REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEINEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE FERHAT ABBAS DE SETIF
UFAS, ALGERIE

MEMOIRE

Présentée à la faculté des sciences
Département D'informatique

Pour l'obtention du diplôme de

MAGISTER

Option : Ingénierie Des Systèmes Informatique

Présentée par : Aïcha AGGOUNE

Contribution à l'étude des requêtes à réponse vide dans un contexte flexible : Une approche basée sur une proximité sémantique entre requêtes

Soutenue le : …/…/2011 **Devant la commission d'examen :**

Mr. A.KHABABA	**Maitre de conférences**	**U.F.A SETIF**	**Président**
Mr. A.MOUSSAOUI	**Maitre de conférences**	**U.F.A SETIF**	**Rapporteur**
Mr. M. TOUAHRIA	**Maitre de conférences**	**U.F.A SETIF**	**Examinateur**

Remerciements

بسم الله الرحمن الرحيم

قال الله تعالى: ﴿ لئن شكرتم لأزيدنكم ﴾ صدق الله العظيم

Je remercie et je loue, d'abord et avant tout ALLAH, sans lequel rien ne se fait ou ne se crée.

Je tiens à exprimer ma profonde gratitude à Monsieur Allel HADJALI, Maitre de conférence à l'IRISA/ENSSAT et à l'Université Rennes 1 de LANNION en France, pour avoir encadré et dirigé mes recherches. Je le remercie pour toute la confiance qu'il a su me porter, et pour la patience, la gentillesse et la disponibilité dont il a fait preuve à mon égard. Ses conseils et remarques constructives m'ont permis d'améliorer grandement la qualité de mes travaux et de ce mémoire.

Je remercie Monsieur Abdelouaheb MOUSSAOUI, Maitre de Conférences à l'université FERHAT ABBES de Sétif en Algérie, pour avoir dirigé mes recherches. Je le remercie également pour toutes les discussions que nous avons eues, desquelles ont découlés de nombreux conseils et remarques constructives. Il peut être assuré de mon sincère respect et de ma profonde gratitude.

Je remercie ensuite le président de jury et l'examinateur, pour avoir accepté et juger ce travail.

Je ne peux enfin clôturer ces remerciements sans remercier du fond du cœur mes parents, qui n'ont eu de cesse de me soutenir et de croire en moi pendant ces looooooooongues études.

Aïcha

Table des matières

Liste des tableaux

Liste des figures

Liste des algorithmes

Introduction Générale

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, qui sont appelées dans cas des requêtes à préférences. Il a été montré que la théorie des sous ensembles flous offre des outils efficaces pour l'expression des requêtes à préférences dans le contexte des bases de données relationnelles. Dans ce cadre, les requêtes sont dites des requêtes flexibles (ou floues).

L'interrogation flexible des BD à l'aide des requêtes flexibles, vise à étendre le comportement bipolaire de l'interrogation classique (données qui satisfont les critères de recherche et celles qui ne les satisfont pas) en utilisant le concept de prédicat flou pour exprimer des préférences de manière à retourner un ensemble de réponses discriminées plutôt qu'un ensemble plat de réponses. Ainsi, une réponse retournée par une requête flexible sera plus au moins pertinente selon son degré de satisfaction aux contraintes d'interrogation.

L'utilisation du formalisme des ensembles flous pour exprimer les requêtes à préférences est motivé par plusieurs raisons, en particulier : le fait qu'il est très bien adapté à l'interprétation de termes linguistiques, qui constituent un moyen commode pour un utilisateur d'exprimer ses préférences et le fait qu'il limite dans une large mesure le risque d'obtention d'une réponse vide, qui représente l'un des problèmes les plus connus dans le domaine de l'interrogation des BD. Cependant, ce risque n'est pas totalement éliminé et plusieurs approches ont été proposées dans la littérature, que l'on peut les classifier en deux grandes familles : les approches guidées par la requête (qui a échoué) ; leur principe est d'agir seulement sur les conditions impliquées dans la requête échouée en appliquant le mécanisme de relaxation sur cette requête afin de rendre moins sélective et élargir le volume du résultat. La seconde famille comprend, des approches guidées par un "Workload of past queries" ; leur principe est de proposer des réponses approximatives à la requête échouée, en exploitant un ensemble "Workload" des requêtes précédemment exécutées par le système dont les réponses sont non vides et une mesure de proximité sémantique entre requêtes.

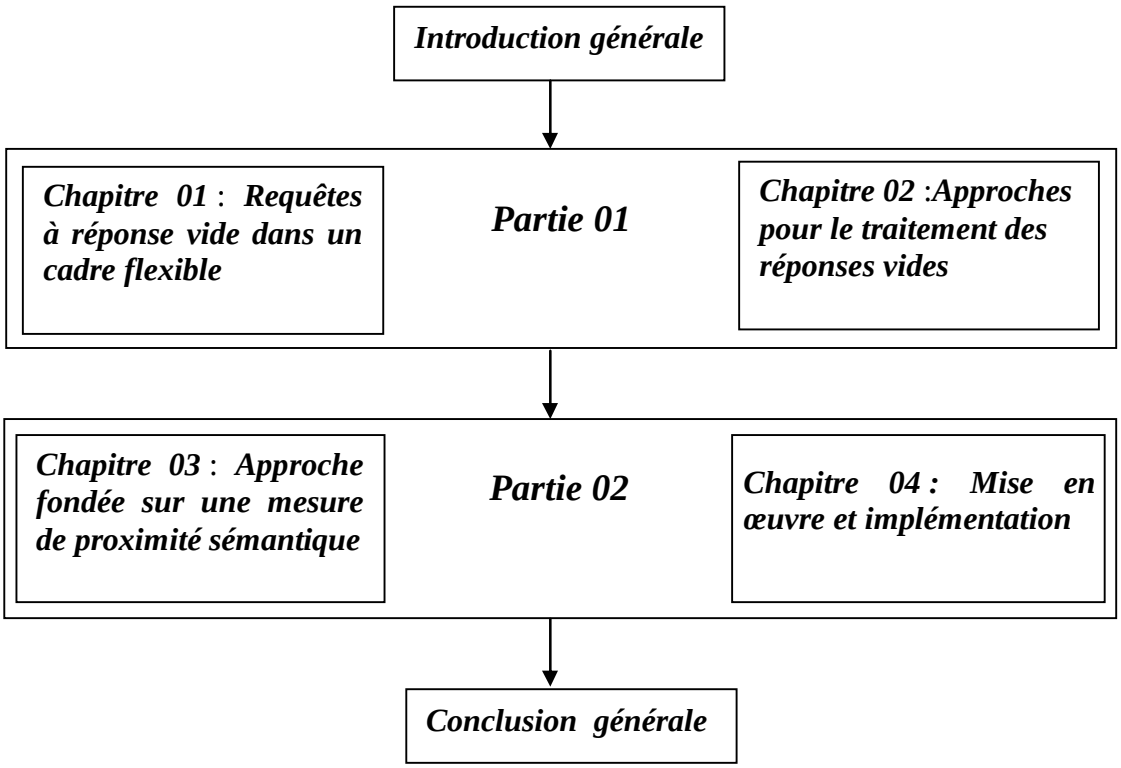
L'objectif de ce mémoire de magister alors, est de proposer une approche pour le traitement des requêtes flexible à réponses vides, en suivant le principe de la seconde famille d'approches précitée. Ainsi, le développement d'un algorithme permettant de rechercher la requête la plus proche sémantiquement parlant à une requête à réponses vides. L'outil de base de cette approche est une mesure de distance particulière dite la distance de Hausdorff, afin d'estimer la proximité sémantique entre requêtes.

Ce mémoire est organisé en deux parties : La première, présente le contexte dans lequel se situe notre travail, c'est-à-dire le problème des requêtes flexibles à réponse vide et la seconde partie, décrit notre contribution, à savoir, la solution que nous proposons ainsi que l'expérimentation que nous avons menée. Nous finissons par une conclusion générale qui, résume brièvement le travail réalisé avec quelques perspectives de recherche future.

La première partie de mémoire est structurée en deux chapitres : le premier, consiste à présenter le problème des requêtes à réponses vides, en donnant les concepts liés au ce problème, à savoir les requêtes flexibles, la présentation de problème et ses causes. Le deuxième chapitre, présente les deux familles d'approches proposées dans la littérature pour le traitement des réponses vides. Pour chaque famille, nous décrivons quelques approches avec des exemples illustratifs. Nous nous sommes efforcés aussi de faire une évaluation entre ces approches.

La seconde partie est structurée aussi en deux chapitres : le premier, présente notre contribution qui consiste à proposer une approche fondée sur une mesure de proximité sémantique entre requêtes, en développant pour celle-ci un algorithme de recherche de la requête la plus proche à une requête à réponse vide. Le second chapitre décrit, l'implémentation et la mise en œuvre de notre approche proposée, les résultats des expérimentations effectuées pour évaluer les performances de l'approche que nous avons proposée.

Pour illustrer mieux le plan du notre mémoire, nous pouvons donner le schéma suivant:



PARTIE 01

ETAT DE L'ART

Chapitre 01

Requêtes à Réponse Vide dans un Cadre Flexible

Chapitre 02

Approches pour le Traitement des Réponses Vides

Chapitre 01

Requêtes à Réponses Vides dans un Cadre Flexible

1. INTRODUCTION

Ces dernières années, le domaine des bases de données a vu émerger des travaux visant à rendre plus flexibles les systèmes, et ce notamment par la prise en compte des préférences des utilisateurs afin de mieux personnaliser les réponses désirées. Un des formalismes permettant d'exprimer les préférences dans les requêtes est la théorie des sous ensembles flous. Dans ce contexte, les requêtes sont dites des requêtes flexibles dont l'avantage est de retourner un ensemble de réponses discriminées plutôt qu'un ensemble plat de réponses.

L'interrogation flexible étend les fonctionnalités des systèmes d'interrogation classiques, cependant les utilisateurs peuvent être confrontés avec les deux problèmes suivants : les réponses pléthoriques et les réponses vides. Dans le cadre de notre travail nous nous intéressons au dernier problème c.à.d, le problème des requêtes à réponse vide (PRV).

Ce chapitre représente un état de l'art qui décrit dans un premier temps quelques concepts de base, à savoir la recherche d'information, quelques types de requêtes, les requêtes flexibles. On présente en plus un rappel sur les notions de base de la théorie des sous ensembles flous qui décrit le formalisme utilisé pour exprimer les préférences dans les requêtes. Dans un second temps, nous présentons le problème des requêtes à réponses vides et ses causes.

2. RECHERCHE D'INFORMATION ET REQUÊTES

Depuis que les ordinateurs sont apparus, des milliards d'informations y ont été enregistrées dans plusieurs bases de données (BD), dans divers domaines de connaissances et sous diverses formes (numériques, textes, images etc.). Etant donné que les ressources informationnelles sont de plus en plus accessibles aux utilisateurs, le principal problème aujourd'hui est de savoir comment accéder à l'information dont on a besoin. Ce qui explique l'importance du domaine de la recherche d'informations et de l'interrogation de bases de données.

2.1. La recherche d'information

La Recherche d'Information (RI) est une branche de l'informatique qui s'intéresse à l'acquisition, l'organisation, le stockage, la recherche et la sélection d'information. D'un point de vue utilisateur, l'accès à l'information peut être effectué à travers un Système de Recherche d'Information (SRI), ou bien à travers un système de filtrage d'information. Un

SRI est un ensemble de programmes informatiques qui a pour but de sélectionner des informations pertinentes répondant à des besoins utilisateurs, exprimés sous forme de requêtes [1]. Un système de filtrage peut être défini comme un processus qui permet d'extraire à partir d'un flot d'informations (News, e-mail, actualités journalières, etc.), celles qui sont susceptibles d'intéresser un utilisateur ou un groupe d'utilisateurs ayant des besoins en information [2].

2.2. Définition d'une requête

Une requête est une instruction permettant de sélectionner un ensemble de données, répondant à un ou plusieurs critères, issues d'une ou plusieurs tables. La requête peut afficher tout ou une partie de l'ensemble des champs dans une structure totalement différente de la structure de l'enregistrement des tables "source" [5].

L'interrogation de base de données à l'aide des requêtes a trois fonctions principales [3] :

- La réalisation de vues présentant tout ou partie de l'information contenue dans la BD.
- La maintenance de la BD, cette opération consiste à archiver et / ou supprimer des enregistrements obsolètes, mettre à jour des données révisables, rechercher et supprimer les doublons indésirables,…etc.
- La recherche d'information dans la BD, cette opération consiste à créer une sous-table contenant les enregistrements répondant à certains critères et appartenant à certains champs. Elle porte à la fois sur les lignes et les colonnes d'une table, ou de plusieurs tables liées par des relations.

2.3. Types des requêtes

Pour assurer les trois fonctions précitées, différents types de requêtes ont été créés, que l'on peut les classifier selon trois critères principaux: le type d'opération exécutée, le nombre d'attributs et le formalisme utilisé (booléen/Flou).

2.3.1. Selon le type d'opération

Il existe différents types de requêtes selon l'opération exécutée et que l'on trouve dans presque tous les SGBD [3] :

- ***Requêtes de Sélection :*** La sélection est l'outil de recherche d'information par excellence, même si ce n'est pas le seul qui soit utilisé. Ces requêtes sont dotées de deux

perfectionnements importants : la jointure et le regroupement. Elles permettent de sélectionner des enregistrements, de faire des calculs et des regroupements. Elles ressemblent beaucoup aux filtres, mais permettent, en plus, de travailler sur plusieurs tables simultanément.

- ***Requêtes d'Ajout :*** Ces requêtes ajoutent les données à la fin d'une table déjà existante.

- ***Requêtes de Maintenance :*** concerne principalement : la mise à jour et la suppression. La première permet de modifier le contenu de certains champs, la seconde permet de supprimer certains enregistrements.

- ***Requêtes d'Analyse Croisée :*** l'analyse croisée est une spécificité d'Access. Comme son nom l'indique, c'est un outil d'analyse qui permet, sous certaines conditions, de réorganiser complètement une table. Ces requêtes présentent ses résultats sous forme de tableau (ex. Excel). On l'utilisera pour comparer des valeurs et dégager des tendances.

2.3.2. Selon le nombre d'attributs

On peut distinguer trois types de requêtes selon le nombre d'attributs utilisé pour effectuer l'interrogation d'une BD, qui sont : requêtes atomiques, requêtes composées et requêtes imbriquées [6].

- ***Requêtes atomiques :*** appelés aussi requêtes simples, où la recherche est faite sur un seul terme (attribut), par exemple : la requête « trouver les employés avec un âge supérieur à 24 ans» qui sélectionne sur le critère « âge».

- ***Requêtes composées :*** les requêtes composées permettent d'effectuer la recherche avec un jeu de termes donné relient entre eux par des connecteurs de conjonctions (et) ou de disjonctions (ou). Par exemple : « trouver les employés avec un âge supérieur à 24 ans et un salaire entre 1000€ et 5000€», cette requête permet de faire la recherche suivant les deux critères ; âge et salaire relient par le connecteur "ET".

- ***Requêtes imbriquées :*** une requête imbriquée est une requête composée de plusieurs sous requêtes dans une instruction SELECT, INSERT, UPDATE ou DELETE, ou dans une autre sous-requête. Une requête imbriquée est également appelée "requête externe" ou "sélection externe" et les sous requêtes contenant la requête imbriquée sont aussi appelées "requêtes internes" ou "sélection interne" [4].

Exemple :

Considérons les extensions des relations Emp de schéma (emp#, âge, dep#) et Dept de schéma (dep#, budget) avec la requête visant à trouver les départements à budget moyen où ne travaille aucun employé jeune. Cette requête floue s'exprime, sous la forme imbriquée suivante:

```
SELECT dep# FROM Dept
 WHERE budget BETEWEEN 10000€ and 30000€
 and dep# not in
(SELECT dep# FROM Emp WHERE âge<30).
```

2.3.3. Selon le formalisme utilisé

Il existe deux formalismes de représentation des requêtes : la logique classique (booléen ou Crisp) dans ce cas la requête est dite "Requête booléenne" et la logique floue dans ce cas la requête est dite requête flexible ou requête floue (appelée aussi requête graduelle) :

- ***Requêtes booléennes :*** une requête booléenne retourne un résultat ou rien du tout. Elle suit le principe de "Tout ou rien". L'interrogation booléenne ne permet à l'utilisateur ni d'utiliser des termes linguistiques vagues et imprécis dans les critères de qualification des données recherchées ni d'exprimer des préférences entre ces critères, ce qui est souvent une demande légitime des utilisateurs [7].

- ***Requêtes flexibles :*** les requêtes flexibles permettent d'exprimer des préférences utilisateurs à l'aide de la théorie des ensembles flous dont l'avantage est de retourner un ensemble de réponses discriminées plutôt qu'un ensemble plat de réponses [6]. Elles permettent aussi d'améliorer la capacité d'expression des langages de requêtes et la satisfaction des besoins des utilisateurs. Les attributs de ces requêtes ne sont alors plus en principe « tout ou rien » mais peuvent être plus ou moins satisfaits [8].

Après avoir présenté les différents types de requêtes, nous présentons dans ce qui suit la définition détaillée de concepts des requêtes flexibles.

3. INTERROGATION FLEXIBLE DE BASE DE DONNÉES

L'interrogation des bases de données nécessite une connaissance précise et détaillée des données et de leur organisation. L'interrogation flexible tente de rendre l'interrogation classique des BD plus souple pour les utilisateurs. Elle permet de prendre en compte des préférences de l'utilisateur dans les requêtes qui sont dites des requêtes à préférences où les ensembles flous sont utilisés comme cadre de modélisation de ces préférences. Dans ce contexte, les requêtes sont dites des requêtes flexibles dont l'avantage est de retourner un ensemble de réponses discriminées plutôt qu'un ensemble plat de réponses.

Nous décrivons dans un premier temps les notions de base de la théorie des sous ensembles flous puis nous présentons en détaille la notion de requêtes flexibles.

3.1. Notions de base sur la théorie des sous ensembles flous

La théorie des sous ensembles flous à été établie par le professeur d'automatique de l'université de Californie à Berkeley, Lotfi A Zadeh qui réalise un premier article dans ce domaine publié en 1965 généralisant la théorie des ensembles classique. Cette théorie (La théorie des sous ensembles flous) permet de présenter des classes d'objets dont les frontières sont mal définies [9].

Un sous ensemble flou E de l'ensemble flou X est défini par une fonction caractéristique μ a valeur dans l'intervalle [0, 1] de *R*, telle que $\mu_E(x)$ exprime dans quelle mesure l'élément x appartient au sous ensemble flou E.

Exemple :

Une fonction d'appartenance possible pour définir le sous ensemble flou « Avoir une vingtaine d'années » est représentée en figure 1.1.

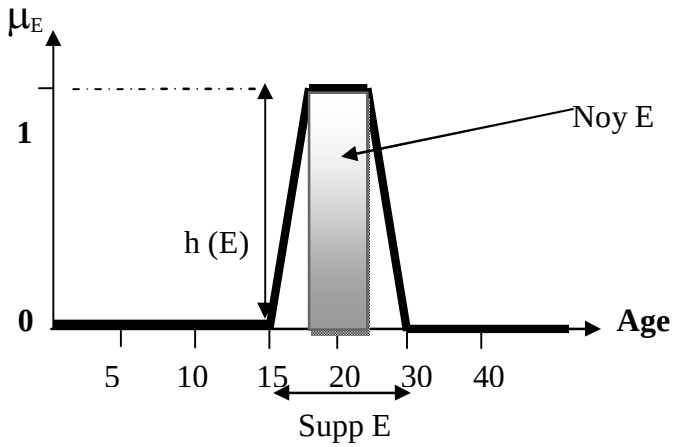


Figure 1.1 Fonction d'appartenance « Avoir une vingtaine d'années ».

On voit que la fonction d'appartenance peut être fixée selon le contexte et l'utilisateur.

Les notions suivantes caractérisent un ensemble flou E [9] :

- *Support de E :* Supp E = $\{x \in X, \mu_E(x) > 0\}$.
- *Hauteur de E* : h (E) =$Sup_{x \in X}$ $\mu_E(x)$, E dit normalisé si h (E) =1.
- *Noyau de E (Core)* : Noy E = $\{x \in X, \mu_E(x)=1\}$.
- *Cardinalité scalaire de E :* $|E|=\sum_{x \in X} \mu_E(x)$.

Si A et B sont deux sous ensembles flous du référentiel X, on dit que :

- A est plus spécifique que B ssi Noy A $\not\subseteq$ Noy B et Supp A $\subseteq$ Supp B.
- A est plus précis que B ssi Noy A= Noy B et Supp A $\not\subseteq$ Supp B.

Opérations sur les ensembles flous :

Les opérations sur les ensembles flous sont des extensions des opérations connues sur les ensembles classiques. Elles permettent de d'écrire des combinaisons logiques entre notions floues, c'est-à-dire de faire des calculs sur des degrés de vérité. En effet, si les valeurs d'appartenances sont restreintes aux valeurs 0 et 1, alors les opérateurs flous (Et, Ou, négation,…etc) devraient donner les mêmes résultats que leurs contreparties classiques [9] [6].

- *Egalité :* A=B ssi $\forall x \in X, \mu_A(x)= \mu_B(x)$
- *Inclusion :* A $\subseteq$ B ssi $\forall x \in X, \mu_A(x) \leq \mu_B(x)$

- *Intersection :* L'opérateur logique correspondant à l'intersection d'ensembles est le ET. A∩B est défini par : $\forall x \in X$, $\mu_{A\cap B}(x) = \text{Min}(\mu_A(x), \mu_B(x))$.

- *Union :* L'opérateur logique correspondant à l'union d'ensembles est le OU. A∪B est défini par : $\forall x \in X$, $\mu_{A\cup B}(x) = \text{Max}(\mu_A(x), \mu_B(x))$.

Remarque : On défini d'une manière plus générale l'intersection (resp. l'union) par une norme (resp. une co-norme).

Exemple :

Reprenons l'exemple précédent. On considère les personnes ayant « Une vingtaine d'années » et celles « Ayant la majorité » (en pointillés sur la figure 1.2, on considère un sous ensemble non flou).

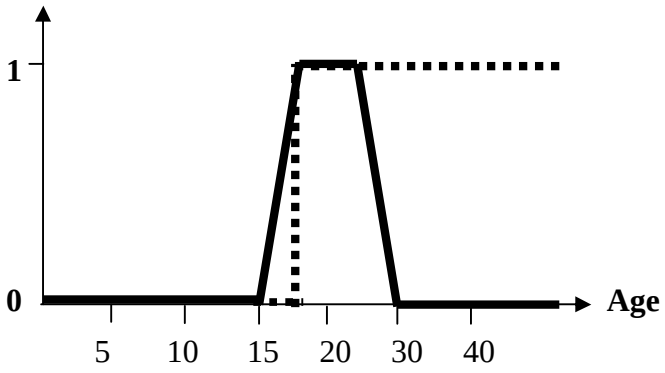


Figure 1.2 Les deux fonctions d'appartenance considérées

Selon les deux opérations : Intersection et Union, on peut caractériser les sous ensembles flous correspondant aux personnes « Ayant une vingtaine d'années et la majorité » (figure 1.3) ainsi que celui des personnes « Ayant une vingtaine d'années ou la majorité » (figure 1.4)

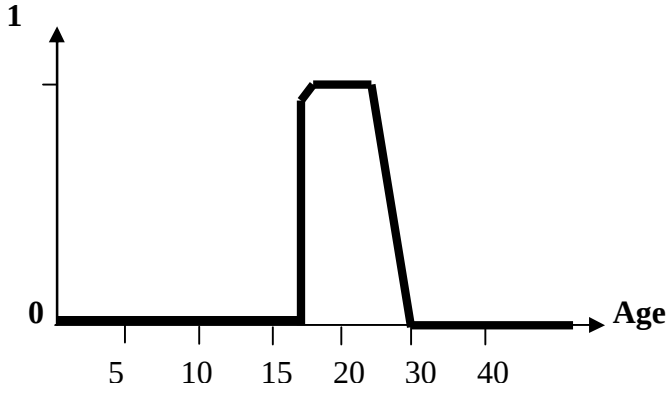


Figure 1.3 Fonction d'appartenance de l'Intersection

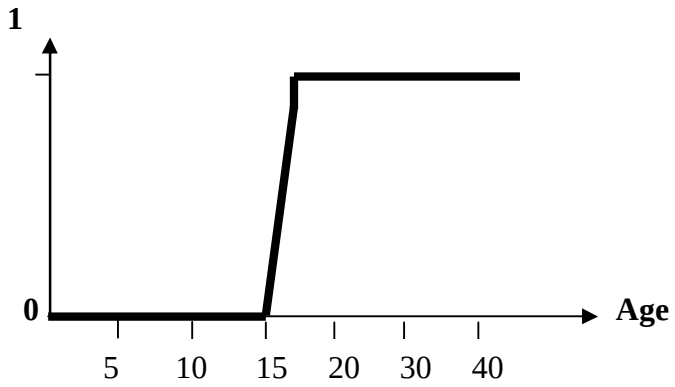


Figure 1.4 Fonction d'appartenance de l'Union

On définit également :

- *Le produit cartésien :* Soit des sous-ensembles flous A_1, A_2, …, A_r définis sur X_1,X_2, …., X_r, respectivement, on définit leur produit cartésien $A = A_1 \times A_2 \times \ldots \times A_r$, comme un sous-ensemble flou de X, tel que :

$\forall x = (x_1, x_2, \ldots, x_r) \in X$, $\mu_A(x) = \text{Min}(\mu_{A1}(x), \mu_{A2}(x), \ldots, \mu_{Ar}(x))$.

- *Le complément :* le complément A^c d'un sous ensemble flou A est : $\forall x \in X$, $\mu_{A^c}(x) = 1 - \mu_A(x)$.

