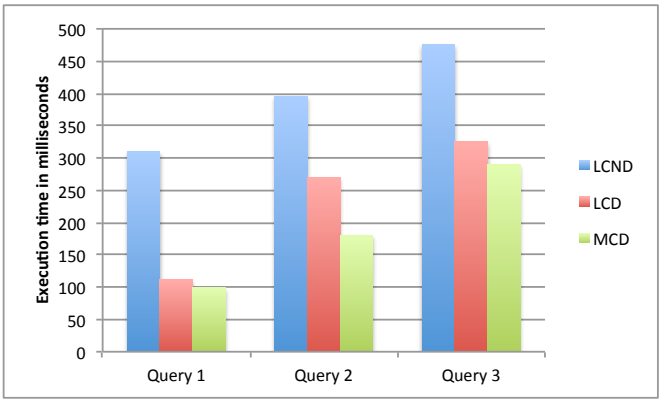

Figure 1 – Temps de calcul / stratégie de mise en œuvre

Les résultats sont reportés dans la figure 1. Il s'avère que la solution "couplage moyen" a les deux principaux avantages suivants :

– elle permet une meilleure gestion des relations floues puisque ces dernières sont directement retournées par le SGBD ;
– elle apporte un gain significatif en termes de performances.

Nous travaillons actuellement à optimiser l'utilisation des index pour évaluer plus efficacement les fonctions utilisateur associées aux prédicats flous.

6 Interface conviviale

6.1 Introduction

À l'évidence, un langage tel que SQLf n'est pas très facile à utiliser pour des utilisateurs novices, non familiers avec les concepts de la théorie des ensembles flous. C'est pourquoi il est nécessaire de définir une interface intuitive pour les aider à formuler leurs requêtes.

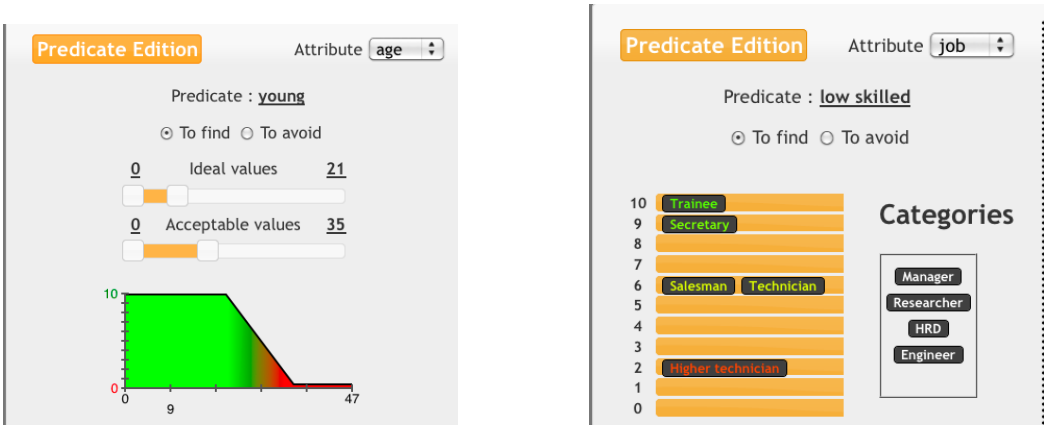

Figure 2 – Définition de prédicats flous numérique (gauche) et catégoriel (droite)

Le prototype que nous avons développé agit comme une interface avec le SGBD et génère des requêtes SQLf directement exécutables par ce dernier par l'intermédiaire d'appels à des fonctions stockées. L'interface est connectée à la base de données de façon à stocker les profils utilisateurs, composés d'informations d'authentification et rassemblant les prédicats et requêtes flous précédemment définis. De cette façon, il est possible de construire un vocabulaire personnel composé d'ensembles flous et de requêtes soumises antérieurement. Cette base de données contient également les informations de connection aux bases accessibles et la liste des attributs sur lesquels des prédicats flous peuvent être définis.

