Responsables du projet :
Olivier Pivert et Grégory Smits
Université de Rennes 1 / IRISA / PILGRIM
ENSSAT de Lannion
6, rue de Kerampont - BP 80518
22305 Lannion cedex

le 20 juin 2011

Prestataire pour l'implémentation :
Thomas Girault
Girault R&D

De PostgreSQL à PostgreSQL-f

Cahier des charges

Les développements s'effectueront sur la version 9.1 de PostgreSQL et seront accompagnés d'une documentation utilisateur ainsi qu'une documentation technique de développement détaillant les stratégies d'implémentation utilisées. Ces documentations seront conformes aux formats standards utilisés par PostgreSQL. La documentation sera écrite en français conformément aux accords passés avec l'association PostgreSQL francophone.

Les lots marqués comme [BASE] sont indispensables pour fournir une version exploitable de PostgreSQL-f et les lots marqués comme [EXTENSION] reposent sur des modifications plus profondes du fonctionnement de traitement des requêtes SQL.

Lot 1 : Prise en compte de prédicats floues [BASE]

a) Par l'intermédiaire de fonctions définies en *pl/pgsql* ou *C*, prendre en compte l'intégration de prédicats flous dans la clause *WHERE* de la requête. Ceci induit une modification dans la stratégie de construction de la relation résultat qui doit contenir les tuples pour lesquels le degré de satisfaction du prédicat est strictement supérieur à 0.
ex. : ``*SELECT* * *FROM* employe *WHERE* **jeune**(age)"

b) Prendre en compte la définition de prédicats flous à l'aide de l'opérateur **is** ou suggérer une syntaxe approachée si la modification du sens de l'opérateur prédéfini **is** est problématique.
ex. : ``*SELECT* * *FROM* employe *WHERE* age *is* **jeune**"

c) Prendre en compte des conjonctions et disjonctions de prédicats booléens et flous. Dans un premier temps la conjonction sera définie par la t-norme **min** et la disjonction par la t-conorme **max**.
ex. : ``*SELECT* * *FROM* employe *WHERE* age *is* **jeune** and (dept = 'finance' *OR* salary *is* **high** "

d) Prendre en compte des modifications dynamiques de la t-norme et de la t-conorme pour utiliser d'autres normes, conormes telles que probabilistes, Lukasiewich, etc.
ex. : ``*SET* **NORME probabiliste**;"

e) Prendre en compte des seuils quantitatifs et qualitatifs dans la définition de la requête. Le seuil quantitatif **K** permet de limiter le nombre de résultats retournés et le seuil qualitatif **ALPHA** de filtrer les résultats peu satisfaisants. Voici quelques syntaxes acceptables pour introduire ces seuils. La syntaxe la plus simple à implémenter sera choisie
ex. : ``*SELECT* **50 0.8** * *FROM* employe *WHERE* age *is* **jeune**", où K=50 et ALPHA=0.8
ex. : ``*SELECT* **[50,0.8]** * *FROM* employe *WHERE* age *is* **jeune**", où K=50 et ALPHA=0.8
ex. : ``*SELECT* **K=50 ALPHA=0.8** * *FROM* employe *WHERE* age *is* **jeune**", où K=50 et ALPHA=0.8

ex. : ``**SET K 50; SET ALPHA 0.8;** *SELECT* * *FROM* employe *WHERE* age *is* **jeune**", où K=50 et ALPHA=0.8

Lot 2 : Quantificateurs et propositions quantifiées floues [BASE]

f) Prendre en compte des quantificateurs définis par des fonctions *pl/pgsql, C* ***ou*** *pl/python*, tout d'abord à l'aide d'une syntaxe prefixée puis une syntaxe infixée avec l'opérateur **is**.
ex. : ``*SELECT* * *FROM* employe *WHERE* **tres**(**jeune**(age))"
ex. : ``*SELECT* * *FROM* employe *WHERE* age *is* **tres jeune**"

g) Prendre en compte des propositions quantifiées qui permettent de traiter des requêtes telle que :
ex. : ``*SELECT* * *FROM* employe *WHERE* **la_plupart**(age *is* jeune, dept = 'finance', salary *is* high)"
ex. : ``*SET* **QUANTIFIER OWA**; *SET* **QUANTIFIER ZADEH**;"