Dans l'exemple précédent de la figure 1.1, le complément de l'ensemble flou est défini par la figure suivante :

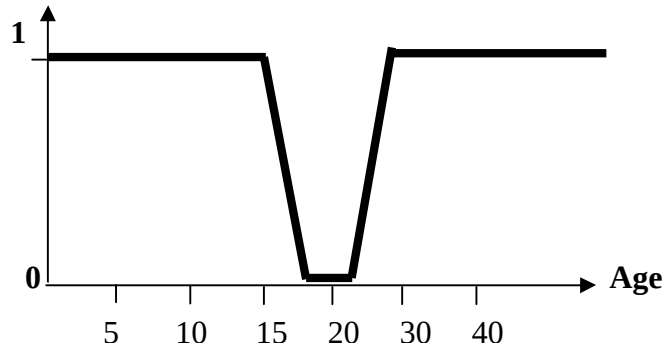


Figure 1.5 Fonction d'appartenance de Complément

- *La différence :* $\mu_{A-B}(x) = \mu_{A \cap B^c}(x) = \text{Min}(\mu_A(x), \mu_{B^c}(x))$.

- *Coupe de niveau α ou α_ coupe :* Une α-coupe est un ensemble strict composé des éléments dont le degré d'appartenance est supérieur ou égal à un seuil $\alpha \in]0, 1]$:

$A_\alpha = \{ x ; \mu_A(x) \geq \alpha\}$ (Voir la figure 1.6).

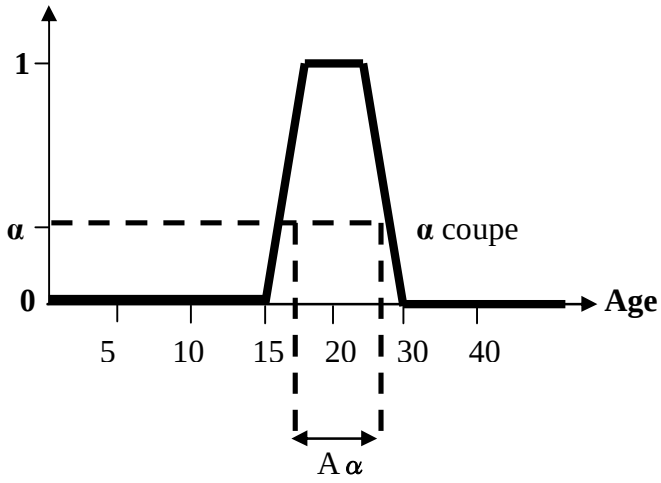


Figure 1.6 Coupe de niveau α d'un ensemble flou

La coupe de niveau α permet de définir une autre caractéristique de l'ensemble flou E appelée « *la cardinalité floue de E* » notée |E|, tel que, $\forall\ n \in N$, $\mu_{|A|}(n) = \text{Sup}\ \{\alpha / |A_\alpha| = n\}$

Pour expliquer cette définition, nous pouvons donner l'exemple suivant :

$A = \{1/u_1, 0.9/u_2, 0.9/u_3, 0.5/u_4, 0.3/u_5\}$

Les α-coupes de A sont :

$A_1 = \{u_1\}$

$A_{0.9} = \{u_1, u_2, u_3\}$

$A_{0.5} = \{u_1, u_2, u_3, u_4\}$

$A_{0.3} = \{u_1, u_2, u_3, u_4, u_5\}$

Donc $|A| = \{1/1 + 0.9/3 + 0.5/4 + 0.3/5\}$. Cette écriture signifie que la cardinalité de A vaut 1 avec un degré 1 et elle vaut 3 avec un degré 0.9,…etc.

Après avoir présenté les concepts de base de la théorie des sous ensembles flous, nous décrivons dans ce qui suit son utilisation dans l'interrogation de base de données. Dans ce cas, on parle de requêtes flexible.

3.2. Définition d'une requête flexible

Un des objectifs de la recherche dans le domaine des bases de données est d'améliorer la capacité d'expression des langages de requêtes. Un des moyens étudiés est la prise en compte

de préférences afin de faciliter l'accès à des informations pertinentes. Les requêtes à préférences sont déterminées par une relation de préférence définit comme suit :

▪ **Relation de préférence :**

Soit une relation r de schéma R (A_1, .., A_k) tel que U_i, $1 \leq i \leq k$, est le domaine de l'attribut A_i, une relation $\succ$ est une relation de préférence sur r si elle constitue un sous-ensemble de produit cartésien (U_1x…xU_k) x (U_1x…xU_k) [13].

Intuitivement, $\succ$ est une relation binaire entre les tuples de la même relation d'une BD.

On dit que le n-uplet t_1 domine le (ou est préférable au) n-uplet t_2 dans le contexte de la relation $\succ$ si $t_1 \succ t_2$.

Si ni $t_1 \succ t_2$ et ni $t_2 \succ t_1$ ne sont vérifiées, on dit que t_1 et t_2 sont incomparables : $t_1 \sim t_2$.

Propriétés:

i) Irréfléxibilité : $\forall$ x ; x $\not\succ$ x

ii) Asymétrie : $\forall$ x, y ; x $\succ$ y $\Rightarrow$ y $\not\succ$ x

iii) Transitivité : $\forall$ x, y, z ; (x $\succ$ y $\wedge$ y $\succ$ z) $\Rightarrow$ x $\succ$ z

iv) Transitivité Négative : $\forall$ x, y, z ; (x $\not\succ$ y $\wedge$ y $\not\succ$ z) $\Rightarrow$ x $\not\succ$ z

v) Connectivité : $\forall$ x, y ; x $\succ$ y $\vee$ y $\succ$ x $\vee$ x=y

La relation $\succ$ est :

- Un ordre partial strict (OPS) si elle est (i) + (iii) (ainsi aussi (ii))
- Un ordre faible (OF) si elle est (iv) + OPS
- Un ordre total (OT) si elle est (v) + OPS

Remarque : Dans la plupart des applications, la relation $\succ$ doit satisfaire au moins les propriétés d'un OPS ; du point de vue utilisateur, OPS a plusieurs propriétés importantes et leur préservation est souhaitable.

▪ **Requête flexible :**

Plusieurs travaux ont été proposés dans la littérature pour introduire la flexibilité dans l'interrogation des BD. La majorité de ces travaux utilisent le formalisme des ensembles flous pour modéliser les préférences et les termes linguistiques ainsi pour évaluer des prédicats comportant de tels termes [10].

Les prédicats dits graduels (ou flous) (i.e. dont le résultat est un degré de satisfaction), comme « jeune » et « bien-payé », sont décrits au moyen d'ensembles flous. Ces critères flous (prédicats flous) peuvent être combinés grâce à des opérateurs de conjonction ou de disjonction ou de moyennes exprimant des effets de compensation entre critères. Un prédicat flou peut également comparer deux attributs en utilisant non seulement les opérateurs usuels (égalité, supériorité, etc.), mais aussi des opérateurs graduels tels que « plus ou moins égal » ou « nettement supérieur à ». Il est possible de modifier (en affaiblissant ou en renforçant) le sens d'un prédicat donné en utilisant un modificateur qui est généralement associé à un adverbe (tel que « très », « plus ou moins », « relativement », « vraiment »). Par exemple, « très cher » est plus restrictif que «cher» et « assez haut » est moins exigeant que « haut » [13].

Les résultats d'une requête flexible sont alors qualifiés en fonction de leur adéquation aux critères de sélection et peuvent être ordonnés selon le degré de pertinence qui indique à quel point la condition est satisfaite. [12]

Pour illustrer la notion d'une requête flexible, considérons un utilisateur qui consulte, via Internet, une BD d'offres de location de biens immobiliers. L'utilisateur souhaite trouver un appartement de préférence dans le 16ème ou le 15ème Arrondissement de Paris ayant une surface « moyenne » et un loyer « modéré » avec une place de parking si possible. Peu importe le mode de formulation de la requête, l'utilisateur souhaite que la qualification soumise à l'évaluation soit la suivante [11] :

```
SELECT * FROM Annonce WHERE Ville = 'Paris'
AND Arrondissement IN (15, 16) AND Loyer 'modéré'
AND Surface 'moyenne' AND Parking = 'oui'
```

Dans cet exemple nous voyons que la requête est modélisée par des termes linguistiques "modéré" et "moyenne" (plutôt que des valeurs absolues) en utilisant le formalisme des ensembles flous qui permet d'évaluer des prédicats graduels. Dans ce cadre, les préférences peuvent être exprimées à deux niveaux distincts : à l'intérieur des conditions élémentaires et aussi dans l'agrégation de ces conditions. Dans le premier cas, l'objectif est d'exprimer que certaines valeurs sont plus adéquates que d'autres, et dans le second cas, les préférences traduisent des niveaux d'importance associés aux conditions élémentaires.

3.3. Langage de requêtes flexibles :

Dans cette sous section, nous présentons brièvement les principales propositions pour exprimer des préférences définies par des ensembles flous d'une base de données classique contenant des données précises. Dans ce cadre nous distinguons trois langages de requêtes de type SQL : **SQLF** (Structured Query Language Fuzzy), **SQLF Étendu** et **FQUERY**.

Nous découvrons d'abord l'algèbre relationnelle étendue basée sur la théorie des ensembles flous pour exprimer les requêtes utilisateurs.

▪ ***Algèbre relationnelle étendue:*** L'algèbre relationnelle est une collection d'opérations permettant d'opérer sur les concepts du modèle relationnel. Elle permet par exemple de sélectionner certains enregistrements d'une relation satisfaisant une condition ou encore de regrouper des enregistrements de relations différentes. Le résultat de toute opération de l'algèbre est une nouvelle relation. Cette propriété implique notamment qu'il n'y a pas de doublons dans le résultat et permet l'écriture d'expressions de calcul. Etant donnée, que le modèle relationnel est basé sur la théorie des ensembles, l'algèbre relationnelle utilise les opérateurs classiques de manipulation des ensembles (union, intersection, différence et produit cartésien) et introduit des opérateurs propres aux bases de données (sélection, projection, jointure, division) [13].

La notion de l'algèbre relationnelle étendue est une généralisation de l'algèbre relationnelle classique (à base de la théorie des ensembles classiques) en utilisant les techniques de la théorie des sous ensembles flous afin de manipuler les requêtes flexibles dans l'interrogation de BD. Le résultat d'une requête flexible est une relation floue pour laquelle chaque tuple t a un degré d'appartenance μ appartenant à [0,1] et décrivant le succès de la requête et on écrit μ/t.

Les opérateurs de l'algèbre relationnelle étendue appliqués sur les requêtes flexibles sont :

- Union: $\mu_{r\cup s}(u) = S(\mu_r(u), \mu_s(u))$

- Intersection: $\mu_{r\cap s}(u) = t(\mu_r(u), \mu_s(u))$

- Produit cartésien: $\mu_{rxs}(uv) = t(\mu_r(u), \mu_s(v))$

- Division: $\mu_{div(r, s, A/B)}(x) = t_s(\mu_s(a) \Rightarrow \mu_r(a, x))$.

- Différence: $\mu_{r-s}(u) = t(\mu_r(u), 1 - \mu_s(u))$

Exemple :

Soit r (a_i, b_i) et s(a_i, b_i) où i=1,2 deux requêtes flexibles avec deux prédicats graduels a_i et b_i:

	1/ <a1, b1>		0.7/<a1, b1>		0.3/<a1, b1>
r	0.6/<a1, b2>	s	0.1/<a1, b2>	r - s	0.6/<a1, b2>
	0.8/<a2, b2>		0.1/<a2, b2>		0.8/<a2, b2>

- Sélection : $\mu_{select(r, cond)}(u) = t(\mu_r(u), \mu_{cond}(u))$ avec cond : la condition de sélection.

- Projection : $\mu_{project(r, Y)}(y) = S_r\, t(\mu_r(u), \mu_{=}(u.Y, y))$

Exemple:

	0.6/<a1, b1>		0.8/<a1>
r	0.8/<a1, b2>	proj(r, A)	0.4/<a4>
	0.4/<a4, b1>		

▪ ***SQLF* (Structured Query Language Fuzzy) :** c'est une extension floue du langage d'interrogation SQL, permettant l'expression de conditions graduelles. Chaque attribut d'un tuple est associé à un degré d'appartenance μ dans [0, 1]. La signification des degrés est identique et ils peuvent être comparés les uns avec les autres, ce qui implique que les critères sont commensurables [14].

Une requête en langage SQLf a la syntaxe suivante :

```
SELECT [distinct][n|t|n,t] <attributes>
FROM <crisp relation>
WHERE <fuzzy condition>
```

Sachant que :

- <Fuzzy condition> présente des préférences exprimées par un ensemble flou commensurable de prédicats.

- Les paramètres n et t du bloc SELECT limitent le nombre de réponses en utilisant une condition quantitative (les meilleurs réponses) ou une condition qualitative (les données qui satisfont la requête selon un niveau plus haut que t).

- Le mot-clé "`distinct`" permet, comme en SQL classique, d'éliminer les doublons du résultat. Avec ce mot-clé, c'est le tuple de plus fort degré d'appartenance qui est retourné (il est le plus représentatif de la requête).

La puissance de SQLF réside dans la variété des outils offerts aux programmeurs pour construire des critères graduels. En effet, une requête flexible est composée de prédicats flous construits à partir de trois méthodes [14]:

1) La première méthode est basée sur le test d'égalité entre une valeur et un terme du langage naturel. Ainsi, l'expression "âge= jeune" est un prédicat flou qui retourne le degré d'appartenance de l'âge dans l'ensemble flou "Jeune".

2) La deuxième méthode est basée sur l'utilisation d'opérateurs de comparaisons graduels. Par exemple l'opérateur "Environ" compare la valeur d'un attribut à une valeur fixe, ex : Environ (Salaire, 10000F) ou la valeur de deux attributs, ex : Environ (R.Salaire, S.Salaire).

3) La troisième méthode est basée sur l'utilisation de modificateurs linguistiques. Ces opérateurs construisent des ensembles flous à partir d'ensembles flous existants. Par exemple le modificateur "très" permet d'accentuer une contrainte du type : "âge= très (jeune)"

Exemple :

Soit une BD possédant, entre autres, une table VOITURE qui comporte les caractéristiques suivantes: la marque, la couleur et le prix d'une voiture. On souhaite obtenir les voitures de marque célèbre et pas chères en appliquant la première méthode de construction des prédicats flous. La requête floue écrit par SQLF est la suivante :

```
SELECT * FROM VOITURE
WHERE Marque = 'célèbre' AND Prix = 'pas_chère';
```

Le traitement de cette requête est réalisé d'abord sur la première condition 'voitures de marque célèbre' et on obtient pour celle-ci les tuples réponses avec leurs degrés d'appartenance puis la réalisation de même traitement sur la deuxième condition 'voitures pas chères' (voir le tableau 1.1).

Marque	*Couleur*	*Prix*
BMW	vert	30000
AUDI	rouge	15100
VW	blanc	15000
Fiat	Noir	9000

⇒

Marque	*Couleur*	*Prix*	*Degré* $_{Marque}$	*Degré* $_{Prix}$
BMW	vert	30000	0.9	0.2
AUDI	rouge	15100	0.7	0.7
VW	blanc	15000	0.5	0.9
Fiat	Noir	9000	0.3	0.9

Tableau 1.1 Traitement d'une requête floue en SQLF

Le degré d'appartenance d'un tuple de cette requête, aura pour valeur le minimum des degrés obtenus pour ses deux conditions et on obtient, comme degré d'appartenance de ces quatre réponses respectivement: 0.2, 0.7, 0.5 et 0.3.

▪ ***SQLF Étendu*** **:** [14] 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. Dans ce cadre l'apparition de SQLF Étendu comme une extension du SQLF permet de présenter des préférences de façon bipolaire c'est-à-dire, que l'on distingue des

préférences obligatoires et des préférences optionnelles, cette extension utilise la même syntaxe que SQLF avec l'ajout d'une nouvelle clause «THEN» aux préférences optionnelles exprimées dans la requête comme suit :

```
SELECT [distinct] [n|t|n,t] <attributes>
FROM <crisp relations>
WHERE <fuzzy condition>
THEN <optional preferences>
```

La clause «THEN » peut impliquer des prédicats booléens et flous en parallèle pour exprimer des préférences optionnelles combinées par plusieurs connecteurs et une réponse à une telle requête doit satisfaire toutes les préférences obligatoires et idéalement, satisfaire au mieux les préférences optionnelles.

Exemple :

Prenons la même relation Voiture avec la requête suivante : Trouver de préférence parmi les voitures pas chères, celles qui sont de marque célèbre. Le prédicat "pas_chère" permet d'exprimer une préférence obligatoire défini par des ensembles flous. Bien que le prédicat "célèbre" permet de décrire une préférence optionnelle.

La requête exprimée par SQLF Étendu est la suivante :

```
SELECT *
FROM Voiture
WHERE Prix = 'pas_chère'
THEN Marque = 'célèbre'
```

Le tableau suivant représente le traitement de cette requête en SQLF Étendu:

Tuple	***Marque***	***Couleur***	***Prix***	μ_{Prix}	μ_{Marque}
T_1	BMW	vert	30000	0.2	1
T_2	AUDI	rouge	15100	0.7	0.8
T_3	VW	blanc	15000	0.8	0.8
T_4	Fiat	Noir	9000	0.9	0.3

Tableau 1.2 Traitement d'une requête floue en SQLF Étendu

Le résultat est évalué sur deux étapes (voir le tableau 1.2): la première, consiste à sélectionner les tuples satisfaisant la préférence obligatoire "pas_ chère" et on obtient:

T_4 (0,9)> T_3 (0,8)> T_2 (0,7)> T_1 (0,2) ce qui signifie que le tuple T_4 est préférable à T_3 et la dernière est préférable à T_2 et ainsi de suite. La deuxième étape, consiste à sélectionner parmi les tuples satisfaisant la préférence obligatoire ceux qui répondent à des préférences optionnelles. Le résultat final est le suivant: le tuple T_4 ne satisfait pas la préférence optionnelle "célèbre", donc il n'est pas sélectionné, alors que les tuples T_3, T_2, T_1 satisfont la préférence optionnelle. En effet, les résultats de cette requête sont : $0.8/T_3$, $0.7/T_2$, $0.2/T_1$.

▪ ***FQUERY :*** Le principe de FQUERY [42] est de permettre d'effectuer des requêtes flexibles c'est-à-dire des requêtes exprimées en SQL, dont la partie condition est étendue pour traiter les prédicats flous définis. Le résultat d'une requête floue est une relation floue pour laquelle chaque tuple a un degré d'appartenance décrivant le succès de la requête.

Exemple :

On considère la table SQL classique "Personnes" qui stocke des informations sur des personnes: le Nom, le Prénom et l'Age. On souhaite obtenir les personnes vieilles. La requête floue écrit par FQUERY est la suivante :

```
SELECT * FROM Personnes WHERE estVieux (Age);
```

Nom	***Prénom***	***Age***
Durand	Jean	48
Dupond	Pierre	21
Dubois	Paul	16
Thomas	Jacques	55
Jagger	Tutu	42

$\Rightarrow$

Nom	***Prénom***	***Age***	***estVieux(Age)***
Durand	Jean	48	0.8
Thomas	Jacques	55	1
Jagger	Tutu	42	0.2

Tableau 1.3 Traitement d'une requête floue en FQUERY

Au-delà de la définition des langages de requêtes flexibles, se pose 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 flexibles, 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 : [15]

- La traduction d'une requête flexible en une requête booléenne (dérivation), étant donné un niveau de satisfaction (seuil qualitatif) à atteindre.

- La compilation de la requête flexible, cette dernière stratégie vise à obtenir de façon automatique une procédure d'évaluation incluant des heuristiques.

4. PROBLÈME DES REQUÊTES FLEXIBLES À RÉPONSE VIDE

Les systèmes d'interrogation actuels présentent des interfaces conviviales facilitant l'interrogation, et surtout de fonctionnalités avancées en cas, par exemple, où les requêtes utilisateurs conduisent à des résultats insatisfaisants. Le plus souvent, les utilisateurs effectuent leur interrogation à l'aide de requêtes booléennes. Dans ce cadre, un élément de la base de données correspond pleinement ou pas du tout à la requête exprimée avec comme possible conséquence de n'obtenir aucune réponse ou au contraire une avalanche de réponses difficile à examiner et exploiter [6].

Il important de noter que, même si de telles requêtes flexibles sont plus souples que les requêtes booléennes, elles préviennent les deux problèmes énoncés précédemment. En effet, il peut très bien arriver qu'aucun élément de la base ne satisfait un tant soit peu les conditions flexibles (problème de réponse vide/ PRV) ou qu'au contraire beaucoup d'éléments soient totalement satisfaisants (problème de réponse pléthorique/PRP) [16].

Dans le cadre de notre travail, nous nous intéressons au problème de réponse vide. Dans la suite, on va définir en détails ce problème ainsi que les causes de son apparition dans les requêtes flexibles.

4.1. Présentation

Un des problèmes auxquels les utilisateurs sont souvent confrontés, dans un processus d'interrogation de BD, est le problème des réponses vides (PRV) dans lequel il n'existe aucun item de la base de données interrogée qui satisfait un tant soit peu la requête de l'utilisateur [6].

Selon P. Godfrey [17] : « *une requête échouée si à chaque fois que son évaluation produit un ensemble vide de réponse. Le problème des réponses vides signifie que l'ensemble des réponses ne permet pas d'informer l'utilisateur sur leur questions et de satisfaire leur besoins* ».

Motro [11] définit le problème de réponses vides comme suit : «*une requête est dite à réponse vide si elle ne produit aucun résultat*»

Les requêtes flexibles, limitent dans une large mesure le risque d'obtention d'une réponse vide. Cependant, ce risque n'est pas totalement éliminé en raison de plusieurs causes que l'on verra dans la sous section suivante.

4.2. Causes d'une réponse vide

Parmi les causes de présence du problème des requêtes à réponses vides, on peut citer [6] [17][12][16] :

- L'utilisateur ne connaît pas tous les détails sur le schéma, sur les données de la BD.
- Mauvaise formulation de la requête utilisateur.
- Mauvaise sémantique des prédicats flous.
- L'utilisateur effectue des conditions très précises et restrictives.
- Plusieurs conditions attachées à la requête, quelque peu contradictoires.
- Manque de données dans BD ce qui rend la demande de l'utilisateur difficile à réaliser.

Après avoir présenté les causes les plus connues au problème de réponses vides, plusieurs approches coopératives ont été proposées pour résoudre ce problème que l'on présentera dans le chapitre suivant.

5. CONCLUSION

Les outils fournis par la logique floue permettent une modélisation des phénomènes pouvons dans un certain sens s'approcher du raisonnement humain. La fait de transcender le principe de « Tout ou Rien » des ordinateurs introduit une souplesse faisant la puissance des outils flous dans de nombreux domaines informatiques, à savoir : la reconnaissance de motifs sur des images, l'élaboration de robots intelligents, entrepôts de données, Data mining flou….etc.

Dans ce contexte l'utilisation de la théorie des sous ensembles flous, semble bien adapté pour exprimer les préférences des utilisateurs dans l'interrogation de base de données, ce qui donne la notion des requêtes flexibles qui prennent en compte ces préférences afin de mieux personnaliser les réponses désirées.

L'introduction du flou au sein des requêtes adressées à des bases de données traditionnelles a enrichi le pouvoir d'expression des langages de requêtes. Cependant, certains

problèmes peuvent encore exister dans le cadre de l'interrogation floue. L'un des problèmes les plus connus est le problème des réponses vides et la façon de les éviter.

Dans le chapitre suivant nous présentons les différentes approches qui traitent ce problème de requêtes à réponses vides avec une évaluation de ces approches.

Chapitre 02

Approches pour le Traitement des Réponses Vides

1. INTRODUCTION

Le problème des requêtes à réponses vides est l'un des problèmes auxquels les utilisateurs sont souvent confrontés lors du processus de l'interrogation de bases de données. Pour pallier ce problème, plusieurs approches ont été proposées dans la littérature et ce, dans le cadre des requêtes booléennes. La plupart de ces approches sont basées sur le mécanisme de relaxation, qui consiste à remplacer une condition par une de ses généralisation qui est donc moins contraignante "Relaxation par généralisation" ou en éliminant certaines conditions de la requête "Relaxation par élimination" [6] [16].

Dans le cadre flexible, on distingue deux familles d'approches coopératives, la première est fondée sur le mécanisme de relaxation et la seconde basée sur le principe d'exploitation des requêtes précédemment exécutées par le système et dont les réponses sont non vides.

L'objectif de ce chapitre alors, est d'étudier dans un premier temps ces deux familles d'approches coopératives et dans un second temps ; nous nous efforçons de faire une évaluation de ces approches.

2. CATÉGORIES D'APPROCHES

Le problème des requêtes flexibles à réponse vide est devenu un thème de recherche de grand intérêt et plusieurs approches ont été proposées pour remédier ce problème, que l'on peut les classifier en deux grandes familles : les approches guidées par la requête et les approches guidées par un ensemble des requêtes précédemment exécutées par le système et dont les réponses sont non vides « Workload of past queries ».

Dans le cadre de notre travail, c'est à cette seconde famille qu'on s'intéresse, pour cela on commence d'abord à citer brièvement quelques travaux de recherche de la première famille d'approches puis on fait une synthèse sur les travaux existants de la seconde famille.

2.1. Approches guidées par la requête

La problématique de traitement flexible des requêtes à réponses vides a fait l'objet de plusieurs travaux qui sont généralement axés sur la relaxation des conditions de la requête afin d'élargir le volume du résultat.

