Préférences et bipolarité dans les langages des requêtes
Preference and bipolarity in query languages

Ludovic LIETARD [1] Daniel ROCACHER [2] Salah-Eddine TBAHRITI [2]

[1] IRISA/IUT Lannion
[2] IRISA/ENSSAT
BP- 150, 22302 Lannion Cedex, France, ludovic.lietard@iut-lannion.fr
6, rue de Kerampont, BP- 80518, 22305 Lannion Cedex, France, {rocacher | tbahriti}@enssat.fr

Résumé :
L'intégration des préférences d'utilisateur dans les requêtes adressées aux bases de données permet d'obtenir des réponses plus pertinentes. Ce papier propose de classifier les différentes propositions intégrant les préférences d'utilisateur dans des requêtes. L'accent est mis sur les différentes expressions des préférences ainsi que sur la notion de la bipolarité de ces préférences. Les différents systèmes de requêtes intégrant les préférences sont positionnés par rapport à cette classification.

Mots-clés :
Base de données relationnelle, langage de requête, préférence, interrogation flexible

Abstract :
The integration of user preferences into queries addressed to a database management system allows to obtain more relevant answers. This paper proposes to classify the different propositions made to integrate user preferences inside queries (a particular attention is put on the different expressions of preferences and the impact it has on the result). The different query languages which integrate user preferences are situated with respect to this classification.

Keywords :
Relational database, query language, preference, flexible query

1 Introduction

La quantité d'information gérée par les systèmes de bases de données devient de plus en plus grande et il est nécessaire que les systèmes d'interrogation deviennent de plus en plus performants. Cette performance peut se mesurer en terme de temps de réponse ou en terme de qualité de l'information délivrée. Un des éléments clés de la qualité est la pertinence des réponses, en particulier, par la prise en compte des préférences des utilisateurs dans les requêtes. Il a été montré que la théorie des ensembles flous offre des outils efficaces pour l'expression des requêtes incorporant des préférences utilisateurs dans le contexte des bases de données relationnelles.

Plus précisément, des conditions avec préférences permettent de décrire le caractère personnalisé des besoins de chaque utilisateur. Un exemple de telles conditions est le suivant, l'utilisateur préfère une voiture *'verte'* à une voiture *'rouge'* ou à une voiture *'noire'*.

En général, il possible de distinguer deux familles d'approches pour l'expression des préférences. Une approche implicite où chaque valeur d'attribut est associée à un score, une valeur étant préférée à une autre si elle a obtenu un meilleur score. Une approche explicite où les préférences sont définies en comparant les valeurs d'attributs deux à deux.

Par ailleurs, dans une requête, les préférences peuvent se présenter de façon *bipolaire* [5], c'est-à-dire, que l'on distingue des préférences obligatoires et des préférences optionnelles. Une réponse à une telle requête doit satisfaire toutes les préférences obligatoires et idéalement, satisfaire au mieux les préférences optionnelles.

L'objectif de cet article est de préciser les caractéristiques des approches implicite et explicite pour l'expression de préférences ainsi que celle de la bipolarité. L'accent est mis sur les conséquences qu'ont ces techniques sur les réponses obtenues.

La section suivante étudie et caractérise les deux approches d'expression de préférences, tandis que la section 3 présente la notion de bipolarité des préférences. Dans la section 4, les principaux systèmes d'interrogation avec préférences sont positionnés par rapport à ces trois aspects.

2 Préférences et relations d'ordre

Un des objectifs d'un système d'interrogation prenant en compte des préférences exprimées par les utilisateurs, est de classer les n-uplets de la base en fonction de ces préférences.

Des préférences élémentaires sont définies sur les valeurs d'attributs, puis composées ensuite afin de définir des préférences plus complexes. Pour un attribut donné, deux façons générales de procéder sont utilisées pour exprimer les préférences des utilisateurs : implicite ou explicite.

