PostgreSQLf : un Système d'Interrogation Floue

Grégory Smits[1], Olivier Pivert[1], and Thomas Girault[2]

[1]Irisa - Univ. Rennes 1, Lannion France
{pivert, smits}@enssat.fr
[2] Ingénieur R&D indépendant - Rennes, France
toma.girault@gmail.com

Résumé Nous présentons dans cet article l'implémentation d'un système d'interrogation floue intégré à un gestionnaire de base de données relationnel. Ce système s'appuie sur le langage SQLf pour exécuter des requêtes comportant des prédicats flous reliés par des conjonctions, disjonctions, quantificateurs ou par d'autres opérateurs graduels.

Keywords. Langage d'interrogation de BD, requête floue, SQLf, PostgreSQL.

1 Introduction

Les ensembles flous constituent un cadre théorique et méthodologique intéressant pour l'interrogation flexible de Base de Données Relationnelles (BDR). En effet, associé à une fonction d'appartenance et une étiquette linguistique, un ensemble flou formalise une propriété graduelle pouvant être ensuite intégrée en tant que préférence dans une requête adressée à une BDR. En ce sens, les requêtes floues étendent les requêtes booléennes à travers la prise en compte de tolérance et de gradualité dans la définition des intentions des l'utilisateurs. La logique floue apporte également un riche ensemble de connecteurs permettant de combiner ces préférences de manière compensatoire ou non [1]. Un langage d'interrogation nommé SQLf [2] a ainsi été proposé pour exploiter l'expressivité des requêtes floues.

Cependant, les Systèmes de Gestion de BDR (SGBDR) commerciaux reposent essentiellement sur un langage *booléen* d'interrogation, à savoir SQL. L'interprétation d'une requête floue sur un SGBDR nécessite donc soit une étape de dérivation [3] afin d'être transformée en requête *booléenne* soit l'utilisation d'une extension dédiée aux requêtes floues. Nous proposons dans cet article la première implémentation d'un système d'interprétation de requêtes floues intégré au cœur d'un SGBDR commercial, en l'occurrence PostgreSQL [1].

2 SQLf et l'Évaluation de Requêtes Floues

2.1 SQLf

Une requête SQLf [3] intègre des prédicats graduels la clause WHEREde la requête. La satisfaction d'un tuple vis-à-vis d'une requête floue étant désormais graduelle (degré de satisfaction dans $[0, 1]$) et non plus binaire, la requête peut être complétée par un seuil qualitatif α exprimant le degré minimum de satisfaction acceptable pour qu'un tuple fasse partie de la relation floue résultat et/ou un seuil quantitatif K indiquant le nombre de tuples attendus. Le squelette d'une requête SQLf est le suivant :

SELECT [distinct] [K α | α | K] attributs FROM relations WHERE condition-floue ;

1. Version actuelle PostgreSQL 9.1.2 `http://www.postgresql.com`

où *condition-floue* peut comporter à la fois des prédicats booléens et flous. Cette expression est interprétée comme la sélection floue du produit cartésien des relations de la clause *from*, la projection des attributs spécifiés dans la clause *select* (par défaut les tuples dupliqués sont conservés à moins que le mot clé *distinct* soit spécifié et l'on ne conserve alors que le degré maximal de satisfaction pour chaque résultat). Les opérateurs relationnels et ensemblistes utilisés dans les requêtes booléennes ont été étendus [2] pour prendre en compte des prédicats flous et retourner des relations floues.

Un exemple typique d'une requête floue adressée à une base d'annonces de voitures d'occasion utilisant les prédicats illustrés par la figure 2.1 serait :

SELECT 20 0.6 * FROM voitures_occasion WHERE annee IS 'recente' AND kilometrage IS 'faible';

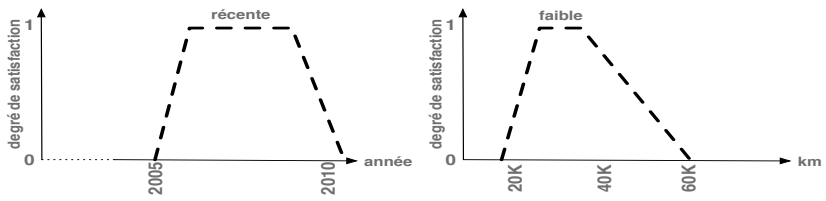

Figure 1. Définition des ensembles flous '*récente*' (*année*) et '*faible*' (*kilométrage*)