Cette première famille d'approches permettant de relâcher les conditions impliquées dans la requête qui a échoué, l'idée essentielle dans ces approches consiste à étendre SQL et à

ajouter une couche supplémentaire au-dessus d'un SGBDR pour évaluer les prédicats flous sur le résultat obtenu [12]. On peut distinguer deux ensembles de travaux sur la relaxation : le premier ensemble consiste à modifier la requête initiale et le second ensemble n'opère aucune modification sur la requête initiale.

2.1.1. Approches orientées modification

Cet ensemble des approches consiste à modifier la requête initiale soit par la modification des conditions de la requête de l'utilisateur [16], soit par l'ajout de nouvelles conditions, ou l'appel à une opération de résumé ou bien par un calcul de distributions de données de la base interrogée.

Depuis les années 80, plusieurs approches ont été proposées pour la relaxation de la requête. Le but principal de ces approches est de modifier une requête à réponse vide en relâchant ses conditions afin de retourner des réponses non vides, tout restant proches sémantiquement de la requête initiale.

En 1992, Gaasreland et al [18] introduisent une approche basée sur le principe de relaxation où l'ensemble des requêtes modifiées associées à la requête à réponse vide peut être organisé sous forme d'une structure de treillis. Cette structure utilise le principe des sous-requêtes à réponses vides minimales qui seront dénotées par l'acronyme MFSs (Minimal Failing Subqueries) et le principe de MGQs (Maximally Generalized failing Queries) pour les requêtes généralisées maximales.

Chu et Chen [19] proposent une approche pour les réponses coopératives de requête, fondée sur la représentation de la connaissance en différents niveaux d'abstraction et l'accomplissement des transformations de requête entre ces niveaux. Cela est réalisé par deux processus fondamentaux : le processus *de l'abstraction de la requête* et le processus *de raffinement de la requête*. le principe consiste à convertir des types d'objet, des noms d'attribut et des valeurs de domaine entre différents niveaux de la connaissance.

Muslea [20] propose un algorithme appelé "LOQR", qui consiste à relaxer les requêtes à réponses vides (RV) sous la forme normale disjonctive, LOQR découvre les relations implicites qui existent parmi les divers attributs du domaine et utilise cette connaissance pour relâcher les contraintes de requête RV. Dans la première étape, LOQR choisi aléatoirement un sous ensemble de la BD pour tester si les valeurs des attributs satisfaites les contraintes de

la requête ou non, cette opération est effectuée en ligne pour chaque requête RV. Dans la deuxième étape LOQR utilise les techniques de "plus proche voisin" pour trouver la règle la plus semblable à la requête RV.

Toutes les approches précédentes s'appliquent à des requêtes booléennes, dans le cadre des requêtes flexibles, Bosc et al [21] proposent une approche, basée sur un mécanisme de relaxation utilisant une relation de tolérance modélisée par une relation de proximité paramétrée.
Le principe de cette approche est d'appliquer une transformation de relaxation, notée $T^{\uparrow}$ sur le prédicat graduel de la requête initiale (requête atomique) effectuée au moyen d'une relation de proximité [6]. Le processus de relaxation est réalisé d'une manière incrémentale afin d'identifier des requêtes retournant un ensemble non vide de réponses. Dans le cas où la requête Q est de type conjonctive (une conjonction des prédicats flous $P_1 \wedge \ldots \wedge P_k$), la modification est réalisée à l'aide d'une modification globale ou une modification locale : la première consiste à appliquer uniformément une transformation T_j à chaque prédicat P_j de Q et la deuxième modification affecte seulement certains prédicats composant la requête.

2.1.2. Approches sans modification de la requête initiale

En 2003, Ounelli et al [12] proposent une approche pour le traitement des requêtes flexibles à réponses vides. Cette approche ne nécessite pas de modifier la requête écrite en SQL mais plutôt de relaxer la requête floue en exploitant une base de connaissances (BC) afin d'éviter les réponses vides et de fournir la réponse la plus proche possible des souhaits de l'utilisateur. La BC est maintenue par un expert du domaine en collaboration avec l'administrateur de la BD. La BC stocke des connaissances sur les attributs relaxables telles que le sens de la relaxation, les termes linguistiques possibles, les quatre valeurs de la fonction trapézoïdale (la fonction d'appartenance) à utiliser pour chaque attribut relaxable. Toutes ces informations sont exploitées par un algorithme de relaxation pour générer une requête flexible à partir de la requête booléenne initiale.

La requête flexible est ensuite transformée en une requête classique puis soumise à l'évaluation par un SGBD. Ceci permet de tirer profit des fonctionnalités d'optimisation du SGBD.

2.2. Approches guidées par un "Workload of past queries"

Dans les approches guidées par un "Workload of past queries", plusieurs travaux ont été développés, qui sont généralement basés sur l'exploitation d'un ensemble de requêtes nommé "workload" précédemment exécutées par le système et dont les réponses sont non vides. En appliquant une mesure de similarité entre requêtes, afin d'obtenir celle qui est la plus proche sémantiquement parlant à la requête à réponse vide, qui reçoit comme réponses approximatives les réponses de cette requête du workload.

Nous présentons dans cette section deux catégories d'approches dont le but est d'éviter les résultats vides ; la première concerne les approches fondées sur le calcul de similarité à base de distance et la deuxième catégorie utilise d'autres mesures de similarité à savoir ; les connaissances, les synopses de type échantillons, histogrammes et ondelettes,…etc.

2.2.1. Utilisation d'une distance entre requêtes

En 1995, V. Raghavan et al [22] présentent une approche basée sur la réutilisabilité des requêtes optimales et précédemment exécutées par le système dans le cadre de la recherche d'information. Son principe consiste à développer une fonction pour mesurer la distance entre ces requêtes et la requête échouée.

Les requêtes optimales sont mesurées à l'aide d'une relation de préférence notée $\succeq$ entre les réponses de ces requêtes sur un ensemble de documents Δ. Si le document D est préférable à D' tel que : D, D'$\in \Delta$ et noté D $\succeq$ D' alors, la requête optimale discrimine les documents les plus privilégiées de ceux qui sont moins privilégiées et donc le document D est utilisé comme résultat de la requête optimale Q_{opt} au lieu de D' et on écrit :
$D \succeq D' \Rightarrow \alpha (Q_{opt}, D) > \alpha (Q_{opt}, D')$, sachant que α est la fonction de recherche entre la requête et le document.

Pour obtenir Q_{opt} de n'importe quelle requête échouée Q, l'algorithme suivant est appliqué pour chaque b$\in$ B avec : B= {b=D-D' : D $\succeq$ D'}.....(1) et $\alpha (Q, b) > 0$ .....(2) :

1) Choisir un vecteur de requête initial Q_0 pour k=0.

2) Soit Q_k le vecteur de requête de k+1^{ième} itérations, identifier l'ensemble de vecteurs de différence de l'ensemble B définis par la formule (1) tels que :

$\Gamma(Q_k)$= {b=D-D' : D $\succeq$ D' et $\alpha(Q_K, b) \leq 0$ }. Si $\Gamma(Q_k)= \phi$ alors $Q_{opt}= Q_K$ est le vecteur de solution et l'algorithme terminé.

3) $Q_{k+1}= Q_k + \sum_{b\in\Gamma(Qk)} b$.

4) k=k+1 et aller à l'étape 2).

Cet algorithme retourne un ensemble des requêtes optimales forme ce qu'on appelle «*La région de solution* » notée C telle que :

(1)$Q\in C \Rightarrow kQ\in C$ pour tout k>0.
(2) $Q, Q' \in C \Rightarrow Q+Q'\in C$.

Dans ce travail un ensemble des réponses relatives à la requête Q est représenté par ψ. On dit que D est classé au-dessus de D' dans ψ noté par DψD', si la fonction de recherche α (Q, D) retourne une valeur supérieur à α (Q, D').

Supposons que ψ_1 et ψ_2 deux ensembles des réponses classées dans l'ensemble de documents Δ et D, $D'\in \Delta$ on a : $\delta_{\psi 1,\psi 2}(D,D')= | \delta^{\psi 1}(D,D')- \delta^{\psi 2}(D,D') |$ où :

$$\delta^{\psi}(D, D')= \begin{cases} +1 & si\ D \Psi D' \\ -1 & si\ D' \Psi D \\ 0 & si\ D\ et\ D' sont\ égaux \end{cases}$$

La fonction de distance entre ψ_1 et ψ_2 peut être définie par trois catégories de mesure suivantes :

1. La fonction de distance effectue entre requêtes appelée *query to query* définit par :
$r\text{-}d^{(1)}(\psi_1, \psi_2) = (1/|\Delta||\Delta\text{-}1|)\sum \delta_{\psi 1,\psi 2}(D,D')$……...……..(1).

2. Soit REL(Δ) (resp. NONREL(Δ)) un ensemble privilégie (resp. ensemble non privilégié) de documents Δ et E= REL(Δ) x NONREL(Δ) le produit cartésien, cette catégorie de mesure est appelée *query-to-solution-region* donnée comme suit:
$r\text{-}d^{(2)}(\psi_1, \psi_2)= (1/2*| REL(\Delta) || NONREL(\Delta)|)\sum_{D,D'\in E} \delta_{\psi 1,\psi 2}(D,D')$ ……...……(2).

3. Soit Δ_1 et Δ_2 deux ensemble de documents, la fonction de distance appelée *solution-region-to- solution-region* donnée par la formule suivante :

$r\text{-}d^{(3)}(\psi_1, \psi_2) = 1 - ((|REL(\Delta_1)| \cap |REL(\Delta_2)|) / (\min(|REL(\Delta_1)|,|REL(\Delta_2)|))$ ……..…(3).

Exemple :

Soit $I=\{0,1\}^4/\{(0,0,0,0)\}$ un ensemble de données et $\Delta=\{D_4, D_8, D_{12}, D_{13}, D_{14}, D_{15}\}$un ensemble de documents de I indique la représentation binaire, par exemple $D_4=(0,1,0,0)$ et soit ψ_1 , ψ_2 deux ensembles des réponses classées dans l'ensemble de documents Δ pour les requêtes Q_1, Q_2 respectivement sachant que :

$Q_1=(1,1,0,0)$ avec $\psi_1=\{D_{12}, (D_{13},D_{14}), (D_4,D_8, D_{15})\}$ et $Q_2=(1,1,1,0)$ avec $\psi_2=\{ D_{14}, (D_{15,} D_{12,}D_{13}), (D_4,D_8)\}$.

La similarité entre Q_1 et Q_2 obtenue par l'application de trois catégories précitées, donne les résultats suivants :

1) $r\text{-}d^{(1)}(\psi_1, \psi_2) = (1/|\Delta||\Delta\text{-}1|) (\delta_{\psi1,\psi2}(D_{12}, D_{13}) + \delta_{\psi1,\psi2}(D_{12}, D_{14}) + \delta_{\psi1,\psi2}(D_{12}, D_{15}) + \delta_{\psi1,\psi2}(D_4, D_{15}) + \delta_{\psi1,\psi2}(D_8, D_{15})) = 1/30 (2+2+2+2+2) = 0.3.$
2) $r\text{-}d^{(2)}(\psi_1, \psi_2) = (1/2*| REL(\Delta) || NONREL(\Delta)|) (\delta_{\psi1,\psi2}(D_{12}, D_{15}) + \delta_{\psi1,\psi2}(D_{13}, D_{15})) = 1/18 (2+2) = 0.22.$
3) $r\text{-}d^{(3)}(\psi_1, \psi_2) = 1 - ((|REL(\Delta_1)| \cap |REL(\Delta_2)|) / (\min(|REL(\Delta_1)|,|REL(\Delta_2)|)) = 0.5$

La mesure de similarité dans les deux premières catégories peuvent être utilisées pour répondre aux requêtes utilisateurs. Autrement dit, si la similarité entre la requête optimale et la requête utilisateur est supérieure à un seuil donné, alors les documents classés de ces mesures constitue les documents retournés pour la requête utilisateur. La dernière catégorie de mesure de similarité, qui calcul la similarité entre deux région de solution, peut être utilisée pour trouver le vecteur initial pour démarrer l'algorithme de recherche des requêtes optimales cité précédemment.

A.Motro [23] développe un système intégré dans un SGBD, appelé "***VAGUE***", permet d'exécuter les requêtes vagues directement sans modification. La mesure de distance est utilisée pour déterminer la similarité entre les valeurs de données où chaque domaine de l'attribut de la BD est équipé par une distance entre ses valeurs, Par exemple : dans une BD des restaurants, il peut y avoir de métrique pour mesurer des distances entre les cuisines, entre les échelles de prix et entre les restaurants. Pour exprimer ces requêtes, VAGUE utilise un comparateur de sélection appelé «*similaire-à*» entre deux valeurs de même domaine, par exemple la requête Q="Liste des restaurants dont la cuisine est française et dont

l'emplacement est Downtown" peut être exprimé par VAGUE comme suit : "Liste des restaurants dont la cuisine est *similaire-à* française et dont l'emplacement est *similaire-à* Downtown".

Le principe de base de ce système est de convertir les données en vecteurs afin de calculer la similarité entre eux, par l'utilisation des données métriques et l'opérateur «*similaire-à*».

Il y a deux cas de l'utilisation de distance dans VAGUE :

i) Dans la métrique référentielle, où la distance entre deux tuples, définit par la distance entre leurs valeurs principales correspondantes.
ii) Dans l'espace de réponse, où la distance est entre les tuples de réponse.

Soit D domaine d'une BD, la donnée métrique pour D, est une fonction $M : D \times D \rightarrow R$, tel que : $\forall x, y \in D$:

(1)$M(x,y) \geq 0$

(2)$M(x, y) = 0$ ssi $x = y$

(3)$M(x, y) = M(y, x)$

(4)$\forall z \in D : M(x, y) \leq M(x, z) + M(z, y)$

En plus de cette définition standard de métrique, en associant deux paramètres : un *diamètre* et un *rayon*. Le diamètre est une borne supérieure sur toutes les distances parmi les valeurs du domaine, il sera utilisé pour estimer des distances inconnues. Le rayon établit la proximité standard, il est également utilisé pour mesurer des distances.

Le comparateur «*similaire-à*» noté ~ entre deux valeurs x, y de domaine D forme ce qu'on appel « *le qualificateur vague* » avec M et r la métrique et le rayon respectivement, on a :
$x \sim y$ si $M(x, y) \leq r$.

Dans l'exemple précédent, la requête Q écrit par la logique de prédicats avec ses qualificateurs vagues comme suit:
$x|(\exists x)(x \in Rs) \wedge (Rs.cuisine \sim française) \wedge (Rs.emplacement \sim Downtown)$
Sachant que Rs signifie un restaurant.

Les requêtes vagues peuvent être exprimées en langage d'interrogation, tel que QUEL qui décrit l'expression précédente de l'exemple comme suit :

```
RANGE OF x IS Rs
RETRIEVE (x. Rs)
WHERE Rs.cuisine ?= française AND Rs.emplacement ?= Downtown
```

Avec ?= désigne l'opérateur ~ par le langage QUEL. VAGUE interroge les qualificateurs vagues les uns après les autres et il donne pour chacun les différentes possibilités.

Dans l'exemple précédent et avec le premier qualificateur vague (Rs.cuisine ~ française), les résultats sont :

```
Analyzing VAGUE Qualification
Rs.cuisine ?= française
Possible Interpretation
1. Restaurants with similar cuisines.
2. Restaurants with similar attributes.
3. None of the above (use exact matches only)
Please select [1-3]:-
```

Les deux premières options représentent la métrique connue M pour le domaine de Rs-cuisine et la dernière option indique la métrique DEFAUT définit par la formule suivante :

$$\text{DEFAUT}(x, y) = \begin{cases} 0 \; si \; x = y \\ 1 \; si \; x \neq y \end{cases}$$

VAGUE comporte plusieurs caractéristiques qui contribuent à sa flexibilité, par exemple : il permet la métrique multiple pour chaque domaine, avec la possibilité de l'utilisateur de choisir la similarité appropriée pour chaque requête à partir d'une interface conviviale, il permet également aux utilisateurs de juger l'importance relative des attributs de la métrique référentielle et d'exprimer leur bonne volonté dans les recherches qui impliquent plusieurs qualificateurs vague.

G. Amato et al [24] développent une approche pour produire des réponses approximatives à une requête échouée avec la probabilité que la région de requête et la région de données partagent quelques objets afin de donner des résultats.
La région de requête définit par les valeurs affectées dans chaque prédicat contenant la requête et la région de données défini les données proprement dites. Le calcul de proximité

des réponses est effectué par l'intersection entre ces deux régions et une mesure de similarité entre les objets retournés dans un espace métrique par l'application d'une distance Euclidienne et une probabilité d'estimation des résultats obtenus.

Considérant M= (D, d) un espace métrique définit par le domaine de données D et la fonction de distance d et soit B_X et B_Y deux régions sous forme de cercles telles que :
$B_X= B_X (O_X, R_X)= \{O_i \in D/ d (O_i, O_X) \leq R_X\}$ et $B_Y= B_Y(O_Y, R_Y)$ avec O_X , R_X (resp. O_Y, R_Y) le centre et le rayon du cercle B_X (resp. B_Y).

La proximité entre deux région B_X et B_Y notée X (B_X, B_Y) est la probabilité qu'un élément O choisi aléatoirement dans le même espace métrique M apparaît dans les deux régions, donnée par la formule suivante :

$$X(B_X, B_Y)= Pr (d(O, O_X) \leq R_X \wedge d(O, O_Y) \leq R_Y)$$ avec Pr désigne la probabilité.

D'autres études telle que celle réalisée par P.Ciaccia et al proposent des techniques pour obtenir une évaluation précise et efficace de proximité dans le domaine de traitement d'images.

Ullas Nambiar et al [25] proposent une approche basée sur l'exploitation d'un workload de requêtes antérieures déjà soumise à la BD où l'on tente d'identifier celles qui soient similaires à la requête RV. La réponse à cette dernière serait un ensemble de tuples ordonnés formés par l'union des réponses aux requêtes jugées similaires à la requête RV.

Cette approche est plus utilisée pour traiter les requêtes imprécises, afin de rendre le système capable de fournir des réponses, non nécessairement exactes, mais qui sont en appariement aussi proche que possible avec les contraintes de la requête qui a conduit des résultats insatisfaisants. L'objectif est de retourner plus de réponses pertinentes possibles à l'utilisateur. Ces chercheurs proposent de supporter les requêtes imprécises sans modifier la BD existante. Ils présentent une approche indépendante de domaine basée sur les techniques de recherche d'information pour estimer la distance entre requêtes.

Le principe de base de cette approche, est de mapper la requête imprécise Q_{ipr} sur une relation R à une requête précise Q_p, existante dans le workload W. Ensuite, déterminer la similarité entre Q_p et les requêtes de W. La similarité entre deux requêtes est calculée à l'aide d'une similarité particulière dite "la similarité de Jaccard", appliquée sur les ensembles de

réponses correspondant à ces deux requêtes à l'aide d'un middleware ***IQE*** (***I***mprecise ***Q***uery ***E***ngine) placé entre la BD et l'utilisateur ayant une requête imprécise (Voir la figure 2.1).

Le moteur de requêtes imprécises ***IQE***, doit identifier tous les tuples de R qui sont similaires à Q_{ipr} , et qui sont en appariement approchés avec les contraintes de la requête. Les réponses aux requêtes imprécises incluent également les tuples qui ont des valeurs similaires aux contraintes. Réponses (Q_{ipr}) ≈ tuples (Q_p) + tuples (Q'), où Q'~Q_p et Q', $Q_p \in$ W.

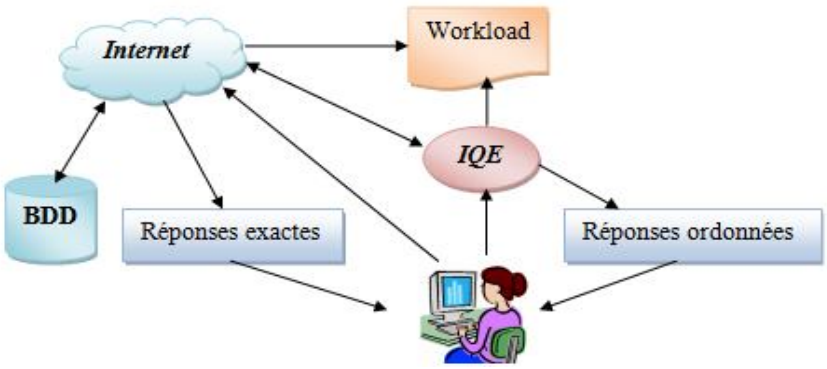


Figure 2.1 L'architecture d'IQE

La similarité entre deux requêtes est calculée comme la similarité entre leurs tuples de réponse appelés "Super tuples", où chaque super tuple comporte un sac de mots-clés (Bag) pour chaque attribut de la relation. Ce bag comporte aussi le nombre d'occurrences de mots-clés pour chaque attribut, par exemple : dans une BD de certains auteurs décrivant des publications. Le tableau suivant représente le super tuple de la requête qui cherche l'auteur "Ullman".

Auteur=Ullman	***Bag***
Co-auteur	C.Li:5, R.Motwani:7 ...
Titre	Data mining: 3, optimizing: 5 ...
Sujet	Integration: 5, learning: 2 ...
Conférence	SIGMOD:5, VLDB:5,....
Année	2000:6,1999:5,....

Tableau 2.1 Exemple d'un supertuple de la requête Author="Ullman"

Dans cet exemple, le mot clé 'SIGMOD' de l'attribut conférence a un nombre d'occurrence égale à 5 qui signifie que l'auteur Ullman dispose de 5 tuples qui lient l'attribut conférence avec la valeur 'SIGMOD'.
La mesure de similarité utilisée est la similarité de Jaccard suivante :

$$Sim_J (A, B) = \frac{|A \cap B|}{|A \cup B|}$$

En applique cette formule sur deux requêtes Q_1 et Q_2 et on trouve [48] :
$Sim_J (Q_1, Q_2) = Sim_J (ST_1, ST_2)$ où ST_1, ST_2 super tuples de Q_1, Q_2 respectivement.

2.2.2. Utilisation d'autres mesures

Il existe d'autres mesures de similarité entre requêtes et qui ne sont pas fondées sur la notion de distance telles que :

Approche à base d'Ontologies :

En 2002, Alain Bidault et al [26] proposent une approche basée sur une mesure de similarité entre la requête initiale de l'utilisateur et des requêtes prédéfinies dans le cadre du système de médiation. Cette mesure repose sur une hiérarchie de concepts, plus deux concepts partagent des propriétés en commun plus ils sont plus proches. Le but de cette approche est d'aider un utilisateur à reformuler sa requête quand il échoue, par l'application des ontologies afin de mesurer la similarité entre prédicats et donc entre requêtes de ces prédicats.

Le principe de cette approche appuyé sur une méthode de proximité entre la requête échouée et l'ensemble des requêtes prédéfinies. Le nombre de requêtes de cet ensemble est limité par une fonction f générée par l'algorithme suivant:

Soit S une source de données et q_s un ensemble des requêtes prédéfinies construit à partir d'un ensemble P_s qui contient les prédicats pour lesquels la source S donne des instances, $P_{su} \subseteq P_s$, est un ensemble inclut les prédicats de S les plus utilisés. La fonction f citée précédemment retourne le nombre moyen des requêtes prédéfinies de q_s définis comme suit :

Si P_k est l'un des prédicats de P_{su} et pour tout prédicats P_j de P_s-P_{su}, P_k est plus utilisé que P_j et : $\sum_{Pi \in Psu-\{Pk\}} Poids(P_i) < f(q_s) \leq \sum_{Pi \in Psu} Poids(P_i)$. Avec $Poids(P_i)$ indiqué le nombre de différentes réécritures de P_i de la source S.

Exemple :

Supposons qu'on a trois prédicats de la source S : Place, Hôtel et Sahara avec la fonction f telle que : $f(q_s) = 3$ et le prédicat Place est plus utilisé que les prédicats Hôtel et Sahara et P_{su}= {Place, Hôtel} avec Poids(Place)=2 et Poids(Hôtel) = Poids (Sahara) = 1, donc si on ajoute Sahara à P_{su} la somme: $\sum_{Pi \in Psu}$ Poids(P_i)= 4 et si on enlève Hôtel, la somme devient égale à 2.

L'algorithme de génération de l'ensemble des requêtes prédéfinies est effectué en 04 étapes :

1) Déterminer l'ensemble des prédicats P_{su} de S.
2) Générer des sous requêtes contenant seulement les prédicats de P_{su}.
3) Ajouter d'autres conditions aux sous requêtes obtenues dans 2) pour deviennent plus précises et en leur permettant toujours d'être récrit par S.
4) Il y'a deux situations possibles :
   - Un nombre de requêtes dépasse le quote-part de S: suppression de certaines de ses requêtes.
   - qs n'est pas atteint : ajout de prochain prédicat de P_{su} et aller à 2).

La similarité entre la requête Q_u et l'ensemble des requêtes prédéfinies Q_p est basée sur le calcul de similarité entre les prédicats contenants ces requêtes. Ainsi que cette similarité entre prédicats peut être réalisée par le calcul de similarité entre deux conjonctions des concepts qui se relient aux mêmes objets, en tenant compte : i) la différence entre le nom des variables de Q_u et ceux de Q_p, ii) le lien entre les variables.

Le calcul de similarité est déterminé par la note de similarité entre deux concepts C et C' de l'ontologies notée : $N_{C \to C'}$, en identifiant ces concepts par une chaine de caractères de taille *k* pour C=[n_1,….,n_k] et de taille *l* pour C'=[m_1,…,m_l], mβ et nα deux nombres de concept C et C' respectivement avec $\alpha \in [1..\kappa]$ et $\beta \in [1..l]$, $com_{\alpha\beta}$: le plus long préfixe commun et $end_{n\alpha}$, $end_{m\beta}$ le reste de la chaine de caractères de C et C' respectivement, tels que $n\alpha = com_{\alpha\beta} + end_{n\alpha}$ et $m\beta = com_{\alpha\beta} + end_{m\beta}$

Ph : correspond à la profondeur de la hiérarchie des concepts représentés par l'ontologie.