Dans le cas d'une expression implicite des préférences, un score, généralement dans $\mathbb{R}$, est défini pour chaque valeur du domaine de l'attribut. Une valeur est préférée à une autre si elle a obtenu un meilleur score. Par exemple, un score peut être une distance par rapport à une valeur optimale. Plus la distance est faible, plus l'élément est préféré.

Exemple 2.1. Soit le schéma de la relation voiture d'une base de données (cf. figure 1). Des scores sont affectés aux valeurs de l'attribut *Marque* comme suit : (BMW = 3, Audi = 2, VW = 1). Plus le score est élevé plus la marque est préférée. Ainsi, une préférence pour les voitures de *grande marque* est alors exprimée.◆

Id	Marque	Prix	Consom.	Nb_Ch	Couleur
#1	Bmw	30000	9l/100	90	Verte
#2	Audi	15100	8l/100	91	Verte
#3	VW	15000	7/100	91	Noire

Figure 1. Relation *voiture*

Les ensembles flous [9] constituent un cadre plus spécifique dans lequel cette idée peut être instanciée. Ainsi, la préférence pour les grandes marques est exprimée par l'ensemble flou suivant :
Grande marque = {1/BMW, 0.7/Audi, 0.3/VW}.
Plus le degré est grand, plus la valeur est préférée. Une conséquence importante de l'utilisation de fonctions de score est qu'elles induisent un ordre total sur les valeurs de chaque attribut.

Dans le cas d'une expression explicite des préférences, les valeurs d'attribut sont comparées les unes par rapport aux autres. En conséquence, ces valeurs peuvent être totalement ou partiellement ordonnées, ce qui est illustré par l'exemple 2.2.

Exemple 2.2. Pour l'attribut couleur, un utilisateur préfère la valeur '*vert*' à la valeur '*rouge*' et préfère ces deux couleurs à toutes autres, qui sont considérées comme étant égales par ailleurs. Cela implique un ordre total sur l'ensemble des couleurs. En revanche, si l'utilisateur ne fournit aucune information pour les autres couleurs, un ordre partiel est alors obtenu (il est impossible de comparer le bleu et le jaune).◆

Dans un cadre plus général, l'objectif est de pouvoir classer les n-uplets à partir de plusieurs préférences élémentaires. Deux directions peuvent être suivies, l'une basée sur la comparaison des vecteurs de scores (c.f. section 2.1) l'autre basée sur la comparaison explicite inter-n-uplets (c.f. section 2.2).

2.1 Préférences élémentaires définies par des scores

Dans le cas où les préférences élémentaires sont exprimées par des scores, chaque n-uplet t_i est associé à un vecteur de scores (s^i_1,…, s^i_n), chaque s^i_j correspond à l'évaluation d'une préférence sur le n-uplet t_is. Deux hypothèses sont à considérer selon que les scores sont commensurables ou pas.

Dans l'hypothèse de **commensurabilité**, les scores d'un vecteur sont tous comparables et basés sur un référentiel d'interprétation commun. Ils peuvent donc être combinés au moyen d'une fonction d'agrégation (*moyenne*, *moyenne pondérée*, *min*, etc.) pour délivrer une note globale au vecteur. En conséquence, une relation *d'ordre total* est établie entre les vecteurs de scores.

La fonction d'agrégation f permet de calculer un score général pour chacun des n-uplets. Une relation de préférence *R* exprimant que « t_i est préféré à t_j », notée $t_i R\ t_j$, se définit [1] comme suit :
$t_i\ R\ t_j \Leftrightarrow f(s^i_i,\ldots,s^i_n) \geq f(s^j_i,\ldots,s^j_n)$. La fonction f est généralement une fonction numérique ad-hoc et qui peut être définie par l'utilisateur. Notons que, ce mécanisme d'agrégation a un sens lorsque la fonction f définie tient compte du sens relatif des scores qu'elle combine.