2.2 Interprétation de Requêtes Floues

Le développement d'un système d'interrogation floue au dessus d'un SGBDR peut s'effectuer selon trois stratégies [4]. La première, qualifiée de faiblement intégrée, repose sur une couche logicielle développée au dessus du SGBDR et chargée de récupérer tous les tuples de la BD et de calculer pour chacun d'eux leur degré de satisfaction vis-à-vis des prédicats floues spécifiés par l'utilisateur. Les tuples dont le degré de satisfaction est supérieur ou égal à α sont conservés pour former la relation floue résultat. Cette version naïve du processus d'interprétation d'une requête floue peut être améliorée à l'aide d'un pré-traitement consistant à effectuer une dérivation de la requête floue [3]. Cette étape de dérivation consiste à traduire la requête floue en requête *booléenne* permettant de retourner uniquement les tuples satisfaisant les contraintes floues à un degré supérieur ou égal à α. La requête floue introduite en Section 2.1 serait ainsi dérivée en :

SELECT * FROM voitures_occasion
WHERE annee BETWEEN 2006 AND 2009 AND kilometrage BETWEEN 22600 AND 46800;

La seconde stratégie d'implémentation, qualifiée de semi-intégrée, est fondée sur l'utilisation de procédures stockées ou externes qui définissent à la fois les fonctions d'appartenance des prédicats flous et les opérateurs nécessaires au calcul de la relation floue résultat. La troisième stratégie, qualifiée de pleinement intégrée, s'appuie sur une implémentation des opérateurs relationnels flous au cœur même du SGBD. Ce dernier type d'implémentation, qui est évidemment le plus efficace, nécessite un effort d'implémentation très conséquent et conduit à un système délicat à maintenir sans nuire à l'efficacité du système d'interprétation des requêtes *booléennes*. Nous proposons une implémentation semi-intégrée au SGBDR PostgreSQL avec un développement des prédicats flous en tant que procédures stockées et des connecteurs flous en tant que fonctions C.

Par rapport à une implémentation faiblement intégrée, notre approche apporte non seulement un gain de performances, mais permet également d'obtenir directement une relation floue comme résultat de l'exécution de la requête par le SGBDR.

3 PostgreSQLf

Afin de faciliter la maintenance et la distribution du code, les fonctionnalités nécessaires à l'évaluation de requêtes floues ont été implémentées sous forme d'une extension [2] (PGXS) du SGBDR libre PostgreSQL.

3.1 Fonctionnalités

La première version du module PostgreSQLf permet de manipuler les principaux éléments qui composent une requête SQLf à savoir :

Prédicats flous : de forme trapézoïdale pour les attributs numériques avec spécification de l'étiquette linguistique associée et des bornes du trapèze,

```
SELECT newTrapezoidalFuzzySet('recente', 2005, 2006, 2008, 2010) ;
```

ou de forme discrète pour les attributs catégoriels avec spécification de l'étiquette linguistique, des catégories concernées et de leurs degrés de satisfaction respectifs.

```
SELECT newDiscreteFuzzySet('francaise', ['peugeot', 'citroen', 'dacia'], [1.0, 1.0, 0.6]) ;
```

Conditions floues : dans la clause WHERE , où $\sim=$ correspond à l'opérateur IS préconisé dans [3].

```
SELECT * FROM voitures_occasion WHERE annee ~= 'recente' ;
```

Modificateurs : pour modifier le degré d'appartenance retourné par un prédicat flou afin le rendre plus restrictif ou tolérant. Les modificateurs disponibles ('très', 'peu') sont définis comme suit : $\mu_{tresP}(x) = \mu_P(x)^2$ et $\mu_{peuP}(x) = \sqrt{\mu_P(x)}$.

```
SELECT * FROM voitures_occasion WHERE annee ~= 'tres recente' ;
```