On a $N_{C \to C} = N_{C' \to C'} = 1$ et $N_{C \to C'} = 0$ si n'a aucune relation entre C et C' sinon la note est calculée en trois étapes suivantes :

- Calculer le rapport de similarité de mβ portée sur nα , noté "$R_{m\beta \rightarrow n\alpha}$", avec la mise à 0 si la valeur est négative : $R_{m\beta \rightarrow n\alpha} = ((|com_{\alpha\beta}| + P_h - |n_\alpha|)/P_h) - 0.002 \times |end_{m\beta}|$
- Calculer le rapport de similarité maximale de C' portée sur nα "$R_{maxC' \rightarrow n\alpha}$" tel que : $R_{maxC' \rightarrow n\alpha} = Max\{R_{m\beta \rightarrow n\alpha}, \beta\}$
- La note de similarité de C' portée sur C définis par la formule (F) suivante :

$N_{C' \rightarrow C} = Avg\ Max\{R_{m\beta \rightarrow n\alpha}, \alpha\}$ ..... (F)

Nous présentons dans ce qui suit les étapes de l'algorithme utilisé dans cette approche :

- Découvrir les requêtes Q_u et Q_p pour obtenir des conjonctions des concepts Q_{up} et Q_{uu}, en remplaçant les prédicats avec leurs types.
- Comparer les concepts de la requête prédéfinie Q_{up} à ceux de Q_{uu} ; cette comparaison est dirigée afin d'évaluer quels prédicats de Q_{uu} doivent être associés aux prédicats de Q_{up}.
- Obtenir $Q_{u'p}$ en appliquant les substitutions des deux prédicats de Q_{up} et de Q_{uu}.

L'algorithme commence par le premier prédicat de Q_{up} avec une meilleure note de similarité et il s'arrête quand tous les prédicats de Q_{up} sont traités.
La note de Q_p portée sur Q_u est la moyenne parmi les notes obtenues pour les prédicats de $Q_{u'p}$ portée sur ceux de Q_{uu} (appliquer la formule F). La première note est calculée par la comparaison entre les prédicats de même objet et la deuxième note prend en compte seulement du nom des concepts et elle est établie pour chaque prédicat de Q_{uu}.

Approche à base de connaissances

Osmar et al [27] développent une approche fondée sur la recherche des requêtes non échouées similaires à la requête échouée par l'application d'une méthode, qui estime la similarité entre ces requêtes. Le principe de cette approche est similaire à l'utilisation de fichier *log* au niveau du serveur, qui conserve touts les événements qui se produisent au sein d'un système donné et y compris la trace des requêtes. Cette structure n'est pas suffisante pour calculer la similarité entre requêtes. En effet l'introduction de nouvelle structure de donnée appelée « Mémoire des requêtes ».

La mémoire des requêtes est un ensemble de requêtes où chaque requête est modélisée comme suit:

- *Bag Terms :* un ensemble non ordonné des termes lient à la requête.

- *Count :* le nombre de fois où la requête a été rencontré dans la mémoire des requêtes.
- *LDate* : la date de la dernière fois où la requête est rencontrée dans la mémoire des requêtes.
- *FDate :* la date de la première fois où la requête est rencontrée dans la mémoire des requêtes.
- *QResults :* Liste ordonnée des réponses retournées.
- *RDate :* la date d'obtention des réponses qui n'est pas nécessairement liée à FDate, et LDate.

La description des dates LDate et FDate a pour but de classer les requêtes similaires à la requête échouée. Ce travail définis sept (07) mesures de similarité qui peuvent être utilisées séparément ou combinées, en utilisant les notations suivantes pour la description de différentes méthodes: Q représente la requête courante, Δ est la mémoire des requêtes, Q.terms et Q.Results indique BagTerms et QResults de Q respectivement, Q.Results[i] décris la i^{ème} réponse de Q.Results avec Q.Results[i].Title et Q.Results[i].text représentent l'ensemble des mots extraits du titres et l'ensemble des mots extraits de résultat de recherche respectivement.

1) Méthode naïve basée sur la requête : $\forall\ q \in \Delta$ / q.terms $\cap$ Q.terms$\neq \emptyset$, cette méthode est utilisée par certain moteur de recherche, elle est simple et facile à retrouver les requêtes proches de Q avec les termes communs entre eux. Les résultats sont classés en fonction des valeurs de *Count* et *LDate* et elles sont parfois intéressantes, par exemple : la requête "Pièces véhicule" est trouvée similaire aux "pièces automobile" et "pièces de Honda".

2) Méthode naïve simplifiée basée sur l'URL :

$\forall\ q \in \Delta$/ q.Results.URL$\cap$Q. Results.URL$\neq \emptyset$, cette méthode est similaire à la première, sauf au lieu de chercher des termes communs d'une requête, en recherchant par URL communes renvoyées par le moteur de recherche.

Le raisonnement est que si les URL renvoyées par deux requêtes ont au moins une URL en commun, alors ces requêtes pourraient être liées. Cette méthode fonctionne bien sur un ensemble limité de requêtes recueillies, mais a tendance à faire des mauvais résultats plus souvent que la méthode suivante.

3) Méthode naïve basée sur l'URL : $\forall q \in \Delta / \theta_m < \frac{q.Results.URL \cap Q.Results.URL}{Q.Results.URL} < \theta_M$

Cette méthode considère l'ensemble d'URL dans les résultats des requêtes avec θ_m est un seuil minimum, tandis que θ_M est un seuil maximum. Cette méthode pourrait donner des résultats intéressants en fonction des seuils fixés par le programmeur. Si θ_M est trop proche de 100%, les requêtes deviennent trop similaires. Si θm est trop petit, les requêtes sont complètement différentes. Dans l'implémentation prototype de cette méthode ces chercheurs ont mis θ_m à 0,2 et θ_M à 0,8. Cette méthode est plus précise et plus souple que la méthode précédente.

4) Méthode basée sur le titre des résultats : $\forall$ q$\in$ $\Delta/\exists$ i, q.Results[i].Title $\cap$ Q.terms$\neq \emptyset$ et q.Results[i] $\notin$ Q.Results. Cette méthode, recherche les requêtes qui ont des titres de leurs résultats figurant dans les termes de la requête initiale. L'idée est que ces résultats n'ont pas été récupérés par la requête initiale.

5) Méthode basée sur le contenu des résultats :

$\forall$ q$\in$ $\Delta/\exists$ i, q.Results[i].Text $\cap$ Q.terms$\neq \emptyset$ et q.Results[i] $\notin$ Q.Results. L'idée de cette méthode est similaire à la précédente, sauf que le contenu des résultats des requêtes est considéré comme des points communs au lieu des titres.

6) Méthode de titre des résultats communs :

$\forall$q$\in$ $\Delta/$ $\exists$ i, j, q.Results[i].Title $\cap$ Q. Results[j].Title $\neq \emptyset$. L'idée principale de cette méthode est que si deux requêtes, avec les titres dans ses résultats renvoyés sont similaires (il ya des mots communs), alors les requêtes sont également similaires.

7) Méthode de contenu des résultats communs :

$\forall$q$\in$ $\Delta/$ $\exists$ i, j, q.Results[i]. Text $\cap$ Q. Results[j]. Text $\neq \emptyset$. Cette méthode est similaire à la précédente sauf que les contenus sont comparés, au lieu des titres.

Les deux dernières méthodes sont particulièrement efficaces pour trouver les requêtes qui sont syntaxiquement différentes mais sémantiquement similaires. Par expériences de ces chercheurs, ils ont montré que sur un petit ensemble de requêtes la dernière méthode fonctionne mieux que la précédente, mais la sélection du contenu des pages au lieu de titres retournés par les moteurs de recherche serait prendre beaucoup de temps. Donc à l'utilisateur de spécifie quelles mesures de similarité, veut utiliser pour trouver des requêtes similaires.

Les trois approches suivantes sont proches dans le principe d'approximation des requêtes, elles ont basées sur le calcul des synopses, qui sont des résumés statistiques des données originales, et peuvent prendre trois types : échantillons, histogrammes et ondelettes.

Approche à base d'Échantillons :

S. Acharya et al [28] développent un système qui fournit rapidement des réponses approximatives aux requêtes dans un entrepôt de données, qui sont très courantes dans les applications OLAP. Ce système appelé **AQUA** (**A**pproximate **QU**ery **A**nswering), il a été conçu pour fonctionner au-dessus de tout SGBD relationnels commerciaux, il calcul des synopses de type échantillons et les stocke dans la BD, ils ont mis à jour quand la BD est modifiée.

Le principe de base de ce système est d'offrir des réponses approximatives, par la réécriture des requêtes d'utilisateur à partir des échantillons et l'exécution des nouvelles requêtes. La nouvelle forme de la requête et la réponse approximatives sont analysés afin de garantir la qualité de la réponse, et diminuer l'erreur. L'architecture de haut niveau d'AQUA est représentée dans la figure 2.2, ainsi que les étapes prises au cours du ce processus. Quand de nouvelles données arrivent, AQUA maintient les échantillons à jour, avec peu ou pas accès aux données originales.

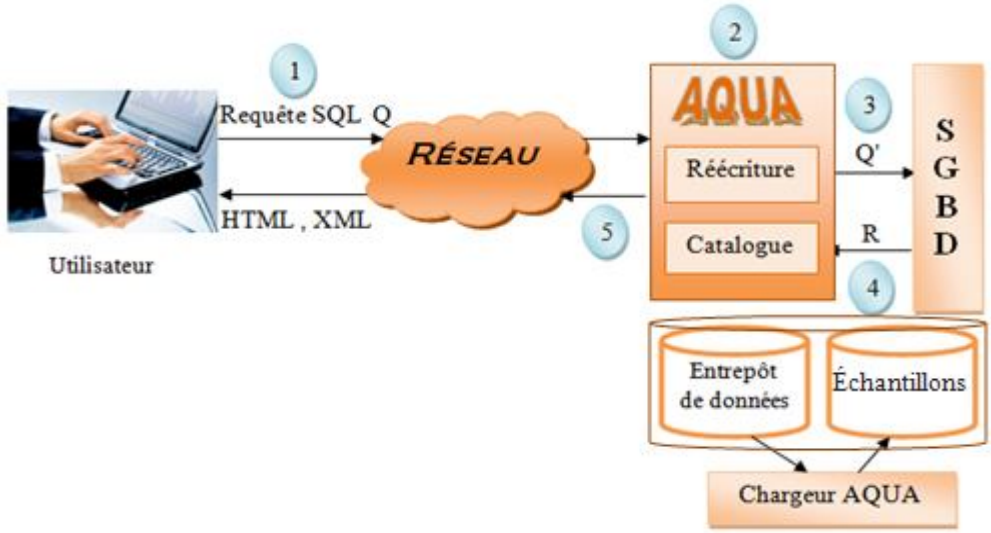


Figure 2.2 L'architecture AQUA

Le processus du système AQUA est s'effectué en 05 étapes suivantes (illustrées dans la figure) :

1) Utilisateur effectue une requête Q sur internet par exemple et attend les réponses.

2) La requête Q est mis à AQUA qui la transforme à Q' à l'aide d'échantillons appropriés à Q, et l'envoie Q' au SGBD.

3) Le SGBD reçoit la requête Q' à partir d'AQUA afin d'interroger l'entrepôt de données et retourner les réponses approximatives de Q'.

4) Le SGBD retourne les résultats R de Q' à partir de l'entrepôt de données, ceci offre des informations au chargeur AQUA pour calculer les échantillons.

5) Les résultats R qui se trouvent dans le catalogue d'AQUA sont par la suite envoyer à l'utilisateur sous forme compréhensible, par exemple une page html.

Exemple :

Soit Q une requête utilisateur qui interroge deux tables relationnelles Line et Order :

```
SELECT sum(quantity)
FROM L Line, O order
WHERE L.orderkey = O.orderkey AND O.orderstatus = F
```

Pour réécrire la requête Q, AQUA remplace les tables Line et Order par les deux tables suivantes: bs-Line et js-Order à l'aide des échantillons qui peuvent considérés comme un seul échantillon de jointure entre Line et Order, de sorte que l'opérateur sum dans la clause select est gradué par 100. La requête réécrite soumis à l'entrepôt présentée comme suit :

```
SELECT 100*sum(quantity)
FROM L Line, O order
WHERE L.orderkey = O.orderkey AND O.orderstatus = F
```

Le tableau suivant représente quelques réponses approximatives avec l'erreur retourné:

Réponses	*sum(quantity)*	*Erreur*
R1	3.778 e+06	1.24 e+05
R2	7.781 e+06	1.77 e+05
R3	3.87 e+06	1.26 e+05

Tableau 2.2 Exemple des réponses d'AQUA

Remarque : le calcul de l'erreur relatif à la requête Q est s'effectué par l'application de la formule suivante :

Erreur (Q)=$\left|\frac{Réponses\ exactes - Réponses\ approximatives}{Réponses\ exactes}\right|$

Approche à base d'Histogrammes :

Y.E. Ioannidis et al [29] proposent une nouvelle technique pour fournir des réponses approximatives à une requête, par l'utilisation d'Histogrammes qui sont stockés sous forme de relation et une mesure d'erreur de ces réponses. Un grand avantage de mettre cette technique est que presque tous les SGBD commerciaux maintiennent les histogrammes et obtient des approximations qui ne nécessitent pas de modifications importantes de ces systèmes.

Avant de présenter le processus d'approximation des réponses à une requête à base d'histogrammes, nous définissons d'abord la mesure d'erreur des réponses retournées et la notion d'histogramme.

Considérons deux ensembles $S_1 = \{u_1, u_2, \ldots, u_n\}$ et $S_2 = \{v_1, v_2, \ldots, v_m\}$ représentent respectivement ; réponses exactes et réponses approximatives. La mesure d'erreur entre ces deux ensembles est effectue en deux étapes :

i) Déterminer quel élément de la réponse approximative correspond le mieux à un élément dans la réponse exacte et vice-versa.

ii) Calculer l'erreur en utilisant la distance euclidienne *dist* entre les objets les uns aux autres. La formule développée dans cette approche pour mesurer l'erreur dans la réponse approximative notée *MAC* (*M*atch *A*nd *C*ompare), définie comme suit :

$$MAC\ (S_1,S_2) = \sum_{i \in n,\ j \in m} mult\ (u_i, v_j) \times dist\ [q](u_i, v_j)$$

Avec $mult\ (u_i, v_j) = max\ (1, max\ (d_1, d_2) - 1)$ avec d_1, d_2 des tranches dans le graphe de données de u_i, v_j respectivement et q: quantum, n'importe quel petit nombre entier, en général il est égale à 1 et $Dist\ [q]\ (u_i, v_j) = \begin{cases} dist\ (ui, vj) & si\ max(d1, d2) = 1 \\ dist\ (ui, vj) + q & sinon \end{cases}$.

Un histogramme H sur un attribut X est construit en utilisant une règle de partitionnement des données en sous-ensembles disjoints appelé «paniers » et il permet d'approximer des fréquences et des valeurs dans chaque panier. En pratique, chaque panier dans un histogramme comporte les informations suivantes: le nombre total de tuples qui se trouvent dans le panier (*tot*), les valeurs minimale et maximale de panier i (min_i, max_i) et le

nombre de valeurs distinctes (*compte*$_i$) avec l'étendue moyenne $E_i = \left|\frac{tot}{Comptei}\right|$ et la fréquence moyenne de panier *moy*=$\left|\frac{maxi-mini}{2}\right|$.

Remarque : Dans cette approche les attributs d'une relation sont supposés de type numérique.

Exemple :

Soit une BD comporte 05 valeurs avec leur fréquence f :

D= {**10**(f=100), **60**(f=120), **70**(f=10), **90**(f=80), **100**(f=2000)} et l'histogramme H correspondant définit par trois paniers de D suivants : {10}, {60, 70,90} et {100} illustrés dans le tableau suivant :

tot	***min$_i$***	***max$_i$***	***compte$_i$***	***E$_i$***	***moy***
100	*10*	*10*	*1*	*100*	-
210	*60*	*90*	*3*	*70*	*15*
2000	*100*	*100*	*1*	*2000*	-

Tableau 2.3 Exemple d'histogramme

Dans ce travail les histogrammes sont stockés sous forme des relations appelées "relation d'approximation" (*ApproxRel*) qui peuvent être calculées en utilisant la requête SQL "Expand.sql" suivante:

```
SELECT (H.min+IC.idx * H.E)
FROM H, IC, IA
WHERE IC.idx ≤ H.Compte AND IA.idx ≤ H.moy
```

Sachant que : I_C et I_A sont deux relations, chacune avec un seul attribut idx, telle que I_C consiste à générer les positions des valeurs au sein de chaque panier et utilise ensuite les valeurs minimale et moyenne pour calculer les valeurs approximatives et I_A repartit chaque valeur en fonction de sa fréquence. La répartition approximative capturée par l'histogramme H ci-dessus se présente comme suit :

Valeur approximative	10	60	70	90	100
Fréquence approximative	100	70	70	70	2000

Tableau 2.4 Exemple d'une répartition approximative

Il est également possible de combiner deux histogrammes sur différents ensembles d'attributs et obtenir un histogramme unique formé par l'union de ces deux ensembles.

Le processus d'approximation des réponses à une requête à base d'histogrammes, consiste à transformer la requête Q sur les relations R_1, R_2,…R_n en requête Q' à l'aide des histogrammes correspondants H_1,H_2,…,H_n qui génèrent les mêmes réponses approximatives quand on utilise les relations d'approximation *ApproxRel* (H_i) calculées par Expand.sql. (Voir la figure 2.3)

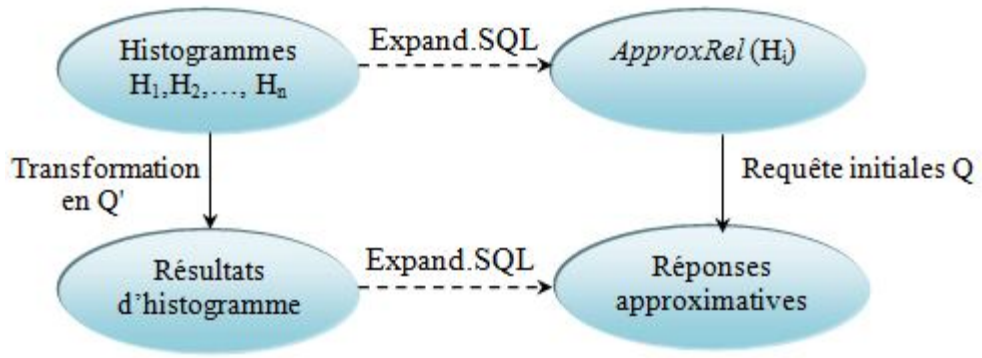


Figure 2.3 Le processus de requêtes à base d'histogrammes

Approche à base d'Ondelettes :

K.Chakrabarti et al [30] proposent une approche basée sur le principe d'approximation par ondelettes (en anglais wavelets), consiste à produire des coefficients d'ondelettes de la BD et l'utiliser pour fournir des réponses approximatives aux requêtes.

Avant de présenter le processus de construction des coefficients d'ondelettes et l'algorithme d'approximation des requêtes, nous découvrons d'abord la procédure de la décomposition des ondelettes multidimensionnelles utilisées dans cette approche.

Les ondelettes sont un outil mathématique utile pour décomposer hiérarchiquement des fonctions ou des signaux. Cette approche utilise la décomposition multidimensionnelle d'ondelettes de Haar. Ces ondelettes sont simples à concevoir, très rapide à calculer, et se

sont bien adaptées dans la pratique pour une variété d'applications telles que : traitement d'image, traitement de signal, interrogation des BD et aux approximations d'OLAP.

La décomposition multidimensionnelle des ondelettes est une généralisation de processus de décomposition unidimensionnelle par l'utilisation de la méthode non standard qui consiste à effectuer des opérations de moyenne et de différence pour chaque ligne k de la matrice A de dimension d x d avec k=1,…,d et la racine de dimension 2 x 2 située aux coordonnées [$2i_1$, $2i_2$] afin de produire la décomposition W_A. (Voir la figure 2.4)

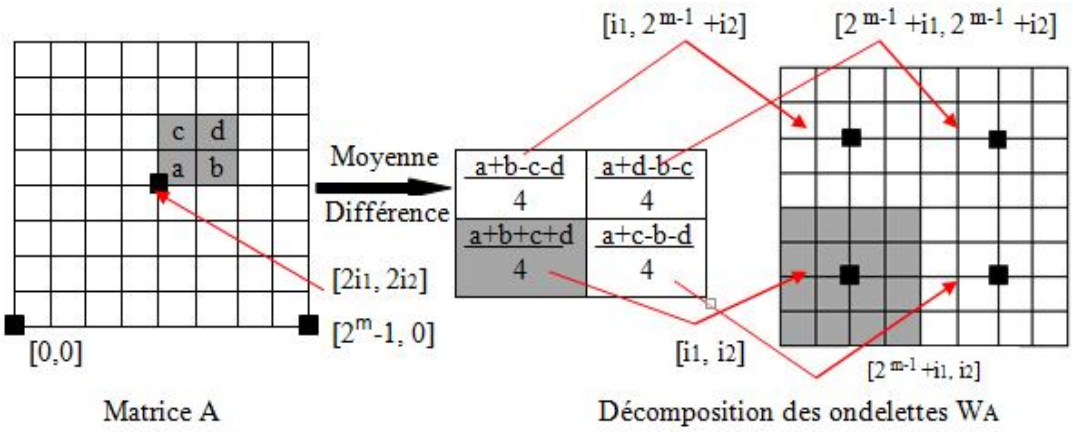


Figure 2.4 La décomposition multidimensionnelle des ondelettes

L'algorithme d'approximation des requêtes est basé sur l'application récursive de propriété principale de la décomposition non standard de Haar qui est l'indépendance entre les calcules de décomposition de matrice A et ceux pour chacun de ses sous matrices en appliquant le principe de "profondeur d'abord" pour l'exploitation de ces données.

Cet algorithme met une nouvelle représentation de l'algèbre relationnelle et les opérateurs SQL (Sélection, la projection, le jointure) qui prennent comme opérandes les coefficients d'ondelettes W_{Ri} plutôt que des tables relationnelles, ce qui simplifie le processus d'approximation des requêtes avec leurs relations R_1,…, R_n et l'application d'une relation de proximité render(W_{Ri}). Cette dernière n'est pas efficace car elle peut contenir autant des tuples que les requêtes originales, ce qui implique que les coûts d'exécution des requêtes à base des W_{Ri} est aussi élevés que ceux des requêtes originales. D'autre part, l'ondelette W_{Ri} est une représentation compressées de render (W_{Ri}) qui est généralement inférieures à R_i. L'exécution de la requête Q par les coefficients d'ondelettes rendre l'exécution classique (sans Wri) plus rapide par l'adaptation des opérateurs "op" (op peut être : SELECT, PROJECT et

JOIN) exploitants le processus d'approximation des requêtes par W_{Ri}, tout en garantissant la sémantique (voir le diagramme de transition de la figure 2.5).

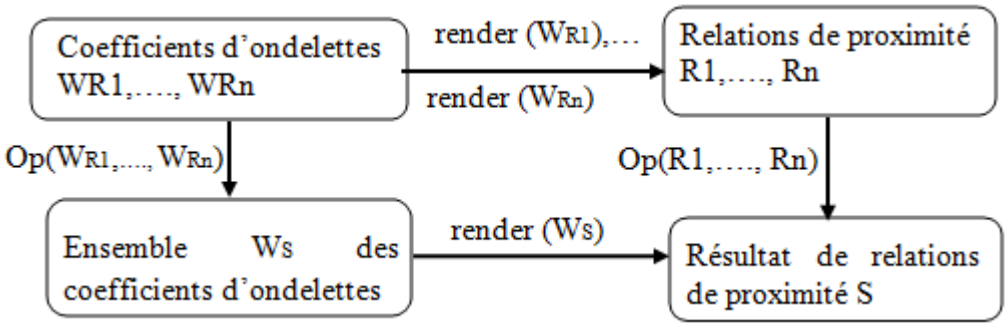


Figure 2.5 Opérateurs de processus d'approximation des requêtes

Remarque : Dans cette approche les attributs d'une relation sont supposés de type numérique.

Exemple :

L'opérateur SELECT, consiste à filtrer les coefficients d'ondelettes W_R qui ne ressemblent pas les lignes sélectionnées et donc ne pas contribuent dans le résultat. La forme générale de cet opérateur est : $\mathbf{SELECT_{pred}\ (W_R)}$ où pred représente une conjonction des prédicats sur un sous ensemble de *d* attributs de R c'est-à-dire :

pred= $(l_{i1} \leq X_{i1} \leq h_{i1}) \wedge \ldots \wedge (l_{ik} \leq X_{ik} \leq h_{ik})$ où l_{ij} (resp. h_{ij}) la borne inférieur (resp. borne supérieur) de la ligne sélectionnée de dimension D_{ij}, j=1,2,…,k et $k \leq d$ avec X_{ij} la valeur maximale de W_R. L'ensemble des requêtes de proximité sont défini dans les k lignes de la dimension $D'=\{ D_{i1}, D_{i2},\ldots, D_{ik}\}$ satisfaisant la condition suivante :

Si $l_{ij} \leq W.X_{ij} \leq h_{ij}$ ou $W.X_{ij} \leq l_{ij} \leq W.Y_{ij}$ (avec $W.X_{ij}$ (resp. $W.Y_{ij}$) la valeur maximale (resp. valeur minimale) de coefficient W) Alors :

Ajouter le coefficient W à l'ensemble de résultat de l'opération W_S donc : $W_S := W_S \cup \{W\}$.