6.2 Définition intuitive de prédicats flous

Comme la sémantique d'un prédicat flou repose sur sa fonction d'appartenance, tout sysytème d'interrogation flou doit fournir aux utilisateurs un moyen commode de définir de telles fonctions. En pratique, la fonction d'appartenance associée à un ensemble flou F est souvent choisie de forme trapézoïdale. Dans ce cas, F est exprimée par un quadruplet $(A,\ B,\ C,\ D)$ où $noyau(F) = [B,\ C]$ et $support(F) = [A,\ D]$. Si la fonction est une "épaule droite" (resp. gauche), elle peut être codée par $(A,\ B,\ k_2,\ k_2)$ (resp. $(k_1,\ k_1,\ C,\ D)$) où $[k_1,\ k_2]$ représente le domaine de l'attribut concerné.

Diverses méthodes ont été proposées précédemment pour définir des termes flous dans un cadre d'interrogation flexible de bases de données. Dans [11], les auteurs décrivent un prototype construit au-dessus du SGBD commercial Microsoft Access, où la définition des prédicats, comparateurs et quantificateurs flous s'opère via une interface graphique. Cependant, le type d'interface considéré est à notre avis encore trop exigeant dans la mesure où il implique une certaine connaissance de ce qu'est une fonction d'appartenance (l'utilisateur doit en effet spécifier directement les bornes des fonctions qu'il souhaite utiliser).

Dans [9, 8], Goncalves et Tineo introduisent des commandes permettant de définir des prédicats (**create fuzzy predicate**) et des comparateurs flous (**create comparator**) dans SQLf, mais leur usage apparaît encore plus difficile pour un utilisateur non-expert.

Pour des requêtes floues simples, une autre solution, proposée dans [15], est d'inclure les définitions des prédicats flous dans la requête elle-même. Cependant, l'inconvénient de cette méthode est qu'elle peut mener rapidement à des expressions de requêtes assez encombrantes.

Dans [1], les auteurs suggèrent d'utiliser la notion d'*ordre flou* pour spécifier des prédicats flous définis sur des domaines numériques. Ils considèrent trois opérateurs, à savoir *vaut au*

moins, *vaut au plus*, et *appartient à*, rendus tolérants au moyen d'une clause *tolerate up to*. Bien qu'intéressante, cette méthode a une expressivité limitée puisqu'elle ne permet de spécifier que des termes flous atomiques sur des attributs numériques. De plus, elle apparaît tout de même d'une convivialité limitée.

L'interface que nous avons développée propose une méthode intuitive et simple pour définir des prédicats flous. Le panneau d'édition illustré en figure 2 permet à l'utilisateur de spécifier des prédicats sur des attributs numériques (gauche) ou catégoriels (droite). Dans le premier cas, la fonction d'appartenance peut être définie en utilisant deux curseurs pour fixer les intervalles de valeurs acceptables et idéales, ou en déplaçant les bornes de la fonction trapézoïdale dans la représentation graphique. Pour les attributs catégoriels, l'utilisateur peut placer les valeurs de l'attribut concerné dans les boîtes correspondant aux degrés de satisfaction d'une échelle prédéfinie. L'interface permet aussi de raffiner des prédicats flous définis par d'autres utilisateurs sur l'attribut considéré (coin inférieur droit de la figure 4), ainsi que de sauvegarder les prédicats flous (dans le profil de l'utilisateur courant) de façon à pouvoir les réutiliser ou les modifier par la suite.

6.3 Construction de requête

Les prédicats flous peuvent ensuite être déplacés et déposés dans la panneau d'édition de requête floue. La figure 4 montre comment la représentation graphique du prédicat *'wellPaid'*, qui est nié pour obtenir la condition *salary is not 'wellPaid'*, est déplacée dans la boîte d'édition qui contient déjà le prédicat *job is 'lowSkilled'*. De cette façon, l'utilisateur peut créer de nouveaux groupes de prédicats ou augmenter un groupe existant. Dès qu'un groupe contient plus d'un prédicat, l'utilisateur doit choisir le connecteur qu'il souhaite utiliser pour effectuer l'agrégation des degrés. Les requêtes sont stockées dans le profil de l'utilisateur courant sous la forme d'un document XML.