Lorsque l'on utilise des prédicats flous, les scores sont des degrés de satisfaction dans [0, 1] et un sens logique leur est attribué. La fonction qui permet de les agréger utilise des opérateurs logiques étendus (*norme triangulaire*, *conorme*, *implication floue*, etc.), ce qui garanti un cadre formel bien fondé. Ainsi, on dispose d'une riche plate-forme d'opérateurs d'agrégation permettant notamment de prendre en compte des effets de compensation et d'importance entre les attributs.

Exemple 2.3. Soit la requête : les voitures de *grande marque* et *pas chère* (les satisfactions des n-uplets – identifiés par leur Id - sont données par la figure 2). Chaque n-uplet est associé à un vecteur de degrés représentant sa satisfaction aux conditions respectives *grande marque* et *pas chère*. Ces degrés sont agrégés suivant une norme triangulaire *min* (exprimant une conjonction) qui donne les degrés finaux : 0.2, 0.7, 0.5. Les n-uplets sont classés comme suit : #2, #3 puis #1, qui signifie aussi que le n-uplet d'Id #2 est préféré au n-uplet d'Id #3 et ce dernier est préféré au n-uplet d'Id #1.◆

Id	*Grande marque*	*Pas chère*
#1	0.9	0.2
#2	0.7	0.7
#3	0.5	0.8

Figure 2. Degrés de satisfaction

Il est également possible dans le cadre des ensembles flous d'appliquer d'autres mécanismes comme les opérateurs, Leximin ou Discrimin [6]. Ces opérateurs offrent un raffinement du minimum.

L'opérateur Leximin (faisant l'hypothèse de commensurabilité) est basé sur une permutation des scores de chaque vecteur afin de pouvoir les comparer. Il est défini comme suit : si t^* et s^* sont deux permutations des vecteurs *t*, respectivement *s*, telles que : $t^*_1 \leq \ldots \leq t^*_n$ et $s^*_1 \leq \ldots \leq s^*_n$ alors $t >_{\text{Leximin}} s \Leftrightarrow \exists\ k \leq n \mid \forall\ i<k\ t^*_i = s^*_i$ et $t^*_k > s^*_k$.
L'opérateur Discrimin procède par suppression des valeurs identiques, puis une agrégation partielle est faite sur les autres valeurs. Il est défini comme suit :
si $D(u, v) = \{i \mid u_i \neq v_i\}$ est l'ensemble d'index pour lesquels les valeurs correspondantes dans les vecteurs de score *u* et *v* sont différentes,
$u >_{\text{Discrimin}} v \Leftrightarrow \min_{i \in D(u,\ v)} u_i > \min_{i \in D(u,\ v)} v_i$.

Dans l'hypothèse de **non commensurabilité** les scores attribués aux différents attributs d'un n-uplet ne sont pas combinables. En conséquence, ils ne peuvent pas être agrégés et seul un **ordre partiel** peut être défini sur les n-uplets. En particulier, l'ordre de Pareto peut être utilisé.

Ordre de Pareto. On souhaite comparer deux vecteurs de valeurs, *v* et *u* tel que $v = (v_1,\ldots,v_n)$, $u = (u_1,\ldots,u_n)$. L'ordre de Pareto est défini par la formule suivante : $v >_{\text{Pareto}} u \Leftrightarrow (\forall i\ v_i \geq u_i$ et $\exists j\ v_j > u_j)$.

Exemple 2.4. Soit la requête suivante qui consiste à trouver les voitures de prix autour de 15.000 €, avec un nombre de chevaux (Nb.Ch) autour de 90 et une consommation (Cons.) autour de 7L/100. Les préférences sur les trois critères sont qualifiées par des distances. (cf. figure suivante).