Approche à base de substitution des requêtes

Récemment Bosc et al [31] proposent une mesure de similarité entre requêtes flexibles. Cette approche suppose que pour chaque domaine des valeurs d'un attribut (de la base de données) une relation de ressemblance est disponible. Le principe de base de cette approche est basée sur un mécanisme de substitution de la requête RV par d'autres requêtes du Workload. L'idée de cette approche est de ne pas ajouter/supprimer certains prédicats de la requête RV Q, mais plutôt de trouver une meilleur "substitution" à Q parmi les requêtes du Workload.

Soit Q définie par : Select A_1, …, A_n From R Where A in E, une requête RVet Q' une requête du Workload D^+ définie par : Select A'_1, …, A'_n From R Where A in E'
La substitution de Q par Q' donne la nouvelle requête Q'' définie comme suit :
Select A_1… A_n From R Where A in (E'-E).

Pour déterminer la requête Q' qui donne une meilleur substitution à Q, la formule suivante est appliquée :

$$SBs(E, E') = \frac{\sum_{x\prime \in sp(E\prime - E)} W(x\prime) \times Sup_{x \in E} min\,(res(x,x\prime), \psi\,(\mu E'(x'), \mu E(x)))}{\sum_{x\prime \in sp(E\prime - E)} W(x\prime)} \quad \ldots\ldots\ldots\ldots\ldots (1)$$

Sachant que : Sp(E) définit le support de E :

$\psi(a, b)= 1- |a-b|$, $W(x')=\mu_{(E'-E)}(x')= \min (\mu_{E'} (x'), 1-\mu_E (x'))$ et res(x, x') signifie le degré de ressemblance entre un élément de E'-E et un élément de E, calculé par l'indicateur de tolérance Z avec la fonction d'appartenance trapézoïdale de support [-α, α] et de noyau [-β, β] : $res(x, y)= \mu_Z(|x-y|)$.

L'algorithme de substitution de requête Q par la requête Q' de D^+ (qui partage au moins un attribut de sa clause "WHERE" avec celle de Q), définit par les cinq étapes suivantes :

1) Remplacer la clause "SELECT" de Q' par celle de Q.
2) Enlever du Q' tout prédicat qui n'apparait pas dans la clause "WHERE" de Q.
3) Remplacer chaque prédicat de Q' qui est strictement inclus dans l'un des prédicats de Q.
4) Calculer la proximité entre le prédicat de Q' et celui qui lui correspond dans Q, en utilisant la formule (1), et remplacer le prédicat de Q' par son union avec celui de Q.
5) Agréger les proximités locales par le biais d'une norme triangulaire (l'idée est d'évaluer la mesure avec laquelle chaque prédicat de la requête de substitution est proche de l'attribut correspondant de la requête échouée) afin de déterminer la plus proche substitution de Q.

3. EVALUATION DES APPROCHES

Après avoir présenté les différentes approches proposées pour traiter le problème des requêtes à réponses vides, nous nous efforçons dans cette section à faire une évaluation entre ces approches.

3.1. Evaluation des approches guidées par la requête

Comme nous avons mentionné dans la sous section 2.1, cette famille d'approches est basée sur la relaxation des conditions impliquées dans la requête qui a échoué. Nous avons distingué deux ensembles de travaux de relaxation : le premier ensemble consiste à modifier la requête initiale et le second ne nécessite pas la modification de la requête initiale.

Dans le premier ensemble d'approches, la stratégie utilisée consiste à modifier les conditions de la requête RV par les techniques de relaxation qui élargissent le spectre de la requête et permettent de retourner à l'utilisateur des réponses approximatives à sa requête initiale. Contrairement, au deuxième ensemble de cette famille d'approches, qui suit le même principe de relaxation mais sans modification de la requête RV en exploitant une base de connaissance contenant différentes connaissances sur les attributs relaxables qui jouent un rôle crucial dans la découverte de relations sémantiques entre données.

Ces approches souffrent d'un inconvénient majeur pour le traitement des requêtes RV à savoir, l'Explosion combinatoire, notamment dans la relaxation des requêtes composées.

3.2. Evaluation des approches guidées par un "Workload of past queries"

Nous avons présenté dans la sous section 2.2 plusieurs approches guidées par un Workload of past queries, et qui peuvent être répartie en deux ensembles différenciés selon la mesure de similarité utilisée entre la requête RV et l'ensemble des requêtes du Workload.

Le premier ensemble utilise une distance pour estimer les réponses à une requête RV, qui donne un meilleur résultat par rapport au second ensemble qui utilise d'autres mesures de similarité telles que les synopses de type échantillons, histogrammes, ondelettes,… etc.

Le mécanisme d'utilisation d'une distance pour mesurer la similarité entre requêtes, offre plusieurs avantages notamment dans les techniques de classification, traitement d'images, application OLAP, réparation des BD,…etc. Elle est simple à implémenter, rapide dans le

calcul et donne également de bonnes performances comparativement à d'autres mesures de similarité [23] [25].

Dans le second ensemble d'approches où plusieurs mesures de similarité ont été utilisées, telles qu'on trouve les trois travaux qui se basent sur le calcule des résumés statistiques appelés « Synopses », comportent trois types, qui ont apparu respectivement: échantillons, histogrammes et ondelettes. Ces trois mesures ont évalué par la meilleure mesure représentée par les synopses de types ondelettes par rapport aux autres types, surtout dans le contexte d'approximation des requêtes des entrepôts de données [30]. Néanmoins, ce type de mesure avec la mesure à base d'histogrammes présentent un inconvénient majeur par rapport à la mesure à base d'échantillons est que, les attributs d'une relation sont supposés tous de type numérique (dans les remarques du chacun de ces deux approches) [29] [30].

3.3. Evaluation de ces deux familles d'approches

Nous nous efforçons dans cette sous section à faire une évaluation globale et une comparaison entre les deux familles d'approches coopératives (les approches guidées par la requête et les approches guidées par un Workload of past queries) dans le cadre du traitement du problème des requêtes flexibles à réponses vides.

Avant de réaliser la comparaison et l'évaluation, nous décrivons d'abord les inconvénients et les avantages de chaque famille d'approches.

Dans la première famille d'approches guidées par la requête, comporte quelques inconvénients suivants [31] :

- La relaxation est effectuée indépendamment des connaissances qui existent dans la BD et de l'historique de l'interaction utilisateurs/BD, ce qui ne donne pas à coup sûr des requêtes qui soient proches, sémantiquement parlant, de la requête à réponse vide obtenue.
- La relaxation ne permet pas de savoir si les requêtes relaxées fournissent une réponse non vide avant de les évaluer.
- La relaxation des requêtes introduit une explosion combinatoire, notamment dans le cas des requêtes composées.

Le principe de cette famille d'approches est d'agir seulement sur les conditions impliquées dans la requête RV et plusieurs inconvénients ont été signalés. Néanmoins, cette famille présente un avantage considérable surtout dans l'espace mémoire occupé par le système qui est estimé faible par rapport à la seconde famille d'approches.

Concernant la seconde famille d'approches (guidées par Workload of past queries), l'inconvénient majeur dû à l'espace mémoire occupé par le Workload et sa gestion ainsi que le choix de la mesure de proximité sémantique utilisée pour estimer l'approximation entre requêtes. Néanmoins, cette famille comporte plusieurs avantages parmi ceux on a :

- Le degré du traitement du problème de réponses vide est élevé par rapport à la première famille.
- Pas de modification de la requête utilisateur (garder le choix des utilisateurs).
- La proximité entre requêtes garantit dans une grande échelle les réponses approximatives pour la requête RV.
- L'utilisation du Workload permet de garder la trace de l'utilisateur et l'historique de l'interaction utilisateurs/BD qui peuvent être utilisé dans d'autres domaines informatiques tels que les IHMs adaptatives [51] [52] [55], le web sémantique [53] et le data mining [54].
- Le traitement est effectué entre requête plutôt que sur la requête RV qui nécessite un processus récursif et incrémental.

Afin d'analyser les performances des deux familles d'approches, on a effectué une comparaison sur une base de données relationnelle contenant 1000 éléments (des données numériques et d'autres chaines de caractères). On a également procédé à un certain nombre d'expériences [25] [28]. Ces expériences nous permettent également de différencier la pertinence et la performance des approches récemment proposées (les 10 dernières années).

Nous avons utilisé trois approches récemment proposées dans chaque famille d'approches. Le tableau ci-dessous contient les résultats de la complexité et le taux du succès c.à.d, le taux du traitement des requêtes à réponses vides.

	Approches guidées par la requête			**Approches guidées par Workloads of past queries**		
	Bosc et al	***Ounelli***	***Muslea***	***Vague***	***AQUA***	***Nambiar***
La complexité	Moyenne	Moyenne	Forte	Faible	Moyenne	Faible
Le taux du succès	50%	43%	30%	60%	50%	75%

Tableau 2.5 Comparaison entre approches

Concernant la complexité de calcul, nous avons donné une vue générale selon les résultats du calcul que nous avons mené. Les résultats de ce tableau montrent clairement que les

approches guidées par un Workload of past queries donnent un meilleur résultat par rapport à la première famille d'approches.

En effet, dans le cadre de notre travail, nous nous intéressant aux travaux de la seconde famille d'approches, où en utilisant un type de distance pour mesurer la proximité sémantique entre requêtes.

4. CONCLUSION

Depuis les années 80, de nombreux travaux ont été effectués autour de la problématique d'interrogation des bases de données. L'un des problèmes les plus connus dans ce domaine est celui concernant les réponses vides et la façon de les éviter. Les requêtes flexibles, limitent dans une large mesure le risque d'obtention d'une réponse vide. Cependant, ce risque n'est pas totalement éliminé, et plusieurs approches ont été proposées dans la littérature pour traiter ce problème et que peuvent être classifiées en deux familles: Les approches guidées par la requête, et les approches guidées par un "workload of past queries".

Après avoir étudié et évalué ces deux familles d'approches et partant de l'hypothèse que les mesures de proximité entre ensembles flous sont généralement dérivées d'une fonction de distance définie sur les univers sous-jacents, nous nous intéressons dans notre travail à la dernière famille, en utilisant une mesure de distance afin d'estimer la proximité sémantique entre requêtes.

Dans ce contexte, l'objectif du chapitre suivant est de proposer une nouvelle approche fondée sur une mesure de proximité (ou de similarité) sémantique à base de distance particulière appliquée entre requêtes flexibles. Nous développons un algorithme de recherche des requêtes les plus proches sémantiquement parlant à la requête à réponse vide considérée.

PARTIE 02

CONTRIBUTION

Chapitre 03

Approche Fondée sur une Mesure de Proximité Sémantique

Chapitre 04

Mise en Œuvre et Implémentation

Chapitre 03 :

Approche Fondée sur une Mesure de Proximité Sémantique

1. INTRODUCTION

Au vu de l'étude accomplie au chapitre précédent, nous présentons dans ce chapitre notre contribution pour le traitement des requêtes flexibles à réponse vide, dans le but de retourner un ensemble de réponses approximatives à la requête. Notre approche, consiste à appliquée le principe des approches, guidées par "workload of past queries" basées, sur une mesure de proximité sémantique entre requêtes. Nous développons aussi un algorithme de recherche des requêtes les plus proches sémantiquement parlant à la requête à réponse vide.

L'objectif de ce chapitre alors, de proposer une approche pour estimer la proximité sémantique entre requêtes flexibles en utilisant une mesure de distance particulière dite la distance de Hausdorff et un algorithme de recherche de la requête la plus proche à une requête à réponse vide, pour cela, nous présentons dans un premier temps, les différentes catégories de distance entre ensembles flous puis les différentes mesures de proximité sémantique entre requêtes. Dans un second temps, nous découvrons notre solution exprimée par un algorithme de recherche.

2. DISTANCES ENTRE DES ENSEMBLES FLOUS

Les méthodes de mesure de distance entre des ensembles flous ont pris une place de plus en plus importante dans divers domaines de connaissance comme la télédétection, data mining, la reconnaissance de formes et l'analyse de données. Généralement, la plupart de ces méthodes sont utilisées entre ensembles classiques contenant des valeurs exactes. Pour cela, une question logique se présente: « si les nombres eux-mêmes ne sont pas connus exactement, comment la distance entre eux est une valeur exacte? ».

Dans ce contexte, nous présentons dans cette section deux catégories de distance utilisées pour mesurer la proximité sémantique entre ensembles flous et qui pouvant être appliquées entre requêtes flexibles. Ces deux catégories de distance sont : les distances classiques (ordinaires) et les distances floues.

2.1. Distances classiques

Il existe dans la littérature plusieurs mesures de distance appliquées entre ensembles classiques et qui peuvent être étendues entre ensembles flous. Nous citons ici celles qui sont les plus généralement utilisées pour mesurer la similarité entre ensembles flous. Dans ce cadre, nous faisons d'abord un rappel du concept de la distance.

Les objets comparés sont considérés comme des points dans un espace métrique et la distance entre ces objets se définit de la façon suivante [36] :

Soit E l'ensemble des individus du domaine étudié et soit une métrique D qui est une fonction de E x E$\rightarrow R^+$, on a : a, b, c $\in E^3$, les quatre propriétés suivantes doivent être vérifiées:

- Si a ≠ b, D (a, b) > 0 ---------- « *La séparabilité* »
- D (a, a) = 0 ---------- « *La réflexivité* »
- D (a, b) = D (b, a) ---------- « *La symétrie* »
- D (a, c) ≤ D (a, b) + D (b, c) ------- « *L'inégalité triangulaire*»

2.1.1. Distance de Hamming

La définition générale de la distance de Hamming [38] est la somme des valeurs absolues des différences entre les fonctions d'appartenance des objets correspondant. Si A et B deux sous ensembles flous dans le même ensemble de référence E, la distance de Hamming entre eux est: $D(A, B) = \sum_{i=1}^{n} |\mu A(xi) - \mu B(xi)|$

Avec n est le nombre d'éléments dans E= {x_1, x_2, …, x_n}. Cette définition s'étend très bien aux ensembles classiques. Pour prendre en compte le nombre d'éléments de l'ensemble de référence, la notion de « *distance de Hamming relative* » est appliquée et elle consiste à diviser le résultat de distance de Hamming ordinaire par le nombre d'éléments de E, soit : $\delta(A, B) = \frac{1}{n}\sum_{i=1}^{n} |\mu A(xi) - \mu B(xi)|$.

Exemple :

Soit un ensemble de référence E= {a, b, c, d, e, f, g} et deux sous ensembles flous du E : A et B qui sont représentés sous la forme suivante :

A= {0.5/a, 1/b, 0.7/c, 0/d, 0.2/e, 0.5/f, 0.9/g}

B= {0.8/a, 0.2/b, 0.7/c, 0.3/d, 0.1/e, 1/f, 0.6/g}

La distance de Hamming est :

D (A, B)= |0.5-0.8|+ |1-0.2|+ |0.7-0.7|+|0-0.3|+ |0.2-0.1|+ |0.5-1|+ |0.9-0.6|=2.3

La distance relative de Hamming est : δ (A, B) = D (A, B)/7= 0.328

Si A et B sont deux sous ensembles continus, la distance de Hamming est généralisée en utilisant « *la surface de Hamming* ». Elle est définie sur tout le segment tel que la réunion des deux sous ensembles possède une fonction d'appartenance non nulle. La formule générale dans ce cas est : $D(A, B) = \int_a^b |\mu A(x) - \mu B(x)| dx$

La mesure de proximité sémantique entre deux ensembles flous A et B est défini par la formule suivante : [50]

$$S(A, B)=2- d\big((A \cap B),\ [1]\big) - d\big((A \cup B),\ [0]\big).$$

2.1.2. Distance Euclidienne

Soit A ={[x_i , $\mu_A(x_i)$, $\nu_A(x_i)$]: $x_i \in E$}et B ={[x_i , $\mu_B(x_i)$, $\nu_B(x_i)$]: $x_i \in E$} deux sous ensembles flous de même ensemble de référence E= {x_1, x_2, …, x_n}, où $\mu_A(x_i)$ (resp. $\mu_B(x_i)$) le degré d'appartenance de x_i au sous ensemble A (resp. B) et $\nu_A(x_i) \rightarrow [0, 1]$ définit par l'inégalité triangulaire suivante: $0 \leq \mu_A(x_i) + \nu_A(x_i) \leq 1$.

La distance euclidienne entre A et B est exprimée par la formule suivante : [39]

$$e_h= \sqrt{\sum_{i=1}^{n} \max\{(\mu A(xi) - \mu B(xi))^2 , (\nu A(xi) - \nu B(xi))^2 \}} \quad \text{..... (1)}$$

Pour prendre en compte le nombre d'éléments de l'ensemble de référence, la notion de « *distance Euclidienne normalisée* » est appliquée et elle définit par la formule suivante :

$$q_h= \sqrt{\frac{1}{n}\sum_{i=1}^{n} \max\{(\mu A(xi) - \mu B(xi))^2 , (\nu A(xi) - \nu B(xi))^2 \}} \quad \text{..... (2)}$$

Exemple :

Soit un ensemble de référence E={x_1, x_2} et deux sous ensembles flous du E : A et B représentés comme suit :

A = {[x_1, 0.19, 0.21], [x_2, 0.01, 0.85]} et B = {[x_1, 0.24, 0.63], [x_2, 0.54, 0.02]}

En appliquant la formule (1), on trouve :

$$e_h= \sqrt{\max\{(0.19 - 0.24)^2 , (0.21 - 0.63)^2 \} + \max\{(0.01 - 0.54)^2 , (0.85 - 0.02)^2 \}}$$

$$= \sqrt{\max\{0.0025,\ 0.1764\} + \max\{0.2809, 0.6889\}}$$

$$= \sqrt{0.1764 + 0.6889}$$

$$= \sqrt{0.8653}$$

$$= 0.9302$$

La distance Euclidienne normalisée est : $q_h= \frac{1}{\sqrt{n}} \times e_h = \frac{1}{\sqrt{2}} \times 0.9302= 0.6578$.

2.1.3. Distance de Hausdorff

La distance de Hausdorff est un outil topologique qui mesure la ressemblance ou la dissemblance de deux sous-ensembles d'un espace métrique sous-jacent. Elle possède les

quatre propriétés de la distance citées précédemment. Elle est très utilisée dans le domaine de traitement d'images, appelée dans ce cadre « *Indicateur de similarité* », elle indique par exemple si deux formes sont les mêmes ou, si elles sont différentes.

Le calcule de cette distance entre deux ensembles A et B, est le maximum de deux quantités ; la première consiste à calculer pour chaque point de A sa plus faible distance à un point de B et on choisit ensuite le point qui maximise cette quantité. La deuxième quantité est l'inverse de la première puisque cette distance est asymétrique, donc on calcule pour chaque point de B sa plus faible distance à un point de A et on choisit ensuite le point qui maximise cette quantité [40][41][49].

La définition mathématique générale de cette distance est la suivante :

$d_H(A, B) = max \{H(A, B), H(B, A)\}$ ..... (1)

La distance de Hausdorff est alors, le maximum de $H(A, B)$ et $H(B, A)$ qui sont souvent appelées les composantes de la distance de Hausdorff.

Sachant que : $H(A, B) = Sup_{a \in A}\ d(a, B)$ et $d(a, B) = Inf_{b \in B}\ d(a, b)$ où $d(a, b)$ représente une distance quelconque, exprimée généralement par l'une des distances les plus populaires: la distance euclidienne définie comme suit : $d(a, b)=|a-b|$ [40].

La formule (1) peut être écrite sous la forme condensée suivante :

$d_H(A, B) = max \{sup_{a \in A}\ inf_{b \in B}\ d(a, b), sup_{b \in B}\ inf_{a \in A}\ d(a, b)\}$ ..... (1')

Notant que d_H est une métrique et, en particulier, la propriété suivante est vérifiée:

$d_H(A, B) = 0$ si et seulement si A = B. Habituellement, les égalités suivantes sont supposées être vraies $d_H(A, \emptyset) = d_H(\emptyset, B) = +\infty$ et $d_H(\emptyset, \emptyset) = 0$.

Exemple:

Soit $A = [a_1, a_2]$ et $B = [b_1, b_2]$ deux intervalles classiques et soit $d(u, v) = |u - v|$. Il est facile de voir que la distance de Hausdorff entre A et B est égale à (par (1')) :

$d_H(A, B) = \max(|a_1 - b_1|, |a_2 - b_2|)$.

La distance de Hausdorff entre les ensembles flous peut être floue ou scalaire. Ci-après, nous nous concentrons seulement sur la version scalaire. Pour l'évaluation floue, plus de détails sont disponibles dans [40] [41].

▪ ***Approche de Dubois et Prade (1980) :***

Dans [41], la distance de Hausdorff a été généralisée à des ensembles flous de la manière suivante :

Soit A et B deux ensembles flous de U, pour tout $r \in R^+$, $D_r(A)$ définit par :

$\mu_{Dr(A)}(u) = \sup_{v \in U} \{\mu_A(v) / d(u, v) \leq r\}$.

Sachant que, $D_r(A)$ est la dilatation de A par r, qui est le résultat de l'application à tous les points de A d'une opération locale de max dans une région de rayon r. Maintenant, la distance de Hausdorff généralisée s'écrit:

$$d_H(A, B) = inf\{r \in R^+ / A \subseteq D_r(B) \wedge B \subseteq D_r(A)\} \quad \text{..... (2)}$$

On peut facilement voir que la dilatation est seulement "horizontale". Il se peut qu'elle ne connu jamais l'autre ensemble et donc $d_H(A, B)$ ne peut pas être définie. Ainsi, deux ensembles flous doivent avoir le même supremum pour calculer la distance de Hausdorff entre eux. Cette observation est un inconvénient majeur du cette définition.

▪ ***Approche de Chaudhuri et Rosenfeld (1996) :***

Une autre définition de la distance de Hausdorff entre deux ensembles flous est proposée dans [40]. Cette définition est plus générale et elle est valide dans le cas de deux ensembles flous avec des degrés d'appartenances maximaux inégales (voir [40] pour plus de détails).

Soit A et B deux ensembles flous discrets et $T = \{t_1, t_2, \ldots, t_m\}$, l'ensemble de toutes les valeurs d'appartenance distinctes de A et de B, en d'autres terme c'est l'ensemble des coupes niveau α distinctes de A et de B. La distance de Hausdorff est définie par l'expression suivante :

$$d_H^2(A, B) = \frac{\sum_{i=1}^{m} t_i d_H(A_{t_i}, B_{t_i})}{\sum_{i=1}^{m} t_i} \quad \text{..... (3)}$$

Avec A_{ti} (resp. B_{ti}) représente l'ensemble d'éléments pour la coupe de niveau t_i de A (resp. B) et $d_H^2(A, B)$ peut être vu comme une distance moyenne des distances de Hausdorff entre les ensembles de niveau t_i de deux ensembles flous.

La distance $d_H(A_{t_i}, B_{t_i})$ est calculée par l'application de la formule (1), nous avons donc :

$d_H(A_{t_i}, B_{t_i}) = \max\{H(A_{t_i}, B_{t_i}), H(B_{t_i}, A_{t_i})\}$.

Exemple:

Soit *U = {1, 2, 3, 4, 5, 6, 7}* l'univers du discours et A, B deux ensembles flous de U défini comme suit : *A = {0.7/1, 0.2/2, 0.6/4, 0.5/5, 1/6} et B = {0.2/1, 0.6/4, 0.8/5, 1/7}.*

Soit l'ensemble $\{\alpha_1 = 0.2, \alpha_2 = 0.5, \alpha_3 = 0.6, \alpha_4 = 0.7, \alpha_5 = 0.8, \alpha_6 = 1\}$ de toutes les coupes niveau α distinctes de A et de B, donc on crée à partir de cet ensemble, l'ensemble d'éléments de *T* définit par : *T = {0.2, 0.5, 0.6, 0.7, 0.8, 1}.*

Le tableau suivant décrit la distance de Hausdorff entre les coupes de niveau α de A et B :

α_i	A_{α_i}	B_{α_i}	$H(A_{\alpha_i}, B_{\alpha_i})$	$H(B_{\alpha_i}, A_{\alpha_i})$	$d_H(A_{\alpha_i}, B_{\alpha_i})$
0.2	{1, 2, 4, 5, 6}	{1, 4, 5, 7}	1	1	1
0.5	{1, 4, 5, 6}	{4, 5, 7}	3	1	3
0.6	{1, 4, 6}	{4, 5, 7}	3	1	3
0.7	{1, 6}	{5, 7}	4	1	4
0.8	{6}	{5, 7}	1	1	1
1	{6}	{7}	1	1	1

Tableau 3.1 Distance de Hausdorff entre les coupes de niveau α de A et B

Pour calculer la distance $d_H^2(A, B)$, il suffit de calculer $d_H(A_{\alpha_i}, B_{\alpha_i})$ pour i=1,6 en appliquant la formule (1), nous obtenons par exemple pour α_1 :

$d_H(A_{\alpha_1}, B_{\alpha_1}) = \max\{H(A_{\alpha_1}, B_{\alpha_1}), H(B_{\alpha_1}, A_{\alpha_1})\}$ avec $A_{\alpha_1} = \{u1, u2, u3, u4, u5\} = \{1, 2, 4, 5, 6\}$ et $B_{\alpha_1} = \{v1, v2, v3, v4\} = \{1, 4, 5, 7\}$.