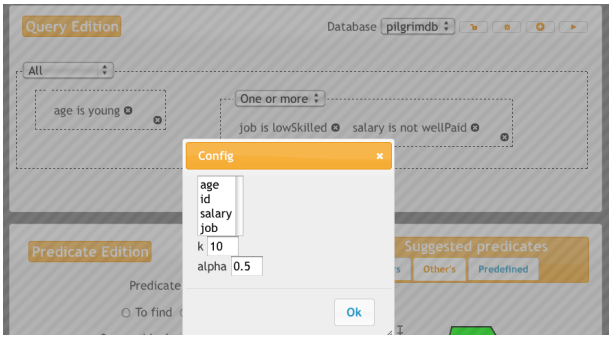

Figure 3 – Paramètres additionnels pour une requête floue

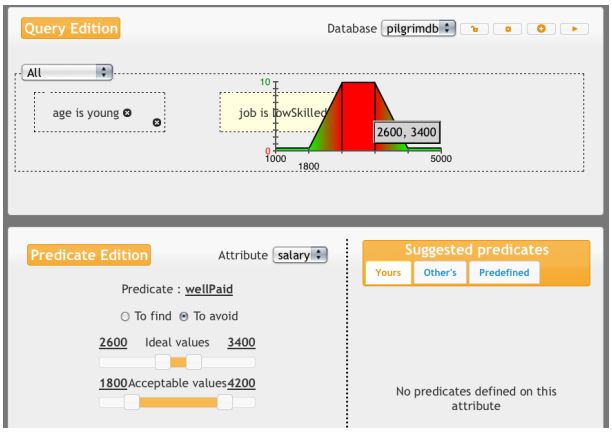

Figure 4 – Vue globale de l'interface

Le coin supérieur droit de l'interface propose un ensemble de boutons permettant de réinitialiser la requête, de se déconnecter, d'exécuter la requête, ou de fixer certains paramètres additionnels (figure 3) tels que : i) la liste des attributs à introduire dans la clause de projection ; ii) un seuil qualitatif α ou quantitatif k.

Quand l'exécution de la requête est déclenchée, les prédicats flous sont transformés en fonctions pl/pgsql, la requête est traduite en SQLf, et le code est alors soumis au SGBD. L'exécution d'une requête floue Q retourne une relation floue où chaque n-uplet t se voit associer un degré de satisfaction $\mu_Q(t)$ (noté simplement *mu* dans le panneau de résultat). Comme les n-uplets sont présentés par ordre décroissant de leur degré de satisfaction, l'utilisateur peut aisément identifier les réponses qui satisfont au mieux sa requête et peut également ajuster le paramètre k pour réduire la taille du résultat s'il

le souhaite.

7 Conclusion

Dans cet article, nous avons présenté une stratégie de mise en œuvre d'un fragment du langage SQLf permettant l'interrogation floue d'une base de données. Cette stratégie, fondée sur un couplage moyen avec un SGBD relationnel, permet d'atteindre de bonnes performances en termes de temps d'exécution. De nombreux domaines d'application pour lesquels l'interrogation flexible apporte une valeur ajoutée significative en termes d'expressivité peuvent bénéficier d'un tel système, sans qu'il soit besoin de faire appel à un module externe pour compiler une requête booléenne dérivée. Nous poursuivons actuellement le développement du prototype présenté dans l'article, de façon à intégrer davantage de fonctionnalités de SQLf et à améliorer encore les performances en exploitant des structures de données internes (index) plus efficaces.