Id	Prix	Nb. Ch	Cons.	*Dist. Prix*	*Dist. Nb.Ch.*	*Dist. Cons.*
#1	30000	90	9/100	*15000*	*0*	*2*
#2	15100	91	8/100	*100*	*1*	*1*
#3	15000	91	7/100	*0*	*1*	*0*

Puisque l'hypothèse de commensurabilité n'est pas considérée ici, les scores sur les valeurs de différents attributs ne sont pas comparables et ne peuvent être combinés. Le vecteur de score v(#1) = (15000, 0, 2) ne peut pas être comparé aux autres vecteurs v(#2) = (100, 1, 1) et v(#3) = (0, 1, 0). En utilisant l'ordre de Pareto, le résultat de l'exemple 2.4 est donné comme suit : v(#3) $>_{\mathbf{Pareto}}$ v(#2) mais v(#1) ne peut être comparé ni à v(#2) ni à v(#3). Le n-uplet d'Id #3 est donc préféré au n-uplet d'Id #2 et le n-uplet d'Id #1 ne peut être comparé ni au n-uplet d'Id #3 ni à celui d'Id #2.◆

2.2 Préférences élémentaires définies explicitement

Dans certains cas, il est difficile pour l'utilisateur d'exprimer ses préférences au moyen de scores sur les valeurs d'attributs. Il est plus facile de comparer les valeurs d'attributs les unes par rapport aux autres (par exemple, si l'on veut indiquer ses préférences sur des toiles de maître, il est plus facile de les comparer deux à deux). Dans ce contexte, la commensurabilité n'a pas d'objet.

Lorsque l'on considère plusieurs préférences atomiques, la relation de préférence entre deux n-uplets consiste alors à comparer les valeurs de chaque attribut les unes par rapport aux autres.

Exemple 2.5. La préférence « une voiture de couleur verte et qui consomme peu », est définie sur les attributs couleur et consommation. Chaque n-uplet est associé à un vecteur composé d'une couleur et d'une consommation. Par exemple v_1 est associé à *(verte, 8)*, v_2 est associé à *(verte, 9)* et v_3 est associé à *(noire, 7)*. L'ordonnancement des vecteurs se base sur les relations de préférences atomiques et l'on peut utiliser une relation d'ordre partiel définie par l'ordre de Pareto ce qui interdit de prendre en considération des phénomènes de compensation entre différentes préférences contrairement à l'approche par score. Pour l'exemple précédent v_1 est préféré à v_2 et v_3 ne peut pas être comparé à v_1 ou v_2.◆

Il est également possible de définir une préférence explicite entre les n-uplets par une expression logique et de s'affranchir ainsi des préférences atomiques.

Exemple 2.6. Pour une voiture de marque 'BMW', un utilisateur préfère celle de couleur verte, pour les Audi, il préfère celle de couleur bleue.
$t_1 > t_2 \Leftrightarrow$ (t_1.marque = 'BMW' $\wedge$ t_2.marque = 'BMW') et (t_1.couleur = 'verte' $\wedge$ t_2.couleur $\neq$ 'verte') ou
(t_1.marque = 'Audi' $\wedge$ t_2.marque ='Audi') et (t_1.couleur = 'bleue' $\wedge$ t_2.couleur $\neq$ 'bleue').
Ici encore, certains n-uplets peuvent ne pas être comparables, et un ordre partiel est obtenu.◆

3 Bipolarité des préférences

En général, les préférences utilisateurs sont considérées comme étant toutes obligatoires. Cependant, Dubois et Prade ont étudié un cadre informationnel bipolaire [5] dans lequel on distingue, d'une part, des préférences obligatoires, appelées *contraintes*, et d'autre part, des préférences optionnelles, appelées *souhaits*. Les *contraintes* et les *souhaits* sont respectivement définis par un ensemble de valeurs acceptables, noté *C*, et un ensemble de valeurs souhaitées, noté *S*. Les n-uplets satisfaisant les contraintes et les souhaits sont

retournés en priorité à l'utilisateur. Si de telles réponses n'existent pas, les n-uplets satisfaisant seulement les contraintes sont délivrés.