On calcule $H(A_{\alpha_1}, B_{\alpha_1}) = sup_{u \in A_{\alpha_1}} inf_{v \in B_{\alpha_1}} d(u, v)$, où $d(u, v) = |u - v|$

$$= sup\{inf(|u1 - v1|, |u1 - v2|, |u1 - v3|, |u1 - v4|),$$
$$inf(|u2 - v1|, |u2 - v2|, |u2 - v3|, |u2 - v4|),$$
$$inf(|u3 - v1|, |u3 - v2|, |u3 - v3|, |u3 - v4|),$$
$$inf(|u4 - v1|, |u4 - v2|, |u4 - v3|, |u4 - v4|),$$
$$inf(|u5 - v1|, |u5 - v2|, |u5 - v3|, |u5 - v|)\}$$
$$= sup(0, 1, 0, 0, 1) = 1.$$

De la même manière, nous pouvons calculer $H(B_{\alpha_1}, A_{\alpha_1}) = sup_{v \in B_{\alpha_1}} inf_{u \in A_{\alpha_1}} d(u, v) = 1$, par conséquent $d_H(A_{\alpha_1}, B_{\alpha_1}) = 1$. Nous procédons d'une manière analogue pour les autres

$d_H(A_{\alpha_i}, B_{\alpha_i})$, ses résultats sont donnés dans le tableau 3.1.

Donc $d_H^2\ (A, B) = (0.2 \cdot 1 + 0.5 \cdot 3 + 0.6 \cdot 3 + 0.7 \cdot 4 + 0.8 \cdot 1 + 1 \cdot 1) / 3.8$

$= 8.1/3.8 \cong 2.13$

Dans le cas d'ensembles flous continus, la formule (3) est modifiée sous la forme suivante :

$$d_H^2\ (A, B) = \int_0^1 t d_H(A_t, B_t) dt \Big/ \int_0^1 t dt = 2\int_0^1 t d_H(A_t, B_t) dt \quad \text{..... .....} \quad (4)$$

Exemple :

Soit U représente l'univers numérique du discours de variable linguistique "âge" et soit A= « *Environ trente* » et B= « *Entre 26 et 28* » deux ensembles flous de U défini par les fonctions d'appartenance : *A= (30, 30, 3, 3)* et *B = (26, 28, 1, 1)*. Pour appliquer la formule (4), nous précisons dans un premier temps A_α et B_α qui sont donnés par:

$A_\alpha = [3\alpha + 27, 33 - 3\alpha]$ et $B_\alpha = [\alpha + 25, 29 - \alpha]$. En utilisant la formule (4), on obtient :

$$d_H^2\ (A, B) = 2\int_0^1 t \max\left(\left|(t+25)-(3t+27)\right|, \left|(29-t)-(33-3t)\right|\right) dt = 7/2$$

Il a été démontré dans [40] que l'expression (3) (resp. (4)) est une métrique et se réduit à la distance de Hausdorff classique quand les ensembles sont non flous (booléens).

2.2. Distances floues

Récemment, plusieurs chercheurs sont focalisés sur le calcul de distance floue entre nombres [43] [44] [45]. Voxman [43] est le premier qui introduit la notion de distance floue entre nombres flous et ce, conformément à la logique suivante : « *la distance entre deux nombres flous devrait également être un nombre flou* ». Deux autres mesures de distance floue ; l'une basée sur la différence d'intervalles (en 2006) et l'autre mesure récemment proposée par S. Abbasbandy et al en 2010.

2.2.1. Distance de Voxman

Avant de présenter la distance de Voxman, nous rappelons la notion de coupe niveau α.

La coupe niveau α ou α-coupe de l'ensemble flou A pour une valeur donnée $\alpha \in [0, 1]$ est le sous-ensemble A_α de X définit par : $A_\alpha = \{x\ ;\ \mu_A(x) \geq \alpha\}$. La représentation α-coupe de A est la paire de fonctions, (L (α), R (α)), définie par :

$$L(\alpha) = \begin{cases} \inf\{x \mid x \in A\alpha\} & si\ \alpha > 0 \\ \inf\{x \mid x \in \text{Supp } A\} & si\ \alpha = 0 \end{cases} \quad \text{et} \quad R(\alpha) = \begin{cases} sup\{x \mid x \in A\alpha\} & si\ \alpha > 0 \\ \sup\{x \mid x \in \text{Supp } A\} & si\ \alpha = 0 \end{cases}$$

La distance floue de Voxman notée Δ est une fonction de F×F dans F définit par [43]:
Δ (A, B) (z) = $sup_{|x-y|=z}$ min$\{A(x), B(y)\}$.

Pour chaque paire de nombres flous A et B, soit Δ_{AB} désigne le nombre flou Δ (A, B). Si les représentations α-coupe de A et B sont ($A_1^L(\alpha)$, $A_1^R(\alpha)$) et ($A_2^L(\alpha)$, $A_2^R(\alpha)$), respectivement, alors la représentation α-coupe de Δ_{AB}, (L (α), R (α)), est donnée par :

$$L(\alpha)=\begin{cases}\max\{A2L(\alpha)-A1R(\alpha),0\}\ si\ \frac{1}{2}(A1L(1)+A1R(1)) \le \frac{1}{2}(A2L(1)+A2R(1)) \\ \max\{A1L(\alpha)-A2R(\alpha),0\}\ si\ \frac{1}{2}(A2L(1)+A2R(1)) \le \frac{1}{2}(A1L(1)+A1R(1))\end{cases}$$

R (α) = max $\{A_1^R(\alpha) - A_2^L(\alpha), A_2^R(\alpha) - A_1^L(\alpha)\}$.

2.2.2. Distance basée sur la différence d'intervalles

Considérons deux nombres flous représentés par leur f.a.t, A_1= (A_1, B_1, a_1, b_1) et A_2= (A_2, B_2, a_2, b_2). Par conséquent, les α coupes de A_1 et A_2 sont représentées par deux intervalles, respectivement $[A_1]_\alpha$ = $[A_1^L(\alpha), A_1^R(\alpha)]$ et $[A_2]_\alpha$ = $[A_2^L(\alpha), A_2^R(\alpha)]$, pour tout α ∈[0, 1].

Comme il peut être possible d'obtenir la distance entre deux intervalles par le biais de leur différence, nous utilisons l'opérateur de différence d'intervalles de deux intervalles $[A_1^L(\alpha), A_1^R(\alpha)]$ et $[A_2^L(\alpha), A_2^R(\alpha)]$ pour calculer la distance floue entre A_1 et A_2. Donc, la distance entre $[A_1]_\alpha$ et $[A_2]_\alpha$ peut être une de ces deux formules suivantes [44]:

$[A_1]_\alpha - [A_2]_\alpha$ si $\frac{A1L(1)-A1R(1)}{2} \geq \frac{A2L(1)-A2R(1)}{2}$

Ou

$[A_2]_\alpha - [A_1]_\alpha$ si $\frac{A1L(1)-A1R(1)}{2} < \frac{A2L(1)-A2R(1)}{2}$

Une autre formule pour calculer la distance floue à base de différence d'intervalles est donnée par la formule suivante : $\tilde{d}(A_1, A_2)$= ($d_{\alpha=1}^L$, $d_{\alpha=1}^R$, θ, σ) sachant que:

θ= $d_{\alpha=1}^L$-max $\left\{\int_0^1 d\alpha L, 0\right\}$ et σ =$\int_0^1 d\alpha R$ - $d_{\alpha=1}^R$

d_α^L= λ $[A_1^L(\alpha) - A_2^L(\alpha) + A_1^R(\alpha) - A_2^R(\alpha)] + [A_2^L(\alpha) - A_1^R(\alpha)]$

d_α^R= λ $[A_1^L(\alpha) - A_2^L(\alpha) + A_1^R(\alpha) - A_2^R(\alpha)] + [A_2^R(\alpha) - A_1^L(\alpha)]$

$$\lambda=\begin{cases} 1 \; si \; \frac{1}{2}(A1L(1) + A1R(1)) \geq \frac{1}{2}(A2L(1) + A2R(1)) \\ 0 \; si \; \frac{1}{2}(A2L(1) + A2R(1)) < \frac{1}{2}(A1L(1) + A1R(1)) \end{cases}$$

2.2.3. Nouvelle distance floue

En 2010, S. Abbasbandy et al [45] proposent une nouvelle forme de distance floue entre nombres flous en utilisant une représentation par un nombre flou triangulaire symétrique. Avant de présenter cette nouvelle méthode, nous découvrons d'abord les notions suivantes :

Soit A_1 et A_2 deux nombres flous de F avec leur représentation de α-coupes $(A_1^L(\alpha), A_1^R(\alpha))$ et $(A_2^L(\alpha), A_2^R(\alpha))$, respectivement. Les notions utilisées dans cette méthode sont : la distance $d(A_1, A_2)$ et la distance $d_\infty(A_1, A_2)$.

$$d(A_1, A_2)=\left|\int_0^{\frac{1}{2}}[(1-\alpha)(A1R(\alpha) - A2R(\alpha)) + \alpha(A1L(\alpha) - A2L(\alpha))]d\alpha + \int_{\frac{1}{2}}^1[\alpha(A1R(\alpha) - A2R(\alpha)) + (1-\alpha)(A1L(\alpha) - A2L(\alpha))]d\alpha\right|.$$

Les quatre propriétés suivantes doivent être vérifiées:

i. $d(A_1+A_3, A_2+A_3)= d(A_1, A_2)$ *pour tout* $A_1, A_2, A_3 \in F$;

ii. $d(A_1+A_2, 0) \leq d(A_1, 0) + d(A_2, 0)$ *pour tout* $A_1, A_2 \in F$;

iii. $d(\lambda A_1, \lambda A_2)= |\lambda| d(A_1, A_2)$ *pour tout* $A_1, A_2 \in F$ *et* $\lambda \in R$;

iv. $d(A_1+A_2, A_3+A_4) \leq d(A_1, A_3) + d(A_2, A_4)$ *pour tout* $A_1, A_2, A_3, A_4 \in F$.

La distance $d_\infty(A_1, A_2)=M$ avec $| A_1^R(\alpha)- A_2^R(\alpha) | \leq M$ et $| A_1^L(\alpha)- A_2^L(\alpha) | \leq M$ et l'application de théorème de valeur moyenne pour les intégrales, nous obtenons :

$$\begin{aligned} d(A_1, A_2) &\leq M\int_0^{\frac{1}{2}}(1-\alpha)d\alpha + M\int_0^{\frac{1}{2}}\alpha d\alpha + M\int_{\frac{1}{2}}^1\alpha d\alpha + M\int_{\frac{1}{2}}^1(1-\alpha)d\alpha \\ &\leq M\int_0^1(1-\alpha)d\alpha + M\int_0^1\alpha d\alpha \\ &\leq M \\ &\leq d_\infty(A_1, A2). \end{aligned}$$

Donc, $d(A_1, A_2) \leq d_\infty(A_1, A_2)$. La nouvelle distance floue notée $\tilde{d}(A_1, A_2)$ est donnée par la formule suivante :

$$\tilde{d}(A_1, A_2)=\left(\frac{d(A1,A2)+d_\infty(A1,A2)}{2}, \frac{d_\infty(A1,A2)-d(A1,A2)}{2}, \frac{d_\infty(A1,A2)-d(A1,A2)}{2}\right)$$

Avec sa représentation de coupes α $(\lambda_\alpha(A_1, A_2), \sigma_\alpha(A_1, A_2))$ où:

$\lambda_\alpha(A_1, A_2) = d(A_1, A_2) + \alpha\left(\frac{d_\infty(A1,A2) - d(A1,A2)}{2}\right) = \left(1 - \frac{\alpha}{2}\right) d(A_1, A_2) + \frac{\alpha}{2} d_\infty(A_1, A_2)$

Cette mesure de distance vérifiant les trois propriétés suivantes:

i. $\tilde{d}(A_1, A_2) = \tilde{0}$ *si et seulement si* $A_1 = A_2$ *;*

ii. $\tilde{d}(A_1, A_2) = \tilde{d}(A_2, A_1)$ *;*

iii. $\lambda_\alpha(A_1, A_2) \leq \lambda_\alpha(A_1, A_3) + \lambda_\alpha(A_3, A_2)$.

Delgado et al. [46] [47] ont étudié deux attributs de nombres flous B, l'ambiguïté noté A(B) et fuzziness noté F(B). L'ambiguïté peut être considérée comme une «diffusion globale» de la fonction d'appartenance, alors que le fuzziness décrit la comparaison entre les ensembles flous et leur complément. Ces notions sont définies comme suit:

$A(B) = \int_0^1 S(\alpha)\,[R(\alpha) - L(\alpha)]\,d\alpha$.

$F(B) = \int_0^1 S(\alpha)\,[q - p]\,d\alpha - \left\{\int_{\frac{1}{2}}^1 S(\alpha)\,[Lc(\alpha) - p]\,d\alpha + \int_{\frac{1}{2}}^1 S(\alpha)\,[R(\alpha) - L(\alpha)]\,d\alpha + \int_{\frac{1}{2}}^1 S(\alpha)\,[q - Rc(\alpha)]\,d\alpha + \int_0^{\frac{1}{2}} S(\alpha)\,[L(\alpha) - p]\,d\alpha + \int_0^{\frac{1}{2}} S(\alpha)\,[Rc(\alpha) - Lc(\alpha)]\,d\alpha + \int_0^{\frac{1}{2}} S(\alpha)\,[q - Rc(\alpha)]\,d\alpha\right\}$.

Où suppB= [p, q] et (L (α), R (α)) la représentation α-coupe de B. Ainsi, B_c est le complément de B avec sa représentation α-coupe (L_c (α), R_c (α)). La fonction S : [0, 1] → [0, 1] est une fonction croissante où S(0)=0 et S(1)=1. On dit que, S est une fonction de réduction régulière si $\int_0^1 S(\alpha)d\alpha = \frac{1}{2}$. Un calcul montre que, pour S(α)=α, on a :

$F(B) = \int_0^{\frac{1}{2}} (R(\alpha) - L(\alpha))\,d\alpha + \int_{\frac{1}{2}}^1 (L(\alpha) - R(\alpha))\,d\alpha$

A_1	A_2	A($\tilde{d}(A_1, A_2)$)	F($\tilde{d}(A_1, A_2)$)	A($\Delta(A_1, A_2)$)	F($\Delta(A_1, A_2)$)
(3, 2, 2)	(4, 3, 1)	1/12	1/8	68/75	17/20
(2, 1, 1)	(4, 1, 1)	0	0	2/3	1
(4, 1, 1)	(6, 2, 2)	1/8	3/16	53/54	4/3
(2, 4, 1)	(3, 2, 2)	7/48	1/32	203/216	1
(2, 1, 1)	(6, 1, 1)	0	0	2/3	1
(3, 2, 2)	(3, 1, 1)	1/8	3/16	1/2	3/4

Tableau 3.2 Comparaison de l'ambiguïté et Fuzziness

Le tableau ci-dessus, montre que l'ambiguïté et fuzziness de $\tilde{d}(A_1, A_2)$ sont moins que l'ambiguïté et fuzziness de $\Delta(A_1, A_2)$ qui est défini par Voxman. Nous pouvons voir que, lorsque le support de A_1 et A_2 sont disjoints, alors $\tilde{d}(A_1, A_2) = d_\infty(A_1, A_2)$ et dans ce cas, A ($\tilde{d}(A_1, A_2)$)= F ($\tilde{d}(A_1, A_2)$) = 0.

3. MESURES DE PROXIMITÉ SÉMANTIQUE À BASE DE DISTANCE

Dans tous les domaines de l'informatique dans lesquels on désire analyser de manière automatique un ensemble de données, il est nécessaire de disposer d'un opérateur capable d'évaluer précisément les ressemblances ou les dissemblances qui existent au sein de ces données. Sur cette base, il devient alors possible d'ordonner les éléments de l'ensemble, de les hiérarchiser ou encore d'en extraire des invariants. Pour qualifier cet opérateur nous utiliserons la notion de similarité ou encore la proximité.

Compte tenu de la forte relation entre proximité et distance, et en utilisant la distance de Hausdorff, on peut estimer dans quelle mesure deux requêtes flexibles sont proches, sémantiquement parlant.

3.1. Proximité sémantique

Le domaine de l'identification de la proximité sémantique a été considéré comme un sujet de recherche fortement recommandé dans les domaines du Web sémantique, de l'intelligence artificielle, de la recherche d'information et de la littérature linguistique.

La proximité sémantique définit par l'appréhension de la liaison entre deux objets et la capacité d'estimer à quel point deux ensembles se ressemblent ou se dissemblent [35] [50].

Il existe deux catégories principales des approches pour calculer la proximité sémantique : les approches basées-distance et les approches basées-contenu d'information. Nous utiliserons dans notre travail une mesure de proximité sémantique à base de distance (notion inverse de la proximité/similarité).

Une relation de proximité est une fonction *S* de E x E dans [0, 1], où E est un espace métrique et [0, 1] est un ensemble de valeurs de vérité qui représentent le degré auquel les objets sont semblables.

La relation de proximité doit satisfaire les trois propriétés suivantes [36]:

- $\forall x \in E : S(x, x) = 1$ ---------- « *La réflexivité*»
- $\forall x, y \in E^2 : S(x, y) = S(y, x)$ ---------- « *La symétrie* »
- $\forall x, y, z \in E^3 : t[S(x, y), S(y, z)] \leq S(x, z)$ ------ « *t-Transitivité* » où t est une t-norme.

Cette relation de proximité *S* s'appelle également t-équivalence. Si S (a, b)=1, alors a et b sont perçus trop similaires ou identiques, tandis que S (a, b)=0 indique que a et b sont totalement différents ou indépendants.

Il existe plusieurs définitions de relation entre la fonction de proximité et la distance. Dans le cas booléen par exemple, la formule la plus populaire est D=1- *S*. Il y'a aussi la formule développée par Koczy entre deux ensembles [37]: $S = \frac{1}{1+D(A,B)}$.

3.2. Mesures de proximité sémantique entre requêtes flexibles

Le choix d'un type de distance pour estimer la proximité sémantique entre des requêtes flexibles est tout à fait crucial. Il s'agit en effet de trouver la meilleure adéquation entre le but qui est le traitement des réponses vides et le comportement effectif de la mesure. Dans ce contexte, on utilise la distances de Hausdorff, pour ses propriétés intéressantes à savoir qu'elle représente une métrique et elle permet d'estimer à quel point deux ensembles se ressemblent, de plus, sur le plan calculatoire cette mesure se réduit toujours au calcul de la distance entre intervalles classiques.

Pour mesurer la proximité sémantique entre deux requêtes composées, il faut d'abord estimer la proximité entre les différents prédicats composants ces deux requêtes. Ci-dessous, on commence par exposer ce second calcul.

3.2.1. Cas des requêtes atomiques

Selon P. Bosc et al [31], le calcul de proximité sémantique entre requêtes, revient à calculer la proximité sémantique entre les prédicats de ces requêtes.

Soit Q=P et Q'=P' deux requêtes atomiques où P et P' sont deux prédicats graduels liés au même attribut A, modélisés au moyen des ensembles flous. Pour estimer à quel point Q et Q' sont proches sémantiquement parlant, on utilise la distance de Hausdorff entre les prédicats flous contenus dans ces deux requêtes et on écrit:

$Dist(Q, Q') = d_H^2 (P, P')$ ……..…….... (5)

Avec *Dist(Q, Q')* représente la distance entre Q et Q'. Plus la distance *Dist(Q, Q')* est petite plus Q et Q' sont proches.

Dénotant par *Prox (Q, Q')* la proximité sémantique entre Q et Q'. Basée sur la mesure *Dist(Q, Q'). Prox (Q, Q')* peut être calculée en deux étapes:

i. Normalisation de *Dist(Q, Q') :* en utilisant une fonction f_{norm} : $R^+ \rightarrow [0,1]$.

ii. Calcul de *Prox (Q, Q')*: en utilisant la relation, $Prox(Q, Q') = 1 - f_{norm}(Dist(Q, Q'))$ …….. (6)

Quelques expressions bien connues de f_{norm} sont [36]: $f_{norm}(x) = \min(1, x)$ et $f_{norm}(x) = x / (1 + x)$.

Une autre approche pour la définition d'une mesure de proximité fondée sur la notion de distance, représentée par la formule suivante [32]:

$$Prox(Q, Q') = \left(1 + \left(\frac{Dist(Q,Q')}{s}\right)^t\right)^{-1} \quad \text{..... (7)}$$

Les constantes positives s et t servent à ajuster la valeur de la mesure de proximité. La forme la plus simple de la formule (7) peut être obtenue en fixant s = 1 et t = 1.

Exemple :

Considérons une base de données relationnelle D, des employés d'une grande entreprise. Supposons que D a été interrogé par trois utilisateurs sur le "salaire" de certains employés et supposons aussi que le salaire maximum est de 10 € (voir le tableau 3.3).

EMP	***Requête***	***Représentation floue***
#1	Q_1 = "Rechercher des employés payés environ 2.8 €"	Environ_2.8 = (2.8, 2.8, 0.6, 0.6)
#2	Q_2 = " Rechercher des employés payés approximativement 3 €"	Approximativement_3 = (2.8, 3.2, 0.4, 0.4)
#3	Q_3 = " Rechercher des employés dont le salaire est très élevé "	Très_élevé = (6, 10, 1, 0)

Tableau 3.3 Exemple de requêtes utilisateur

Supposons maintenant que l'on est intéressé à rechercher les employés "bien payé" dans l'entreprise où le prédicat flou "bien payé" est modélisé par la f.a.t suivante (5, 10, 2, 0). Dans certaines situations particulières (comme cela sera expliqué plus loin), il pourrait être souhaitable d'estimer la proximité sémantique entre cette nouvelle requête, nommée Q, et les requêtes utilisateur précédentes. En appliquant la formule (4), on trouve :

$Dist(Q, Q_1) = 6$, $Dist(Q, Q_2) = 17/3$, $Dist(Q, Q_3) = 8/3$.

Par la formule (7), on obtient :

$Prox(Q, Q_1) = 0.14$, $Prox(Q, Q_2) = 0.15$, $Prox(Q, Q_3) = 0.27$.

On peut observer que $Prox(Q, Q_1) < Prox(Q, Q_2) < Prox(Q, Q_3)$. Par conséquent, Q est plus proche de Q_3 que Q_2 (resp. Q_1). Dans le cas où Q retourne des réponses vides, il est pratiquement intéressant de fournir à l'utilisateur les réponses de Q_3 à Q.

3.2.2. Cas des requêtes composées

Soit D une BD relationnelle contenant n attributs $A_1, A_2,\ldots, A_n$ avec $D(A_i)$ étant le domaine des valeurs relatives à A_i. Nous faisons l'hypothèse que $D(A_i)$ est fermé et borné.

Dans cette étude, nous nous intéressons seulement à des requêtes composées conjonctives de la forme $Q = P_1 \wedge P_2 \wedge \ldots \wedge P_k$ (le symbole '$\wedge$' dénote la conjonction 'et' interprété par l'opérateur 'min') et P_i (i = 1, k) est un prédicat flou relatif à A_i. En pratique, c'est ce type de requêtes qui est souvent utilisé et qui pose problème car la satisfaction d'une requête exige la satisfaction de tous les prédicats composant la requête.

Soit également Q' une requête flexible de la forme Q' = $P'_1 \wedge P'_2 \wedge \ldots \wedge P'_s$ et P'_j ($j=1,s$) est un prédicat flou relatif à A_j. Pour estimer la proximité sémantique entre les deux requêtes composées Q et Q ', on distingue trois cas:

Cas 1: Q et Q' portent exactement les mêmes attributs.
Cas 2: Q' possède tous les attributs spécifiés dans Q.
Cas 3: Q' ne couvre pas tous les attributs spécifiés dans Q.

Cas 1: Dans ce cas, Q' peut s'écrire sous la forme: Q'= $P'_1 \wedge ... \wedge P'_k$ avec $(P_i, P'_i) \in D(A_i)$, pour i = 1, k, qui signifie que P_i et P'_i sont des prédicats liés le même attribut A_i. Dans ce cas, Prox(Q,Q') est estimée en trois étapes:

i) Pour chaque couple (P_i, P'_i), i = 1, k, calculer Dist $(P_i, P'i) = d_H^2 (P_i, P'_i)$;

ii) Pour chaque couple (P_i, P'_i), i = 1, k, calculer Prox (P_i, P'_i) en utilisant, par exemple, la formule (7);

iii) Calculer Prox (Q, Q') = $\min_{i=1,k}$ Prox (P_i, P'_i).

Cas 2: Dans ce cas, Q' possède tous les attributs spécifiés dans Q. Elle peut donc s'écrire sous la forme:

$$Q'=P'_1 \wedge ... \wedge P'_k \wedge P'_{k+1} \wedge \ldots \wedge P'_s$$

Avec $(P_i, P'_i) \in D(A_i)$, pour i = 1, k et les prédicats P_j (pour j = k +1, s) ne sont pas spécifiées dans Q.

Pour estimer la proximité entre Q et Q', nous proposons deux stratégies :

(S_1) *Augmentation de Q* : L'idée est de compléter *Q* par des contraintes sur les attributs manquants (i.e., A_j pour j = k +1 à s). Puisque l'utilisateur ne spécifie aucune contrainte sur ces attributs, toutes les valeurs de leurs domaines sont autorisées. Ainsi, *Q* s'écrit :

$$Q = P_1 \wedge ... \wedge P_k \wedge D(A_{k+1}) \wedge \ldots \wedge D(A_s).$$

Avec cette réécriture de *Q*, on retrouve les mêmes conditions que dans le cas 1.
Prox(Q, Q') peut donc être obtenue en appliquant les trois étapes de premier cas.

(S_2) *Affaiblissement de Q'* : L'idée ici est de se focaliser uniquement sur les *attributs spécifiés* par l'utilisateur et de mesurer la proximité entre leurs contraintes spécifiées dans

Q et celles présentes dans *Q'*. Les attributs superflus présents dans *Q'* peuvent donc être supprimés. Ainsi, *Q'* est réécrite sous la forme suivante :

$$Q'_R = P'_1 \wedge ... \wedge P'_k .$$

Il est facile de voir que Q'_R est une variante relaxée de *Q'*. La relation suivante $\Sigma_{Q'} \subseteq \Sigma_{Q'_R}$ est toujours vérifiée (c.à.d, les réponses de *Q'* sont aussi des réponses de Q'_R).