Le quantificateur la_plupart sera défini comme une fonction externe (pl/pgsql, C ou pl/python) et différentes implémentations (Zadeh, Yager Competitive Type Aggregation et Yager OWA) seront proposées pour étudier la complexité de chacune (voir le complément d'explications sur les quantificateurs fourni).

h) Adapter l'implémentation des quantificateurs pour prendre en compte des opérateurs de moyenne arithmétique, geométrique, harmonique et éventuellement pondérée avec l'introduction de poids dans la proposition.
ex. : ``*SELECT* * *FROM* employe *WHERE* **arith_mean**(age *is* jeune, 0.5, dept = 'finance', 0.2, salary *is* high, 0.3)"

Lot 3 : Opérateurs graduels [BASE]

i) En plus des opérateurs classiques de comparaison (<, >, <=, ...) et d'appartenance (in), prendre en compte des opérateurs graduels (~, in~, …) associés à des fonctions de distances définies comme des fonctions *pl/pgsql, C* ou *pl/python*.
ex. : ``*SELECT* * *FROM* employe *WHERE* age ~ 50;"
ex. : ``*SELECT* * *FROM* employe *WHERE* dept **in~** ('accounting', 'finance', 'R&D');"

Lot 4 : Requêtes imbriquées [BASE]

j) L'opérateur *in* permet également de tester l'appartenance d'une valeur dans une relation résultant de l'exécution d'une requête imbriquée. Cet opérateur peut facilement être étendu pour tester l'appartenance d'une valeur à une relation floue. L'opérateur inF doit être associé à une mesure de distance sur le domaine de définition de l'attribut concerné (nodep dans l'exemple ci-dessous). Il faudra proposer une solution pour déclarer la mesure de distance à utiliser. Cet mesure correspond à une fonction pl/pgsql, C ou pl/python prenant deux paramètres sur lesquels la distance doit être calculée et retourne une distance sur l'intervalle unité.
ex. : ``*SELECT* * *FROM* employe *WHERE* nodep **inF** (*SELECT* nodep *FROM* department *WHERE* budget *is* low);"

Lot 5 : Définition dynamique de prédicats flous [EXTENSION]

k) Prendre en compte la définition dynamique de prédicats flous dans la requête au lieu de reposer uniquement sur des prédicats définis au préalable par l'intermédiaire de fonctions pl/pgsql. Prendre en compte une clause supplémentaire de définition des prédicats flous comme le suggère la syntaxe suivante où la fonction d'appartenance trapézoïdale est donnée pour le prédicat *jeune* ainsi que la définition discrète du prédicat *élevé* :
ex. : ``*SELECT* * *FROM* employe *WHERE* age *is* jeune and grade is eleve **WITH** jeune (null, null, 30, 38) *AND* élevé (technicien/0.4, ingenieur/0.6, mcf/0.7, professeur/0.8, directeur/0.9);"

Lot 7 : Groupement de données à partir de partitions floues et prédicats flous de restriction de groupes [EXTENSION]

l) La clause *GROUP BY* de *SQL* permet de regrouper des tuples qui possèdent des valeurs identiques sur les attributs de regroupement spécifiés dans la clause *GROUP BY*. Dans l'article [BOSC2010] une stratégie de groupement flou est proposée. Cette stratégie repose sur la définition d'une partition floue du domaine de définition du (des) attribut(s) concerné(s) par le regroupement. Il faut donc compléter la syntaxe des requêtes SQL pour prendre en compte une clause spécifiant la partition à utiliser. Dans l'exemple ci-dessous, `recente' et 'ancienne' correspondent à des ensembles flous définis par des fonctions pl/pgsql. La syntaxe de la définition des partitions peut évidemment évoluer en fonction des particularités syntaxiques de PostgreSQL.

m) Seules les fonctions d'agrégat dérivées de *count* peuvent être utilisées. Implémenter les fonctions **count** et **count-g** définites dans [BOSC2010].

n) La clause HAVING permet d'introduire des conditions de sélection des groupes construits par la clause GROUP BY. Prendre en compte l'intégration de conditions floues dans la clause HAVING afin d'associer à

ex. : ``*SELECT* * *FROM* employe *GROUP BY* dernièreAugmentation *USING* **part(dernièreAugmentation) = (recente,ancienne)**; "