Exemple 3.1. Un utilisateur veut acheter une voiture *pas chère* (< 15000 €) et souhaite qu'elle soit *de couleur verte*. La condition sur le prix est une contrainte qui détermine un ensemble de voitures acceptables (celle dont le prix est < 15000 € par exemple), puis la condition sur la couleur exprime un souhait, qui, s'il est satisfait, favorise la réponse associée.◆

Les utilisateurs caractérisent souvent les contraintes par la négation (voitures *pas* chères) et c'est d'ailleurs la raison pour laquelle elles sont également appelées préférences négatives. Les souhaits étant appelés préférences positives.

La propriété fondamentale des contraintes et des souhaits est que l'ensemble des valeurs souhaitées est un sous-ensemble de celui des valeurs acceptables ($S \subseteq C$). En effet, il est incohérent de souhaiter des valeurs non acceptables (préférer les voitures asiatiques, mais souhaiter une voiture japonaise).

Il est également possible d'envisager des contraintes et des souhaites définis par des ensembles flous. Dans ce contexte la condition d'inclusion $S \subseteq C$ se réécrit $\mu_S(t) \leq \mu_C(t)$, pour tout tuple *t* de la relation.

Les contraintes et les souhaits sont de nature différente (par exemple, un souhait non satisfait ne rejette pas un n-uplet, au contraire d'une contrainte non satisfaite). En conséquence, les degrés exprimant les contraintes et les souhaits sont *non commensurables* et ne peuvent pas être combinés dans une expression logique. Ils doivent être traités indépendamment.

La contrainte étant impérative, il est possible de traiter les n-uplets par un ordre lexicographique sur les contraintes et les souhaits. Ainsi, les souhaits permettent de départager les ex-æquo sur les contraintes et un *ordre total* peut être obtenu sur *C* et *S*.

Ainsi, un n-uplet *u* est classé avant un autre *u'* si $\mu_C(u) \geq \mu_C(u')$ ou $((\mu_C(u) = \mu_C(u'))$ et $\mu_S(u) \geq \mu_S(u'))$.

4 Langages avec préférences

Nous présentons dans cette section, de façon non exhaustive, les principaux langages de requêtes avec préférences. Nous cherchons à les positionner par rapport aux deux axes précédents.

4.1 SQLf

Le langage SQLf [3] étend le langage SQL en permettant à l'utilisateur de construire des requêtes sur des conditions atomiques définies par des ensembles flous. Chaque condition atomique associe un degré de satisfaction $\mu \in [0, 1]$ à une valeur d'attribut. Pour tous les attributs d'un n-uplet, la sémantique des degrés est la même ce qui implique que tous les critères sont *commensurables*.

Une requête SQLf a la syntaxe suivante :

Select	[distinct] [*n*|*t*|*n*,*t*] < *attribut* >
from	< *relations strictes* >
where	< *condition floue* >

où <condition floue> peut incorporer des blocs de requêtes imbriquées ou un partitionnement. Les paramètres *'n'* et *'t'* du bloc *Select* limitent le nombre des réponses en utilisant un calibrage quantitatif (les *'n'* meilleures réponses) ou un calibrage qualitatif (les données qui satisfont la requête avec un seuil supérieur à *'t'*).

Exemple 4.1. Soit la relation *employé* (cf. figure 3), et soit la requête suivante : Trouver les employés d'*environ 40 ans* et qui ont un *salaire élevé*. *Environ 40* et *salaire élevé* sont des critères de sélection définis par des ensembles flous. En particulier, pour les n-uplets de la relation *employé* nous avons :

Salaire élevé = {0.9/2900, 0.8/2800, 0.7/2700}, *Environ 40* = {0.8/38, 0.6/37, 0.8/42}.