Cette stratégie prendra tout son intérêt dans le cadre du traitement des requêtes à réponse vide puisque l'objectif est de proposer des solutions approximatives et alternatives à ces requêtes. Ainsi, si la mesure $Prox(Q, Q'_R)$ est élevée (ou au dessus d'un certain seuil), il est tout à fait acceptable de proposer les réponses de Q' comme réponses approximatives à la requête Q si son évaluation ne produit aucune réponse (i.e., $\Sigma_Q = \varnothing$). $Prox(Q, Q'_R)$ peut également être calculée comme dans le premier cas.

La stratégie S_2 permet de mesurer la proximité entre les contraintes des attributs présents dans les deux requêtes *Q* et *Q'*. Cette proximité peut être caractérisée de locale dont l'intérêt en pratique est démontré ci-dessus. Quant à la stratégie S_1, elle calcule une proximité globale entre *Q* et *Q'*. Tous les attributs même ceux non spécifiés dans Q sont pris en compte dans ce calcul. Dans le cadre du problème auquel on s'intéresse, cette propriété présente un désavantage non négligeable. En effet, supposons que Q = "Salaire = environs de 3 €" est une requête à réponse vide. Soit Q' = "Age = 27 ans $\wedge$ Salaire = [2.8, 3.2]". Le fait de tenir compte de la proximité entre la valeur de l'âge *27* et son domaine *D(Age)* (i.e. *Prox(27, D(Age))*, avec *D(Age) = [0, 100]*, affaiblirai significativement la proximité globale (basée sur l'opérateur *min*) entre Q et Q'. Ce qui pourrait conduire au rejet de *Q'* et donc de ses réponses alors que celles-ci pourraient intéresser l'utilisateur car elles satisfont approximativement son critère sur l'attribut "Salaire".

Cas 3: Dans ce cas, *Q*' ne couvre pas tous les attributs spécifiés dans *Q*. Elle peut donc s'écrire sous la forme :

$$Q' = P'_1 \wedge ... \wedge P'_b \wedge P'_{k+1} \wedge \ldots \wedge P'_s,$$

Avec $b < k$ et $(P_i, P'_i) \in D(A_i)$, pour $i = 1$ *à* b, et les prédicats P_j (pour $j = k+1$ *à* s) ne sont pas spécifiées dans *Q*. Pour calculer *Prox(Q, Q')* :

(i) On ajoute à Q' les domaines des attributs manquant parmi ceux spécifiés dans Q, Q' s'écrit donc $Q' = P'_1 \wedge ... \wedge P'_b \wedge D(A_{b+1}) \wedge \ldots \wedge D(A_k) \wedge P'_{k+1} \wedge \ldots \wedge P'_s$;

(ii) On applique l'une des deux stratégies décrites dans le cas 2.

4. APPROCHE PROPOSÉE

Nous présentons dans cette section, notre contribution, qui consiste à offrir une mesure permettant d'estimer la proximité sémantique entre requêtes flexibles lors du traitement des réponses vides. Pour cet objectif, nous avons, *primo*, intégré les principes des ensembles flous au contexte d'interrogation flexible de BD dans le but de fournir un ensemble de réponses discriminées plutôt qu'un ensemble plat de réponses, *secundo*, proposé une mesure de proximité sémantique entre requêtes pour le traitement des réponses vides, cette mesure utilise une distance particulière dite la distance de Hausdorff et *tertio* développé un algorithme de recherche des requêtes les plus proches sémantiquement parlant à la requête à réponse vide.

4.1. Principe

Notre approche vise à traiter les requêtes flexibles à réponse vide. Un des principes de cette approche est l'utilisation d'un ensemble des requêtes précédemment (Workload) évaluées par le système et dont les réponses sont non vides.

Dans cette sous section, nous montrons comment la mesure de proximité introduite (en Section 3) peut être utilisée pour traiter le problème de réponse vide.

Etant donné une requête Q à exécuter sur une relation R. Les étapes suivantes explicitent les traitements à effectuer pour retrouver un ensemble de réponses approximatives pour Q si elle échoue.

- Etant donné le workload, calculer la proximité sémantique de Q avec les requêtes du workload.
- La réponse à la requête Q est un ensemble de tuples ordonnés de la requête la plus proche à Q, en affectant à cet ensemble le même degré d'appartenance représenté par la valeur de proximité sémantique calculée entre ces deux requêtes.

Nous illustrons dans la figure suivante le principe du traitement des réponses vides.

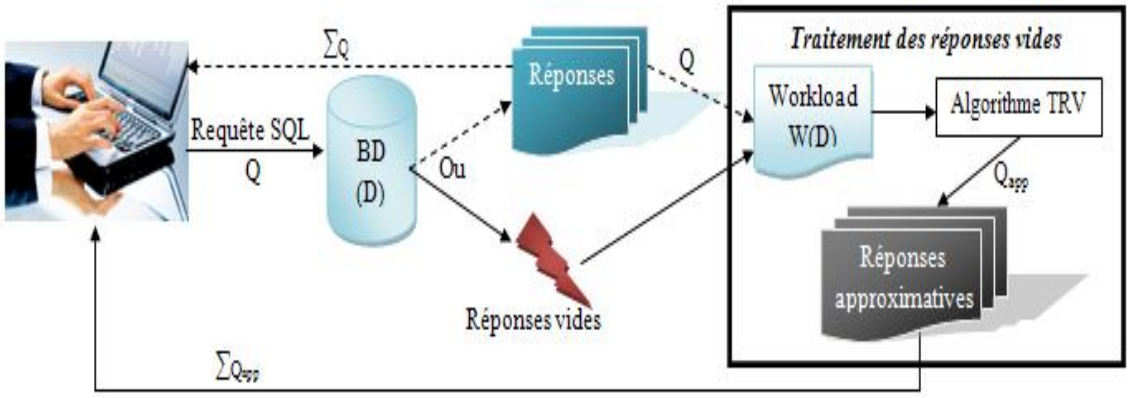


Figure 3.1 Schéma générale de l'approche proposée

Quand l'utilisateur effectue une interrogation sur une base de données traditionnelle (contient des données précises) nommée D, nous distinguons deux cas :

1) Un ensemble des réponses $\sum_Q$ retourné à l'utilisateur et mettre à jour le Workload W(D) (en flèches pointillées).

2) Un ensemble vide de réponses. Dans ce contexte un traitement des réponses vides est accompli via le Workload W(D), en appliquant notre algorithme TRV (**T**raitement des **R**éponses **V**ides) afin de retourner un ensemble de réponses approximatives $\sum_{Q_{app}}$ (où Q_{app} est la requête la plus proche sémantiquement parlant à Q).

4.2. Démarche de notre approche

Notre démarche se décompose de la façon suivante : une mesure de proximité sémantique est utilisée comme brique de base pour permettra l'estimation à quel point deux requêtes se ressemblent, cette mesure est ensuite utilisée pour développer un algorithme permettant de rechercher la requête la plus proche (sémantiquement parlant) à une requête à réponse vide. Soit $Q = P_1 \wedge \ldots \wedge P_k$ une requête flexible à réponse vide.

Supposons que nous disposons d'un workload *W(D)* (de la BD nommée D), c.à.d, une collection de requêtes que le système a exécutées et dont les réponses sont non vides. L'intérêt d'utiliser le workload *W(D)*, pour répondre approximativement à la requête *Q*, est qu'il peut révéler que, dans le passé, certaines requêtes précédemment exécutées par le

système dont les réponses sont non vides et que ces requêtes sont proches à *Q* sémantiquement parlant.

D'un point de vue utilisateur, il est plus utile de lui retourner les réponses de la requête la plus proche à *Q*, comme réponse à cette dernière, que de lui fournir une réponse vide. Pour ce faire, nous procédons de la manière suivante :

i) Partitionnons d'abord le workload *W(D)* en trois sous ensembles :

$$W_{=}(D, Q) = \{Q' / Q' \in W(D) \wedge |D(Q') \cap D(Q)| = k\},$$

$$W_{>}(D, Q) = \{Q' / Q' \in W(D) \wedge |D(Q') \cap D(Q)| > k\}, \text{ et}$$

$$W_{<}(D, Q) = \{Q' / Q' \in W(D) \wedge |D(Q') \cap D(Q)| < k\},$$

Où *D(Q)* désigne la liste des domaines d'attributs spécifiés dans *Q* et |*E*| désigne la cardinalité de l'ensemble *E*.

ii) *(a)- Si* $W_{=}(D, Q) \neq \varnothing$, pour élément *Q'* de $W_{=}(D, Q)$, nous estimons la mesure de proximité *Prox(Q, Q')* comme expliqué en section 3. Sur la base de cette mesure, nous effectuons un tri dans l'ordre décroissant des requêtes de $W_{=}(D, Q)$ afin d'ordonner ces requêtes de la plus proche à la plus distante de *Q*.

(b)- Sinon Si $W_{>}(D, Q) \neq \varnothing$, alors nous considérons $W_{>}(D, Q)$ et nous appliquons (a).

(c)- Sinon, nous considérons $W_{<}(D, Q)$ et nous appliquons (a).

iii) On choisit la requête la plus proche à *Q*, nommée Q_{App}, de l'ensemble ordonné $W_{=}(D, Q)$ *(ou* $W_{>}(D, Q)$ si (b) est appliqué, *ou* $W_{<}(D, Q)$ si (c) est appliqué), i.e., le premier élément de cet ensemble. Puis, nous retournons les réponses de Q_{App} comme réponses approximatives à la requête *Q* et l'affecter comme degré d'appartenance, la valeur de proximité mesurée $Prox(Q, Q_{app})$.

Remarque 01 : Si deux requêtes possèdent la même valeur de proximité sémantique, alors on choisissant la requête qui possède le plus grand nombre de réponses.

Remarque 02: Dans le cadre de notre travail, on n'utilise pas la formule standard de la distance de Hausdorff. Une partie de cette formule suffit pour mesurer à quel point la requête Q' est similaire à Q. La propriété de la symétrie de la distance n'est pas vraiment pertinente

dans ce cas, car c'est Q' qui est appelée à remplacer Q et non l'inverse. C'est pour cela on utilise juste la distance orientée de Hausdorff :

$$Dist(P_i, P'_i) = d_H^2(P_i, P'_i) = H(P'_i, P_i) = sup_{u \in P'i}\ inf_{v \in Pi}\ d(u, v)$$

4.3. Algorithmes de recherche

Nous présentons dans cette sous section deux algorithmes de recherche (Algorithmes TRV) de la plus proche requête à la requête à réponse vide Q. Le premier algorithme est utilisé dans le cas où Q est atomique (Q=P où P est un prédicat graduel) et le deuxième algorithme est employé quand Q est une requête composée.

4.3.1. Cas des requêtes atomiques

Selon notre démarche précitée, nous présentons dans cette partie le cas de l'interrogation à l'aide des requêtes atomiques.

```
Entrées: Q: requête initiale  /* Q := P */
         W_Q(D): ensemble des requêtes

          Begin
         01: exécuter Q;
         02: if ∑_Q≠Ø then
         03: Begin
         04:  if Q∉ W_Q(D)then W_Q(D):=W_Q(D)∪{Q};
         05: end;
         06: Return ∑Q;
         07: end
         08: else  /* ∑_Q=Ø */
         09: Begin
         10:  Filtrer(W_Q(D));
         11:  TriDesc(W_Q(D));
         12:  Q_App= Premier(W_Q(D));
         13:  Return ∑Q_App;
         14: end;
         15:end
Sorties: ∑Q ou bien ∑Q_App
```

Algorithme 01. Algorithme TRV d'une requête atomique

L'algorithme TRV d'une requête atomique prend en entrées, la requête utilisateur Q, le sous-ensemble $W_Q(D)$ qui est égale au début le Workload W(D).

La première étape (lignes 01 à 08) de cet algorithme consiste à déterminer si la requête utilisateur Q a réponse vide ($\sum_Q=Ø$) ou non ($\sum_Q\neq Ø$). Dans ce dernier cas, l'algorithme détermine si la requête Q existe ou non dans le Workload $W_Q(D)$ afin de décider, soit à ajouter cette requête au $W_Q(D)$ et afficher ses réponses, ou bien afficher directement les réponses (cas où Q existe déjà dans $W_Q(D)$).

La deuxième étape (lignes 09 à 14) est effectuée quand PRV est détecté. Dans ce contexte la première chose à réaliser, consiste à rechercher les requêtes du $W_Q(D)$ qui portent exactement le même attribut spécifié dans Q, ceci est réalisé par la procédure `Filtrer(`$W_Q(D)$`)`, qui modifie les éléments du $W_Q(D)$, en gardant seulement les requêtes de proximité non nulle, puis, on trie le $W_Q(D)$ (ligne 11) par la procédure `TriDesc(`$W_Q(D)$`)`; dans l'ordre décroissant selon le degré de proximité. Ensuite, on sélectionne la requête la plus proche à Q nommée Q_{App} (ligne 12) et on affiche ses réponses approximatives à la requête Q ($\sum Q_{App}$), qui prennent comme degré d'appartenance la valeur de proximité Prox (Q, Q_{App}).

Nous définissons dans ce qui suit, les différentes procédures utilisées dans notre algorithme :

1. *Procédure de filtrage de $W_Q(D)$* : Etant donné le Workload $W_Q(D)$, pour déterminer si ses requêtes possèdent le même attribut de Q, en exécutant l'algorithme suivant.

```
Procedure Filtrer(var set E)
    01: begin
    02: let Aux = E; E = ∅;
    03: while Aux ≠ ∅ do
    04:    begin
    05:      Q' = Premier (Aux);
    06:      calculer Prox(Q, Q');
    07:      if Prox(Q, Q')>0 then E = E ∪ {Q'};
    08:      Aux = Aux – {Q'};
    09:    end while
    10: end
```

Algorithme 02. Algorithme de filtrage de $W_Q(D)$

2. *Procédure de tri de $W_Q(D)$:* comme nous avons dit, dans l'algorithme 01, on réalise un tri dans l'ordre décroissant de requêtes du $W_Q(D)$ selon la valeur de proximité associée. Nous avons choisi le tri par sélection pour sa simplicité, dont l'algorithme est le suivant :

```
Procedure TriDesc(var set E)

    01: begin
    02: for i=1 to |E|-1 DO
    03: for j=i to |E| DO
    04:    begin
    05:      if Prox(Q,Qi)<Prox(Q,Qj) then
    06:      x=E.i;  /* E.i est le ième element de E */
    07:      E.i=E.j;
    08:      E.j=x;
    09:    end
    10: end
```

Algorithme 03. Algorithme de tri de $W_Q(D)$

3. *Procédure de calcul de proximité sémantique:* la mesure de proximité sémantique entre requêtes est fondée sur l'utilisation de distance de Hausdorff en appliquant la formule (7), dont l'algorithme est le suivant :

```
Procedure Prox(requête Q,Q')
Entrées: Q = P et Q' = P'

    01: begin
    02: calculer Dist(P,P');
    03: calculer Prox(P,P');
    04: Prox(Q,Q')= Prox(P,P');
    05: Return Prox(Q,Q');
    06: end

Sortie: proximité sémantique entre Q et Q'
```

Algorithme 04. Algorithme de calcul de proximité sémantique

Concernant la méthode pour calculer la distance de Hausdorff, on utilise l'approche de *Chaudhuri* et *Rosenfeld,* qui est plus générale et elle est valide dans le cas de deux ensembles flous avec des degrés d'appartenances maximaux inégales.

4.3.2. Cas des requêtes composées

Dans le cas de l'interrogation d'une BD à l'aide des requêtes composées, l'algorithme de recherche de la requête la plus proche à Q est le même que l'algorithme des requêtes atomiques, sauf les algorithmes des procédures utilisées sont différents.

Nous utilisons la mesure de proximité sémantique citée dans la sous section 3.2.2 où, on a distingué 3 cas possibles qui sont traités dans la procédure `Filtrer (`$W_Q(D)$`)` suivante :

```
Procedure Filtrer(var set E)
  01: begin
  02: let Aux=E; E=∅; Aux1=E, Aux2=E, cas1=faux;  cas2=faux;
  03: while Aux ≠ ∅ do
  04:    begin
  05:      Q' = Premier (Aux);
  06:      if s=k then /* premier cas */
  07:      begin
  08:        cas1=vrai;
  09:        calculer Prox(Q, Q');
  10:        if Prox(Q, Q')>0 then E = E ∪ {Q'};
  11:      end;
  12:      Aux = Aux – {Q'};
  13:    end;
  14: if cas1=vrai then exit
  15: else
  16: begin
  17:   while Aux1 ≠ ∅ do
  18:   begin
  19:      Q' = Premier (Aux1);
  20:      if s>k then /*deuxième cas */
  21:      begin
  22:        cas2=vrai;
  23:        Q'Rel= P'1Λ P'2Λ…ΛP'k ;
  24:         calculer Prox(Q, Q'Rel);
  25:        if Prox(Q, Q'Rel)>0 then E = E ∪ {Q'};
  26:      end;
  27:      Aux1 = Aux1 – {Q'};
  28:   end;
  29:  if cas2=vrai then exit
  30: else  /*troisième cas */
  31:   begin
  32:   Q'= Premier (Aux2);
  33:   Q''=Augmenter (Q');
  34:   Q''=Affaiblir (Q");
  35:   calculer Prox(Q, Q'');
  36:   if Prox(Q, Q'')>0 then E = E ∪ { Q'};
  37:     Aux2 = Aux2 – {Q'};
  38:   end;
  39: end;
  40: end.
```

Algorithme 05. Algorithme de filtrage de $W_Q(D)$ (cas des requêtes composées)

Les modifications effectuent dans la procédure *Prox(Q,Q')* sont défini dans l'algorithme qui suit :

```
Procedure Prox(requête Q,Q')
    01: begin
    02: for i=1 to k do
    03: begin
    04:   calculer Dist(Pi,P'i);
    05:   calculer Prox(Pi,P'i);
    03: end for;
    04: calculer Prox(Q,Q') = min i=1,kProx(Pi,P'i);
    05: Return Prox(Q,Q');
    06: end
```

Algorithme 06. Algorithme de calcul de proximité (cas des requêtes composées)

Dans ce qui suit, nous présentons les procédures utilisées dans le 3ème cas, qui sont : `Augmenter (Q')` et `Affaiblir (Q")`.

```
Procedure Augmenter (requête Q')
Entrées: Q et Q'

    01: begin
    02: i=1; j=1;
    03: while i=j do i++; j++;
    04: while i<=k do
    05: begin
    06: Q'= Q'∪ Di
    07: i++;
    08: end;
    09: Q"=Q';
    10: end.

Sortie: requête augmentée Q"
```

Algorithme 07. Algorithme de l'augmentation d'une requête

La procédure ci-dessous, présente la stratégie de l'affaiblissement d'une requête ; qui consiste à supprimer tout les attributs supplémentaires qui ne sont pas spécifiées dans la requête à réponses vides Q.

```
Procedure Affaiblir (requête Q")
Entrées: Q et Q"

    01: begin
    02: j=k+1;
    03: while j<=s do
    04: begin
    05: Q"= Q"-Pj ;
    06: j++;
    07: end;
    08: end.

Sortie: requête Q" relaxée
```

Algorithme 08. Algorithme de l'affaiblissement d'une requête

4.4. Exemple illustratif

Pour illustrer notre approche proposée, considérons une base de données D, représente le Travail, l'Expérience et le Salaire des sept employés, comme le montre dans le tableau 3.4.

EMP#	Travail	EXP	SALAIRE
t_1	Vice Président	15	7
t_2	Ingénieur conception	12	5
t_3	Ingénieur Système	3	4.5
t_4	Ingénieur conception	5	3.8
t_5	comptabilité	10	4.7
t_6	Secrétaire	5	6
t_7	Ingénieur logiciel	4	5.5

Tableau 3.4. Base de données des employés

Supposons que les deux requêtes suivantes Q_1 = *Autour_ 4* $\wedge$ *Entre_ 4_ 5* et Q_2= *Approximativement_10*$\wedge$ *Environ_5* ont été utilisé pour interroger D. Par exemple, Q_1 signifie que l'on est intéressé pour rechercher les employés dont l'expérience est autour de 4 ans et dont le salaire se situe *entre 4 et 5 €*. Le prédicat flou *Autour_4* (resp. *approximativement_10*) est modélisé par un ensemble flou discret défini par (*0,5/3, 1/4, 0.5/5*)

(resp. (*0,5/9, 1/10, 0.5/11, 0.2/12*)), tandis que *Entre_4_5* (resp. *Environ_5*) est modélisé par un ensemble flou continu défini par la f.a.t (*4, 5, 0.5, 0.5*) (resp. (*5, 5, 1, 1*)).

Supposons également que l'ensemble des réponses de chaque requête exécutée est stockée dans la workload *W(D)* (voir le tableau 4.2). Maintenant, nous allons examiner une nouvelle requête Q = *Proche_de_6* $\wedge$ *bien_ payé*, où *Proche_de_6 = (0,2/4, 0.5/5, 1/6, 0.5/7, 0.2/8)* et *bien payé = (7, 10, 1, 0)*. On peut voir que *Q* retourne un ensemble vide de réponses.

Requête	**Ensemble des réponses Σ_Q**
Q_1	$\{0.5/t_3, 0.16/t_4\}$
Q_2	$\{0.2/t_2, 0.7/t_5\}$

Tableau 3.5. Workload de base de données

Pour répondre à *Q*, nous appliquons notre approche, on obtient :

- Selon l'algorithme 01, la requête *Q* est échouée, dans ce cadre, nous exécutons la procédure de filtrage de *W(D)* et en calculant la proximité sémantique décrit dans l'algorithme 06 en tenant compte la remarque indiquée dans la démarche de notre solution, nous obtenons :
  Prox (Q, Q_1) = min (0,34, 0,17) = 0,17, Prox (Q, Q_2) = min (0,2, 0,18) = 0,18.

- Par l'algorithme 03, on obtient $Q_{App} = Q_2$.

- Les réponses approximatives à Q retournées à l'utilisateur sont les réponses de la requête Q_2 c'est-à-dire $\{t_2, t_5\}$. Chaque tuple réponse t est associée par un degré d'appartenance similaire, qui est la valeur de proximité sémantique entre Q et Q_2 et on obtient alors, les réponses suivantes : $\{0.18/t_2, 0.18/t_5\}$.

5. CONCLUSION

Nous avons proposé une approche pour le traitement du problème des requêtes flexibles à réponses vides. Notre solution est fondée sur une mesure de proximité sémantique entre requête à réponse vide et les requêtes du Workload, afin de retourner à l'utilisateur un ensemble des réponses approximatives de la requête la plus proche (sémantiquement parlant) à la requête à réponse vide.

Le principe de base de la mesure de proximité sémantique entre requêtes, consiste à calculer la proximité sémantique entre les prédicats contenants ces requêtes, en utilisant la distance de Hausdorff. L'ensemble des réponses approximatives retourné à la requête échouée est un ensemble de tuples réponses ordonnés de la requête la plus proche, en affectant à cet ensemble le même degré d'appartenance représenté par la valeur de proximité sémantique calculée entre ces deux requêtes.

Pour appliquer notre approche, nous avons développé des algorithmes de recherche de la requête la plus proche à une requête à réponse vide, où l'implémentation et la mise œuvre de ces algorithmes fera l'objet du chapitre suivant, qu'il montre aussi les bonnes performances de notre approche par rapport aux approches proposées dans la littérature.

Chapitre 04 :

Mise en Œuvre et Implémentation

1. INTRODUCTION

Dans le chapitre précédent, nous avons présenté notre approche pour le traitement des requêtes à réponses vides. Nous illustrons maintenant son utilisation dans l'environnement de programmation "Java". Celui-ci offre une grande souplesse grâce à ses caractéristiques fonctionnelles.

Nos efforts sont orientés alors, vers la réalisation et la mise en œuvre de notre approche, les résultats des expérimentations effectuées pour évaluer les pertinences et les performances.

Dans ce chapitre, nous commençons à décrire la démarche et l'architecture générale selon laquelle notre application a été implémentée. Ensuite, nous présentons le fonctionnement de l'interrogation flexible où nous décrivons les prédicats flous utilisés et l'exécution d'une requête flexible. Enfin, nous présentons l'implémentation de l'approche proposée avec quelques exemples de traitement des requêtes à réponses vides selon les deux types de requêtes : requêtes atomiques et requêtes composées.

2. DÉMARCHE

L'objectif de ce chapitre, est la réalisation concrète de notre solution pour le traitement des requêtes à réponses vides, qui sera implémentée dans une application représente quelques caractéristiques techniques d'une unité centrale (UC) d'une entreprise d'achat des matériels informatique. L'étude de ce type de matériel informatique nous a permis de définir des prédicats flous pour ses caractéristiques à savoir, la vitesse du processeur, la taille de la mémoire centrale, la taille du disque dur… etc.

Notre démarche selon laquelle notre application a été implémentée se décompose en deux grandes parties :

1) *La réalisation concrète de l'interrogation flexible de BD* : L'introduction de la flexibilité sur les éléments de BD satisfaisant une interrogation, basée sur la théorie des sous-ensembles flous et en utilisant le concept de prédicat flou pour exprimer des préférences d'une manière à retourner un ensemble de réponses discriminées plutôt qu'un ensemble plat de réponses. Les résultats d'une requête flexible sont alors plus au moins pertinents selon leur degré de satisfaction aux contraintes de l'interrogation.

2) *La réalisation concrète de notre approche :* C'est la partie la plus importante, où nous concentrons notre effort sur l'implémentation et l'expérimentation de l'approche que nous avons proposé dans le chapitre précédent pour le traitement des requêtes à réponse vide.

3. ARCHITECTURE GÉNÉRALE DE L'APPLICATION

Pour atteindre notre objectif, nous présentons dans cette section, l'architecture générale de l'application en suivant notre démarche précitée (voir la figure 4.1).