Conjonctions et disjonctions : Les opérateurs SQL AND et OR ont été respectivement étendus à l'aide des opérateurs && et || , où par défaut l'opérateur minimum est utilisé pour la conjonction et le maximum pour la disjonction. Différentes implémentations (*zadeh*, *probabiliste*, *lukasiewicz* et *weber*) de ces opérations ont été réalisées et peuvent être sélectionnées à l'aide de l'instruction :

```
SELECT set_norme('norme') ;
```

```
SELECT * FROM voitures_occasion WHERE annee ~= 'très recente' && km ~= 'faible' ;
```

Seuils : pour calibrer la qualité des résultats (α) ou la quantité (K) :

```
SELECT set_alpha(0.4) ; SELECT set_K(20) ;
```

Quantificateurs : intervenant dans des conditions floues du type $Q\,X\,are\,A$, où A est un prédicat flou, X un ensemble de tuples et Q (le quantificateur) exprime une opération d'agrégation telle que "la plupart". Différentes méthodes d'interprétation des quantificateurs (Zadeh, CTA et OWA [5]) ont été implémentées et peuvent être sélectionnées à l'aide de l'instruction : SELECT set_quantifier('owa') ; . Le quantificateur *la_plupart* est prédéfini.

2. Version actuelle disponible à l'adresse `https://github.com/postgresqlf/PostgreSQL_f`

SELECT * FROM voitures_occasion WHERE la_plupart(annee ∼= ‘tres recente’, km ∼= ‘faible’, marque = ’peugeot’, consommation ∼= ’faible’, prix ∼= ’raisonnable’) ;

Opérateurs graduels : pour effectuer des inclusions (opérateur *in* ∼) ou comparaisons (opérateur ∼) graduelles en s’appuyant sur des mesures de distance définies sur le domaine de définition des attributs concernés. Une variante de ces opérateurs a également été définie pour gérer l’inclusion graduelle vis-à-vis des résultats d’une requête imbriquée. La requête suivante permet de récupérer les voitures de marque ‘peugeot’, ‘renault’ ou d’une marque proche sémantiquement (selon la mesure de distance définie sur cet attribut).

SELECT * from voiture_occasion WHERE année ∼ 2008 && marque *in* ∼ (‘peugeot’, ‘renault’) ;

3.2 Performances

Pour quantifier le gain de performance d’un système d’interrogation floue semi-intégré par rapport à des systèmes non intégrés avec et sans stratégie de dérivation, nous avons utilisé une relation nommée **randomtable** composée de 100 000 tuples générés pseudo-aléatoirement sur trois attributs (id, attnum, atttext), le premier étant la clé primaire, le second de valeur numérique de domaine de définition $[0, 200]$, le troisième textuel ayant comme domaine de définition un ensemble de 20 chaînes de caractères distinctes (écriture alphabétique des vingt premiers nombres). Nous avons ensuite défini deux prédicats flous à l’aide des instructions suivantes :

SELECT newTrapezoidalFuzzySet(’faible’, 0, 0, 10, 15) ;

SELECT newDiscreteFuzzySet(’dizaine’, [’huit’, ’neuf’, ’dix’,’onze’,’douze’], [0.6, 0.8, 1.0, 0.8, 0.6]) ;

Nous avons ensuite comparé le temps d’exécution des 3 requêtes suivantes selon la stratégie d’implémentation Non Intégrée Sans Dérivation (NISD), Non Intégrée Avec Dérivation (NIAD) et Semi Intégrée (SI) à l’aide du module PostgreSQLf. Les temps d’exécution NISD, NIAD et SI intègrent la construction de la relation floue résultat.

- Requête 1 : SELECT * FROM randomtable where attnum ∼= ’faible’ ; dérivée en SELECT * FROM randomtable where attnum BETWEEN 0 AND 15 ;,
- Requête 2 : SELECT * FROM randomtable WHERE attnum ∼= ’faible’ && atttext ∼= ’dizaine’ ; dérivée en SELECT * FROM randomtable WHERE attnum BETWEEN 0 AND 15 AND atttext IN (’huit’, ’neuf’, ’dix’,’onze’,’douze’)) ;,
- Requête 3 : SELECT * FROM randomtable WHERE la_plupart(attnum ∼= ’faible’, atttext ∼= ’dizaine’) ; dérivée en SELECT * FROM randomtable WHERE attnum BETWEEN 0 AND 15 or atttext IN(’huit’, ’neuf’, ’dix’,’onze’,’douze’)) ;.