Id	*Nom*	*Age*	*Salaire*
#1	Pierre	38	2900
#2	Jean	37	2800
#3	Yvette	42	2700

Figure 3: Relation *employé*

Chaque n-uplet est associé à un vecteur qui représente sa situation vis-à-vis des conditions atomiques. Ainsi, le résultat final est évalué puis calculé à l'aide d'une norme triangulaire (*min* par exemple) pour exprimer une conjonction entre les critères. On obtient alors : Pierre (0.8) > Yvette (0.7) > Jean (0.6) qui signifie que le n-uplet #1 est préféré au n-uplet #3 qui est préféré au n-uplet #2.◆

En SQLf, les préférences sont considérées uniquement comme des *contraintes* et sont prises en compte au travers de l'expression de prédicats flous *commensurables*, modélisés à l'aide d'ensembles flous de valeurs plus ou moins satisfaisantes. En conséquence, les résultats sont *totalement ordonnés*.

De tels prédicats peuvent être combinés au moyen d'une riche plateforme d'opérateurs de la logique floue dont certains, traduisant par exemple des effets de compensations ou d'importances relatives entre critères, n'ont pas de correspondant dans la logique booléenne. Contrairement à d'autres approches (comme PreferenceSQL, cf. section 4.2), la sélection des données et l'exploitation de préférences pour les ordonner sont traitées simultanément.

Bien que SQLf offre aux programmeurs une grande variété d'outils pour construire des conditions graduelles, ce langage ne permet pas d'exprimer des préférences *optionnelles (souhaits)*, seules des préférences *obligatoires* peuvent être exprimées.

4.2 Preference SQL

Preference SQL [7] est une extension du langage SQL considérant les requêtes comme composées de deux parties : l'une booléenne (partie WHERE) visant à sélectionner des n-uplets et l'autre (partie PREFERRING) spécifiant un ordonnancement des éléments sélectionnés :

Select <sélections>
from <liste relations>
where <prédicats *must* >
Preferring <prédicats *light*>

A la différence des prédicats booléens de sélection obligatoires dits « *must* » qui permettent de sélectionner les résultats acceptables sans l'influence des préférences, les prédicats de type « *light* » sont satisfaits au mieux possible et permettent au final d'ordonner ces résultats. Les prédicats « *light* » expriment des préférences sur différents attributs et sont définies explicitement ou implicitement par des distances. Preference SQL se place donc dans le cadre de *non commensurabilité* des préférences ce qui implique un ordre partiel.

La juxtaposition des prédicats « *light* » traduit que les préférences ont toutes la même importance et un ordre de Pareto est utilisé pour classer les résultats. Du fait de l'ordre partiel, Preference SQL ne délivre que les n-uplets non dominés (tel qu'il n'en existe pas de meilleurs). Si aucun n-uplets ne satisfait les prédicats « *light* », tous les n-uplets satisfaisant les prédicats « *must* » sont délivrés. Il est également possible d'attribuer des importances relatives de plus en plus faibles aux prédicats « *light* ».

Nous concluons que Preference SQL suit le modèle bipolaire, les prédicats « *must* » *(conditions booléennes)* représentant les contraintes, les prédicats *light (préférences)* représentent *les souhaits* (préférences optionnelles). En outre l'hypothèse de non commensurabilité n'a pas à être considérée pour les préférences.

4.3 Top-K queries

Dans cette approche [8], une fonction *f* ad-hoc d'ordonnancement (ranking) est utilisée pour

classer les n-uplets correspondant aux préférences et retourne les *k* meilleurs. Cette fonction est calculée sur les valeurs d'attributs numériques et peut intégrer des scores élémentaires (qui peuvent être calculés sur des attributs non numériques).

Exemple 4.2. Soit la relation R de la figure suivante décrivant des personnes par leur nom, leur âge, leur poids et leur taille. Le surpoids d'une personne décrite par un n-uplet est calculé par la fonction suivante: $f(t) = t.poids - (t.taille - 100)$.

Id	*Nom*	*Age*	*Poids*	*Taille*
#1	Pierre	31	90	180
#2	Jean	29	78	160
#3	Yvette	22	65	170

Figure 4: Relation *Personne*

La requête « chercher les deux meilleurs réponses (k = 2) à : trouver les personnes en surpoids » utilise la fonction *f* comme fonction d'ordonnancement.