Nous avons également conçu et développé une interface utilisateur pour la spécification de requêtes floues, avec l'objectif de montrer que les éléments principaux d'une requête floue peuvent être définis aisément par un utilisateur novice. Ceci nous a conduit à déterminer les limites entre ce qui peut être compris et manipulé par un utilisateur non-expert (prédicats flous, propositions quantifiées floues, etc) et ce qui doit être réservé au mode expert d'édition de requête (questions impliquant un partitionnement flou de relation, par exemple). Nous travaillons actuellement à de nouvelles fonctionnalités pour le mode non-expert, telles que la possibilité offerte à un utilisateur de définir ses propres quantificateurs flous, d'assigner des poids aux prédicats d'une condition complexe, ou d'exprimer des requêtes floues imbriquées.

Références

[1] U. Bodenhofer and J. Küng. Fuzzy orderings in flexible query answering systems. *Soft Comput.*, 8(7) :512–522, 2004.

[2] P. Bosc, L. Liétard, and O. Pivert. Quantified statements and database fuzzy querying. In P. Bosc and J. Kacprzyk, editors, *Fuzziness in Database Management Systems*, pages 275–308. Physica Verlag, 1995.

[3] P. Bosc and O. Pivert. On the evaluation of simple fuzzy relational queries : principles and measures. In R. Lowen and M. Roubens, editors, *Fuzzy logic – State of the Art*, pages 355–364. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1993.

[4] P. Bosc and O. Pivert. SQLf : a relational database language for fuzzy querying. *IEEE Trans. on Fuzzy Systems*, 3 :1–17, 1995.

[5] P. Bosc and O. Pivert. SQLf query functionality on top of a regular relational database management system. In O. Pons, M. Vila, and J. Kacprzyk, editors, *Knowledge Management in Fuzzy Databases*, pages 171–190. Physica-Verlag, Heidelberg, Germany, 2000.

[6] D. Dubois and H. Prade. Using fuzzy sets in flexible querying : Why and how ? In *Proc. of the 1996 Workshop on Flexible Query-Answering Systems*, pages pp. 89–103, 1996.

[7] J. Galindo, J. M. Medina, O. Pons, and J. C. Cubero. A server for fuzzy SQL queries. In *Proc. of FQAS'98*, pages 164–174, 1998.

[8] M. Goncalves and L. Tineo. Sqlf flexible querying language extension by means of the norm sql2. In *Proc. of FUZZ-IEEE'01*, pages 473–476, 2001.

[9] M. Goncalves and L. Tineo. SQLf3 : An extension of SQLf with SQL features. In *Proc. of FUZZ-IEEE'01*, pages 477–480, 2001.

[10] A. Hadjali, S. Kaci, and H. Prade. Database preference queries - a possibilistic logic approach with symbolic priorities. *Ann. Math. Artif. Intell.*, 63(3-4) :357–383, 2011.

[11] J. Kacprzyk and S. Zadrożny. FQUERY for ACCESS : Fuzzy querying for a windows-based DBMS. In P. Bosc and J. Kacprzyk, editors, *Fuzziness in Database Management Systems*, pages 415–433. Physica Verlag, 1995.

[12] M. Lacroix and P. Lavency. Preferences : Putting more knowledge into queries. In *Proc. of the 13rd VLDB Conference*, pages 217–225, 1987.

[13] Y. López and L. Tineo. About the performance of sqlf evaluation mechanisms. *CLEI ELECTRONIC JOURNAL*, 9(2), 2006.

[14] O. Pivert and P. Bosc. *Fuzzy Preference Queries to Relational Databases*. Imperial College Press, London, UK, 2012.

[15] O. Pivert and G. Smits. On fuzzy preference queries explicitly handling satisfaction levels. In S. Greco, B. Bouchon-Meunier, G. Coletti, M. Fedrizzi, B. Matarazzo, and R. R. Yager, editors, *IPMU (1)*, volume 297 of *Communications in Computer and Information Science*, pages 341–350. Springer, 2012.

[16] A. Urrutia, L. Tineo, and C. Gonzalez. FSQL and SQLf : Towards a standard in fuzzy databases. In J. Galindo, editor, *In Handbook of Research on Fuzzy Information Processing in Databases*, pages 270–298. Information Science Reference, Hershey, PA, USA, 2008.