Les résultats obtenus sont exposés dans la Figure 2. Une implémentation semi-intégrée permet non seulement de manipuler plus facilement des relations floues depuis les applications puisqu’elles sont directement retournées par le SGBDR, comme l’illustre la Figure 3, mais offre également un gain non négligeable en termes de temps d’exécution. Nous travaillons actuellement à une meilleure utilisation des index au sein des fonctions utilisateurs définissant les prédicats flous afin de réduire encore les temps de traitement.

4 Conclusion et Perspectives

Dans cet article, nous avons présenté une implémentation des fonctionnalités de base du langage SQLf dédié à l’interrogation floue de BDR. Cette implémentation intégrée au SGBDR PostgreSQL nous a permis d’obtenir de bonnes performances

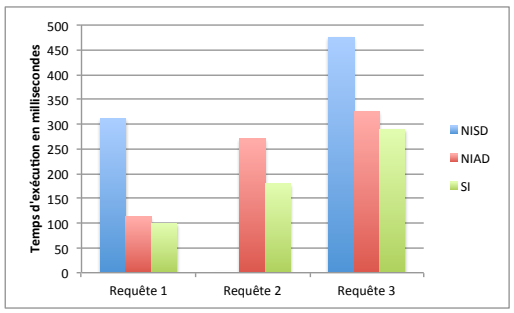

Figure 2. Temps d'exécution en fonction du niveau d'implémentation

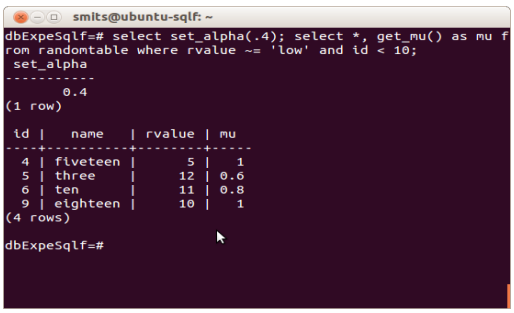

Figure 3. Utilisation en ligne de commande du module PostgreSQLf

d'évaluation de ces requêtes floues. De nombreux contextes applicatifs pour lesquels l'interrogation flexible de BDR apporte une amélioration significative de l'expressivité du langage de requête mis à la disposition des utilisateurs peuvent désormais bénéficier des performances de PostgreSQLf et ne plus avoir recours à une phase de dérivation externe des requêtes. Nous poursuivons actuellement le développement de ce système avec comme objectifs de proposer une syntaxe de définition dynamique de prédicats flous, une clause de regroupement flou (group by) et également d'améliorer les performances en s'appuyant sur des structures de données internes plus efficaces. Nous développons également une interface graphique permettant à des utilisateurs de construire des requêtes floues sans disposer de connaissances sur les langages d'interrogation formels de BD ni sur la logique floue.

Références

1. Dubois, D., Prade, H. : Using fuzzy sets in flexible querying : Why and how ? In : Proc. of the 1996 Workshop on Flexible Query-Answering Systems. (1996) pp. 89–103
2. Bosc, P., Pivert, O. : SQLf : a relational database language for fuzzy querying. IEEE Transactions on Fuzzy Systems **3** (1995) 1–17
3. Bosc, P., Pivert, O. : SQLf query functionality on top of a regular relational database management system. In : Pons, O., Vila, M. A., Kacprzyk, J. : Knowledge Management in Fuzzy Databases (2000)
4. Urrutia, A., Tineo, L., Gonzalez, C. : Fsql and sqlf : Towards a standard in fuzzy databases. Fuzzy Databases." In Handbook of Research on Fuzzy Information Processing in Databases (2008) 270–298
5. Kacprzyk, J., Ziolkowski, A. : Database queries with fuzzy linguistic quantifiers. Systems, Man and Cybernetics, IEEE Transactions on **16** (1986) 474 –479