Le résultat est donné comme suit : $f(t_2)$=18, $f(t_1)$=10 et $f(t_3)$=-5 où les n-uplets t_1, t_2 et t_3 sont respectivement associés aux clés #1, #2, #3. L'ordonnancement des n-uplets est alors : t_2, t_1 puis t_3 et seuls t_2, t_1 sont retournés.◆

Exemple 4.3. Une deuxième requête consiste à « trouver les personnes jeunes en surpoids » où la préférence *être-jeune* est définie par un ensemble flou (un score élémentaire $\mu_{jeune}(t.\text{age})$ pour chaque n-uplet *t*). Dans ce cas, la fonction d'ordonnancement doit rendre *commensurable* le surpoids et le score. Il est possible de la définir par : $f_1(t) = \mu_{jeune}(t.age) + f(t) / poids$ (f étant la fonction définie dans l'exemple 4.2). Ceci signifie qu'un surpoids est d'autant plus important qu'il est associé à un poids léger. Plus un n-uplet *t* décrit une personne jeune et en surpoids, plus la valeur $f_1(t)$ sera grande. Le résultat est donné dans le tableau ci-dessous. ◆

Id	*Age*	*Poids*	μ_{jeune}	*f(t)*	$f_1(t)$
#1	31	90	*0,4*	*10*	*0,55*
#2	29	78	*0,5*	*18*	*0,73*
#2	22	65	*0,8*	*-5*	*-0,87*

Le système *top-k queries* est donc basé sur trois points ; la sélection des n-uplets vérifiant les conditions de la clause WHERE, l'ordonnancement de ces n-uplets suivant une fonction d'ordonnancement et la délivrance d'un nombre *k* de résultats souhaités. Les conditions de la clause WHERE correspondent à des contraintes tandis que l'ordonnancement des n-uplets par la fonction de ranking s'apparente à une notion de souhait. En effet, cette fonction ne permet pas de rejeter des éléments. Notons que la fonction de ranking doit rendre *commensurable* les données qu'elle combine.

4.4 Preference Queries

Cette approche [4] propose un cadre algébrique pour formuler des requêtes avec préférences et l'opérateur algébrique « *winnow* ». Celui-ci permet de sélectionner *les meilleurs* n-uplets, c'est-à-dire, ceux qui ne sont pas dominés au sens de la relation de préférence. Il est défini par la formule suivante: si R est une relation de base de donnée et *r* est une instance de R alors, $w_{\succ}(r) = \{t \in r \mid \neg \exists t' \in r, t' \succ t\}$. En notant $\succ$ la relation de préférence

L'opérateur *Skyline* [2] est une mise en œuvre de *winnow* où les préférences sont spécifiées à partir d'un jeu d'opérations prédéfinies.

L'intérêt de *winnow* se justifie lorsque la relation de préférence délivre un ordre partiel car il permet de sélectionner des éléments non-dominés (et non comparables entre eux). Au niveau syntaxique, il n'existe pas une formulation distincte de notion bipolaire de contraintes et de souhaits. Cependant, il a été démontré [10] qu'une requête bipolaire, s'appliquant sur une relation *T* et où les

contraintes *C* et souhaits *S* sont des conditions booléennes peut se définir au moyen de l'opérateur winnow. En effet, une telle requête bipolaire se traduit par l'opérateur winnow suivant : $w_R(\sigma_C(T))$ où σ est une sélection sur la contrainte *C* et *R* est la relation de préférence définie par : $R(t1, t2) \Leftrightarrow S(t1) \wedge \neg S(t2)$.

5 Conclusion

Les préférences sont principalement employées pour filtrer et personnaliser l'information recherchée par les utilisateurs. Nous avons présenté deux familles d'approches pour les représenter : implicite et explicite. Dans l'approche implicite des mécanismes de scores numériques commensurables ou non sont utilisés pour représenter les préférences. Un ordre total est établi sur l'ensemble des résultats lorsque les préférences sont commensurables et un ordre partiel est obtenu si les préférences sont non commensurables. Dans l'approche explicite les préférences sont spécifiées par des relations binaires de préférences et dans la plupart des cas, un ordre partiel est obtenu sur les n-uplets.

Par ailleurs, les préférences peuvent être considérées comme étant des contraintes (préférences obligatoires) et des souhaits (préférences optionnelles). Il en ressort que cette vision bipolaire des préférences permet d'apporter un raffinement de l'ensemble des réponses : satisfaire les contraintes, puis, si possible, les souhaits.

Nous avons également étudié les principaux systèmes d'interrogation avec préférences. Les systèmes *Preference SQL* et *Preference Queries* se basent sur un ordre partiel, en conséquence, délivrent à l'utilisateur des n-uplets non dominés. *Preference SQL* incorpore également une notion de bipolarité au travers la clause *Preferring*. Le système *top-k queries* utilise une fonction de score *f* ad-hoc et délivre les k-meilleures réponses de l'ordre total obtenu par *f*. Cette fonction de score reste toutefois difficile à établir. Le langage *SQLf* utilise la théorie des ensembles flous pour définir les préférences et fait l'hypothèse de commensurabilité. Il offre un cadre fondé pour combiner des préférences obligatoires. Cependant, il ne permet pas d'exprimer des préférences optionnelles. Il serait intéressant d'introduire cette nouvelle dimension dans *SQLf*.

Références

[1] Agrawal F., Wimmers E.L., ''A Framework for Expressing and Combining Preferences''. In *Proc. of Int. Conf. on Management of Data (ACMSIGMOD)*, pp. 297-306., Dallas, USA, 2000.

[2] Börzsönyi S., Kossmann D., and Stoker K., ''The Skyline Operator''. In. *Proc. 17th Int. Conf. on Data Engineering*, Heidelberg, Germany, pp. 421-430, 2001.

[3] Bosc P., Pivert O., ''SQLf: a relational database language for fuzzy querying''. *IEEE Trans. on Fuzzy Systems*, vol(3) pp.1-17, 1995.

[4] Chomicki J., ''Preference Formulas in Relational Queries''. In *ACM Trans. on Database Systems (TODS)*, vol. 28(4), 2003.

[5] Dubois D., Prade H., ''Bipolarity in flexible quering''. *Proc. of the 5th Int. Conf. on Flexible Query Answering Systems (FQAS)*, Copenhagen, Denmark, 2002.

[6] Dubois D., Fargier H. and Prade H., ''Beyond min aggregation in multicriteria decision: (ordered) weighted Min, Discrimin, Leximin''. In The Ordered Weighted Averaging Operators - Theory and Applications. R.R. Yager, J. Kacprzyk (Eds.), Kluwer Academic Publ., Boston, pages 181-192, 1997.

[7] Kießling W., ''Foundations of Preferences in Database Systems''. In *Proc. of the 28th Int. Conf. on Very Large Databases (VLDB)*, pp. 311-322., Hong Kong, China, 2002.

[8] Li1 C., Soliman M.S., Chang C.K., Ilyas I.F., ''RankSQL: Supporting Ranking Queries in Relational Database Management Systems''. In *Proc. of the 31 th Int. Conf. on Very Large Databases (VLDB)*, pp. 1342 – 1345 Trondheim, Norway, 2005.

[9] Zadeh L.A., ''Fuzzy sets". In *Jour. Of Information and Control*, vol.8, pp. 338-353, 1965.

[10] Zadrozny S., Kacprzyk J., ''Bipolar Queries andQueries with Preferences'' (Invited Paper). In *Proc. of the 17th Int. Conf. on Database and Expert Systems Applications (DEXA'06)*, pp. 415-419 Krakow, Poland, 2006.