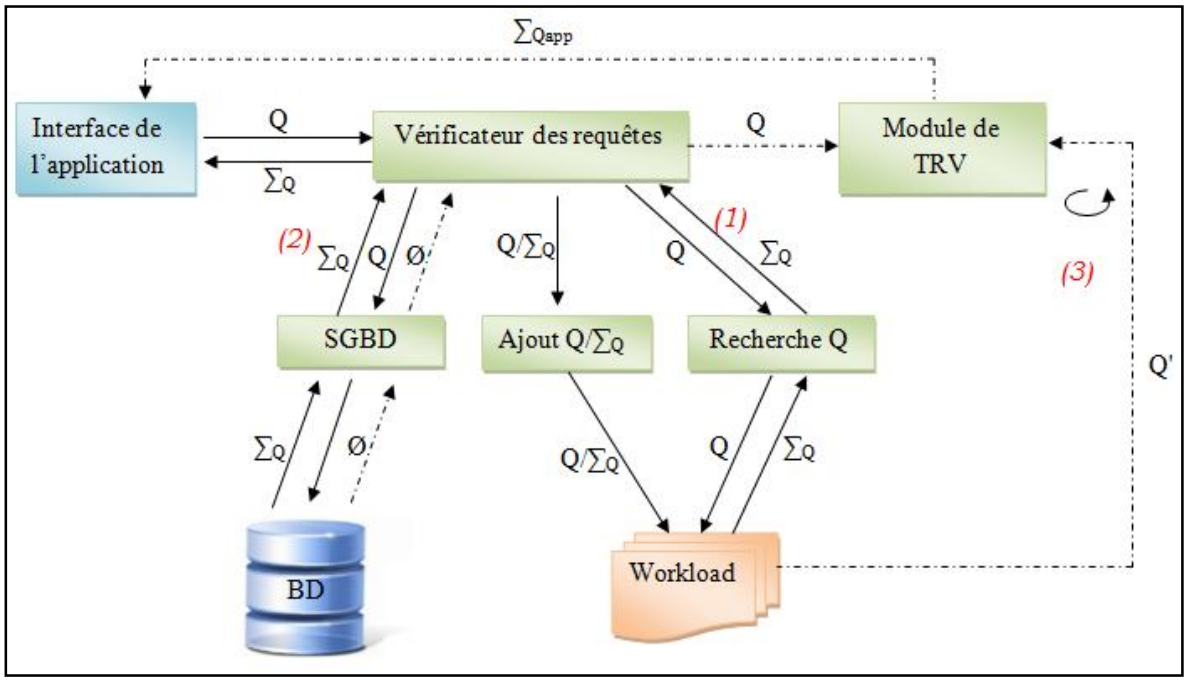


Figure 4.1. L'architecture générale de l'application

L'architecture générale illustrée dans la figure 4.1, montre les principaux composants de notre application, qui se commence à partir de l'interface d'interrogation de BD. Cette dernière permet aux utilisateurs de formuler leurs requêtes, et selon le vérificateur des requêtes, il détermine d'abord (le symbole (1)) si la requête existe ou non dans le workload afin de retourner directement ses réponses à l'utilisateur sans passée par le SGBD, sinon (le symbole (2)), il vérifier si la requête à réponse vide ou non à partir de SGBD qui exécute cette requête. Dans le cas où la requête ayant des réponses, il retourne ses réponses à l'utilisateur qui seront affichées dans l'interface et mettre à jour le Workload (ajout de cette requête avec ses réponses dans le Workload).

Dans le cas opposé (le symbole (3)), la requête Q est traitée via le module de traitement des réponses vides "Module TRV" qui fondé sur notre approche réalisant une estimation à base de proximité sémantique entre la requête échouée Q et chaque requête Q' du Workload (le signe ↻ signifie le parcourt de toutes les requêtes du Workload filtré) afin de déterminer la requête la plus proche sémantiquement parlant à Q nommée Q_{app}, où ses réponses seront retournées à l'utilisateur comme réponses approximatives à Q.

Pour illustré mieux le fonctionnement de notre architecture, nous présentons dans la section suivante les différents prédicats flous utilisés dans notre application et le fonctionnement d'une requête flexible.

4. FONCTIONNEMENT DE L'INTERROGATION FLEXIBLE

Comme nous avons dit dans la section 2, que notre application représente quelques caractéristiques techniques des unités centrales de matériel informatique, défini par la relation classique suivante : UC (**Num_UC**, Desig, V_Proc, T_MC, T_DD, Prix).

Pour effectue une interrogation flexible de BD relationnelle classique, il est nécessaire d'utiliser le concept de prédicat flou associé à un attribut d'une relation. Dans ce cadre nous présentons dans un premier temps les différents attributs utilisés, leur désignation, le type de données et les prédicats flous établés (voir le tableau 4.1) et dans un second temps le fonctionnement d'une requête flexible.

Attributs	***Désignation***	***Type***	***Prédicats flous***
Num_UC	La clé primaire de la relation, est un numéro séquentiel de l'UC	Entier	Pas de prédicat flou
Desig	La désignation de l'UC	Chaine de caractère	Pas de prédicat flou
V_Proc	La vitesse du processeur	Réel	Lente, Moyenne, Rapide
T_MC	La taille de la mémoire centrale	Entier	Très_Petite, Petite, Moyenne, Grande
T_DD	La taille du disque dur	Entier	Petit, Moyen, Grand
Prix	Le prix d'achat d'une unité centrale	monétaire	Moins_Chère, Chère, Plus_Chère

Tableau 4.1. Description des attributs utilisés et leurs prédicats flous

Les champs, **Num_UC** (un numéro séquentiel) et Desig (une chaîne de caractères), nous n'allons pas leur appliquer des prédicats flous. Les champs V_Proc, T_MC, T_DD et Prix sont des données qu'on peut définir des prédicats flous, qui sont représentés dans la sous section suivante.

4.1. Les prédicats flous

Nous présentons dans cette sous section les différents prédicats flous associés aux attributs de notre table SQL:

- ***Les prédicats flous de l'attribut V_Proc :*** L'attribut V_Proc possède trois (03) prédicats flous : Lente, Moyenne et Rapide, qui sont modélisés par des ensembles flous discrets défini respectivement comme suit : (1/1.2, 0.6/1.6, 0.66/1.66), (0.5/2, 0.65/2.1, 0.7/2.13, 0.8/2.2, 1/2.3, 1/2.4), (0,53/2.53, 0.6/2.6, 0.66/2.66, 0.77/2.7, 0.87/2.8, 0.97/2.93, 1/3, 1/3.06, 1/3.2).

- ***Les prédicats flous de l'attribut T_MC:*** L'attribut T_MC possède quatre (04) prédicats flous : Très_Petite, Petite, Moyenne et Grande, qui sont modélisés par des ensembles flous discrets défini respectivement comme suit : (1/0.128, 1/0.256, 0.9/0.384, 0.5/0.500, 0.5/0.512, 0.3/0.628, 0.2/756), (1/1, 0.5/2, 0.25/3), (0.3/2, 1/3, 0.7/4), (1/8, 0.7/6, 0.5/4).

- ***Les prédicats flous de l'attribut T_DD:*** L'attribut T_DD possède trois (03) prédicats flous : Petit, Moyen et Grand, qui sont modélisés par des ensembles flous discrets défini respectivement comme suit : (1/40, 0.9/80, 0.5/160, 0.3/250), (0.7/250, 0.9/300, 1/320, 0.5/500, 0.35/520, 0.2/640), (0.1/320, 0.4/500, 0.5/520, 0.8/640, 0.9/750, 1/1000, 1/1500, 1/2000).

- ***Les prédicats flous de l'attribut Prix:*** L'attribut Prix possède trois (03) prédicats flous : Moins_Chère, Chère, Plus_Chère, qui sont modélisés par des ensembles flous continus défini respectivement par les quatre valeurs (A, B, a, b) de la f.a.t: (250, 400, 150, 150), (640, 900, 100, 100), (1500,5000, 550, 0). La représentation graphique de ces prédicats est illustrée dans la figure suivante:

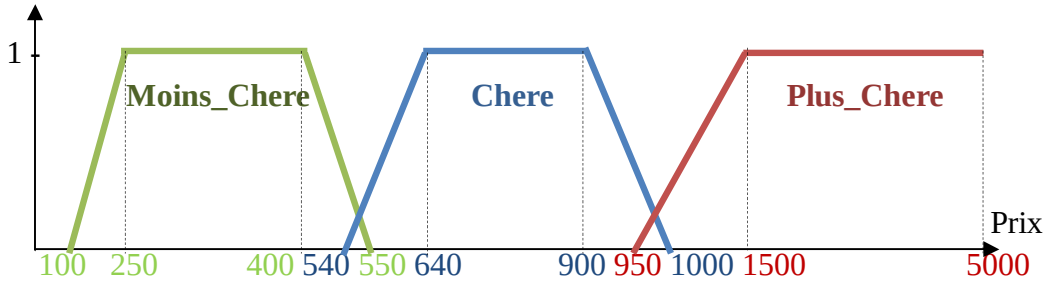


Figure 4.2. Représentation graphique des prédicats flous du l'attribut "Prix"

Pour utiliser ces prédicats flous dans l'interrogation flexible de BD, nous avons choisi de stocker tous les prédicats flous modélisés par des ensembles flous discrets dans une seule table à deux colonnes ; la première indique le nom du prédicat flou et la deuxième définit l'ensemble de couples (μ, x) sachant que : μ est le degré d'appartenance de l'élément x (voir le tableau 4.2).

Nom de Prédicat	***Couples (μ, x)***
Lente	(1, 1.2), (0.6, 1.6), (0.66, 1.66)
Moyenne	(0.5/2, 0.65/2.1, 0.7/2.13, 0.8/2.2, 1/2.3, 1/2.4)
Rapide	(0,53, 2.53), (0.6, 2.6), (0.66, 2.66), (0.77, 2.7), (0.87, 2.8), (0.97, 2.93), (1, 3), (1, 3.06), (1, 3.2)
Petite	(1, 1), (0.5, 2), (0.25, 3)
Moyenne	(0.3, 2), (1, 3), (0.7, 4)
Grande	(1, 8), (0.7, 6), (0.5, 4)
Petit	(1, 40), (0.9, 80), (0.5, 160), (0.3, 250)
Moyen	(0.7, 250), (1, 320), (0.5, 500), (0.2, 640)
Grande	(0.1, 320), (0.4, 500), (0.5, 520), (0.8, 640), (0.9, 750), (1, 1000), (1, 1500), (1, 2000)

Tableau 4.2. Les prédicats flous modélisés par des ensembles flous discrets

Les autres prédicats flous qui sont modélisés par des ensembles flous continus, quatre valeurs au maximum sont nécessaires pour représenter les fonctions floues envisagées. Pour ne pas avoir à stocker les valeurs des ordonnées nous avons choisi un système de code pour chaque "patron de fonction floue" où en utilisant le rapprochement de nombre flou type-LR. On stockera également ce code qui, combiné aux valeurs numériques nous permettra de

retrouver le prédicat flou. On peut donc, stocker les valeurs et le code d'un prédicat dans un même tableau (fixe) de cinq éléments (voir le tableau 4.3).

Nom de Prédicat	***Valeurs***
Moins_Chère	[1], [250], [400], [150], [150]
Chère	[1], [640], [900], [100], [100]
Plus_Chère	[2], [1500], [5000], [550], [ ]

Tableau 4.3. Les prédicats flous modélisés par des ensembles flous continus

La première valeur indique le code associé au prédicat flou suivi par les quatre valeurs de la fonction d'appartenance (A, B, a, b).

Les figures suivantes montrent la convention de codage des différentes formes de prédicats flous.

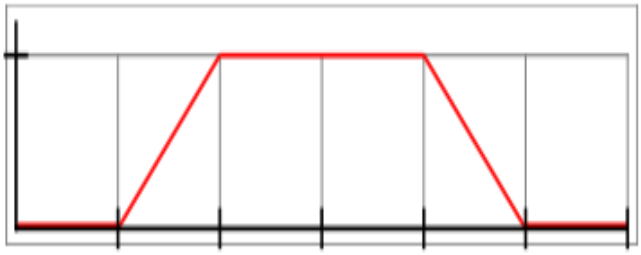


Figure 4.3. Code 1 : Trapèze

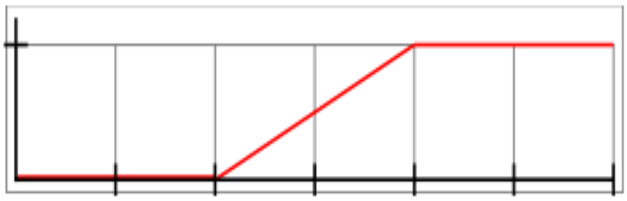


Figure 4.4. Code 2 : Côte

Avant de présenter le calcul de degré d'appartenance, nous rappelons le concept de nombre flou type-LR :

D'une façon générale une fonction d'appartenance μ de sous ensemble flou E est représentée comme suit :

$$\mu_E(x)=\begin{cases} f(x)\ pour\ x \in [A-a, A] \\ 1\ \ pour\ x \in [A, B] \\ h(x) pour\ x \in [B, B+b] \end{cases}$$

Où $f\ (x)$ et $h\ (x)$ sont les fonctions monotones croissantes et décroissantes de $[A-a, A]$ et $[B, B+b]$, respectivement. Un cas particulier de nombre flou générale appelé « nombre flou type-LR » où les fonctions $f\ (x)$ et $h\ (x)$ sont approchées par $L\left(\frac{A-x}{a}\right)$ et $R\left(\frac{x-B}{b}\right)$, respectivement.

Le calcul de degré d'appartenance d'une réponse t_i à une requête flexible est basé sur le système de code, les quatre valeurs de la f.a.t et le rapprochement de nombre flou type-LR :

Si Code= 1 : Trapèze

- 1 : Si $t_i \geq A$ et $t_i \leq B$
- 0 : Si $t_i \leq A-a$ ou $t_i \geq B+b$
- $\frac{A-ti}{a}$: Si $t_i > A-a$ et $t_i < A$
- $\frac{ti-B}{b}$: Si $t_i > B$ et $t_i < B+b$

Si Code= 2 : Côte

- 1 : Si $t_i \geq A$
- 0 : Si $t_i \leq A-a$
- $\frac{A-ti}{a}$: Si $t_i > A-a$ et $t_i < A$

Après avoir présenté les prédicats flous que nous avons utilisés dans notre application, nous montrons dans la sous section suivante l'exécution d'une requête flexible.

4.2. Exécution d'une requête flexible

Prenons notre base de données possédant une table UC qui stocke des données sur quelques caractéristiques techniques d'une unité centrale. Le tableau suivant décrit quelques données de notre table UC.

Num_UC	Desig	V_Proc	T_MC	T_DD	Prix
1	PACKARD Bell iMedia i4573FR	2,93	4	1000	469,90 €
2	COMPACK Presario CQ5325FR	2.7	3	500	287,90 €
3	HP Pavilion p6340FR	2.7	4	1000	439,00 €
4	ACER Aspire AZ5610-UF9p	2.66	4	750	718,50 €

Tableau 4.4. Exemple de données de la table UC

L'exemple suivant montre les différentes étapes de l'exécution d'une requête SQL flexible. On souhaite obtenir les désignations des unités centrales moins chères. Le prédicat Moins_Chère est donc appliqué au champ Prix. La première étape consiste à réécrire la requête afin de la transmettre au SGBD. Celui-ci nous renvoie les réponses ordonnées de la requête avec leur degré d'appartenance.

Desig	Prix	Degré
COMPACK Presario CQ5325FR	287,90 €	1
HP Pavilion p6340FR	439,00 €	0.74
PACKARD Bell iMedia i4573FR	469,90 €	0.534

Tableau 4.5. Exemple d'application d'un prédicat

Dans le cas d'une interrogation flexible à l'aide d'une requête composée conjonctive à deux prédicats flous par exemple. Celle-ci est envoyée directement au SGBD. Un tuple retourné par un seul des deux prédicats ne sera pas présent dans le résultat. Le degré d'appartenance des tuples valides aura pour valeur le minimum des degrés calculés pour les deux prédicats.

On souhaite par exemple d'obtenir la désignation et la taille (en Giga) de disque dur des unités centrales moins chères avec une grande taille de disque dur. Le prédicat Moins_Chère est donc appliqué au champ Prix et le prédicat Grand est appliqué au champ T_DD.

Desig	T_DD	Prix	Degré
HP Pavilion p6340FR	1000	439,00 €	0.74
PACKARD Bell iMedia i4573FR	1000	469,90 €	0.534
COMPACK Presario CQ5325FR	500	287,90 €	0.4

Tableau 4.6. Exemple d'application de deux prédicats

Après avoir exécuté et vérifié le succès des requêtes flexibles via le Vérificateur des requêtes, elles sont met dans le Workload W(D) avec leurs nombre de réponses pour utiliser dans le processus de traitement des réponses vides. Donc, le Workload W(D) est une table à deux colonnes, contenant les requêtes et leurs nombre de réponses (voir le tableau 4.7).

Requête	***Nbre***
Select Desig, Prix From UC Where Prix(Moins_Chère)	43
Select Desig, Prix, T_DD From UC Where Prix(Moins_Chère) and T_DD(Grand)	28

Tableau 4.7. Exemple de données du Workload

Après avoir présenté la première partie de notre démarche qui consiste à réaliser l'interrogation flexible de BD, nous présentons dans la section suivante, la réalisation concrète de notre approche pour le traitement des requêtes à réponse vide.

5. EXEMPLES DU TRAITEMENT DES RÉPONSES VIDES

Dans cette section nous présentons notre expérimentation, à savoir l'interrogation flexible de BD, traitement des requêtes à réponse vide à l'aide de l'approche que nous avons proposée. Notre application est implémentée en utilisant le langage orienté objet JAVA et le SGBD ACCESS 2007.

Au début de l'application, notre travail était prévu en 3 étapes. Dans un premier temps, la réalisation de l'interrogation flexible de BD. Puis une phase de réalisation du module permettant le traitement des requêtes à réponses vides. Et enfin une phase de test afin de s'assurer du bon fonctionnement du module réalisé.

Cependant lors de la première phase de notre travail, de nombreux problèmes ont été découverts et qui sont résolus. Pour effectuer une interrogation flexible de BD, l'utilisateur

sélectionne les attributs qu'il souhaite voir afficher dans la réponse puis il réalise ses conditions à travers une collection des composants graphiques (Widget) facilitant la formulation de la requête et minimisant les risques d'erreurs de syntaxe ou de sémantique. (Voir les figures 4.5 et 4.6)

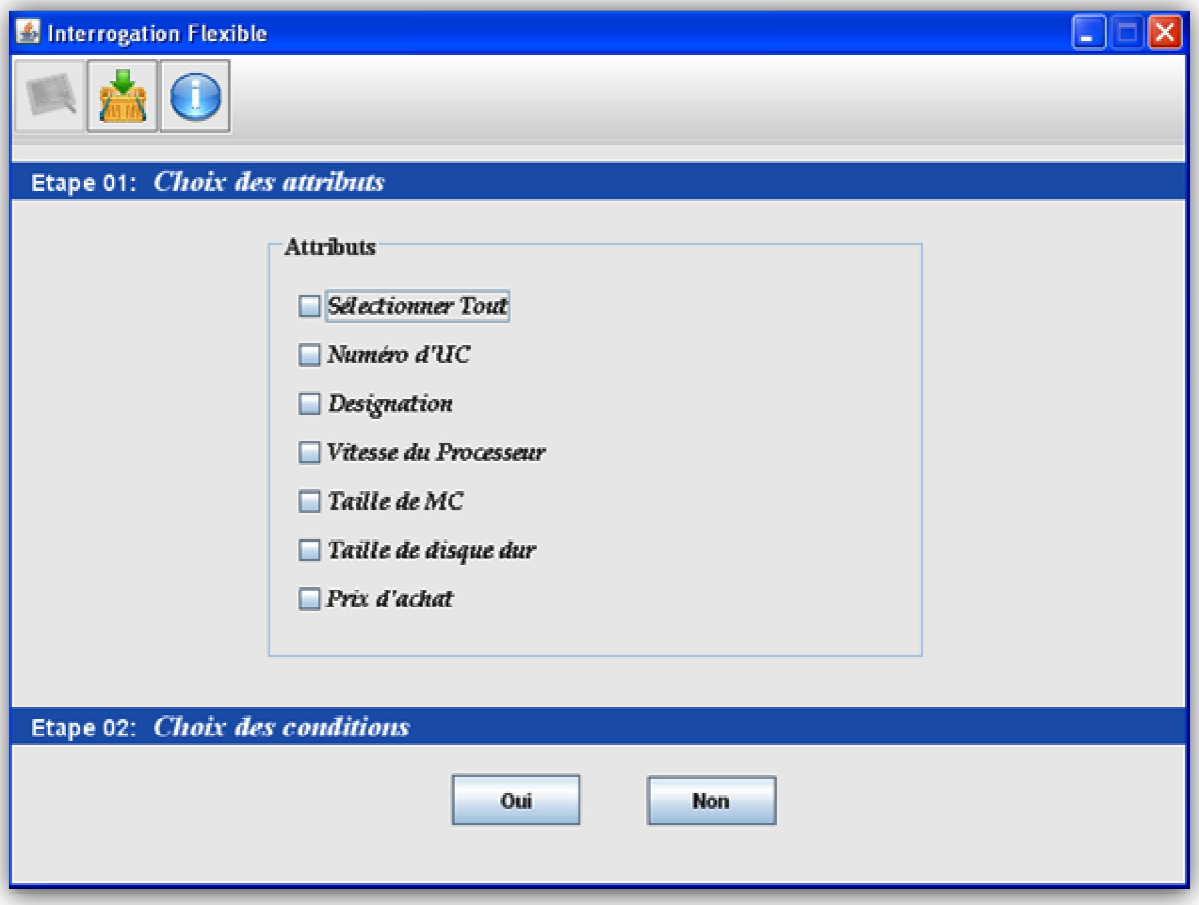


Figure 4.5. Sélection des attributs

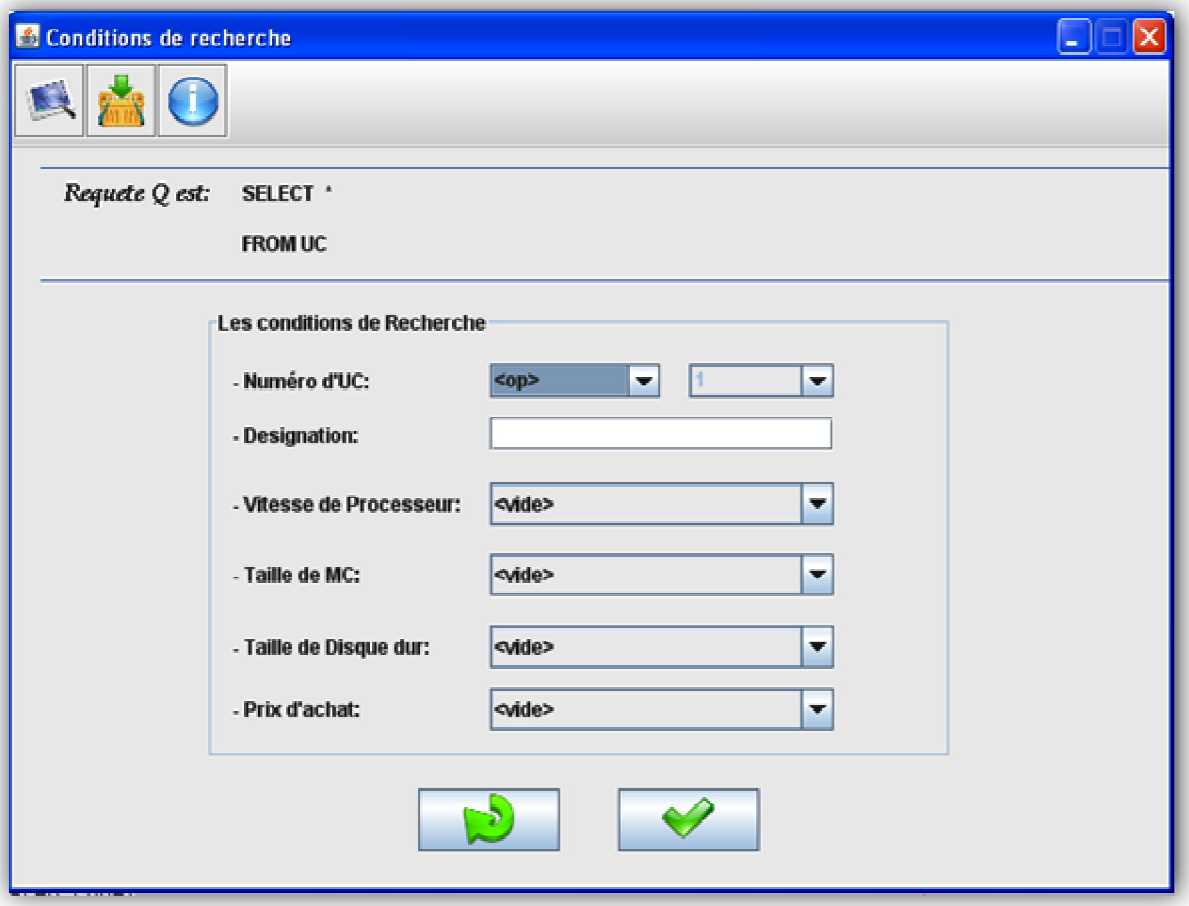


Figure 4.6. Formulation des conditions

Une fois la requête formulée elle est envoyée au SGBD et donne à l'utilisateur la possibilité de confirmer l'envoi ou de modifier la requête. Dans les figures suivantes nous montrons l'exécution des requêtes dont les réponses sont non vides.

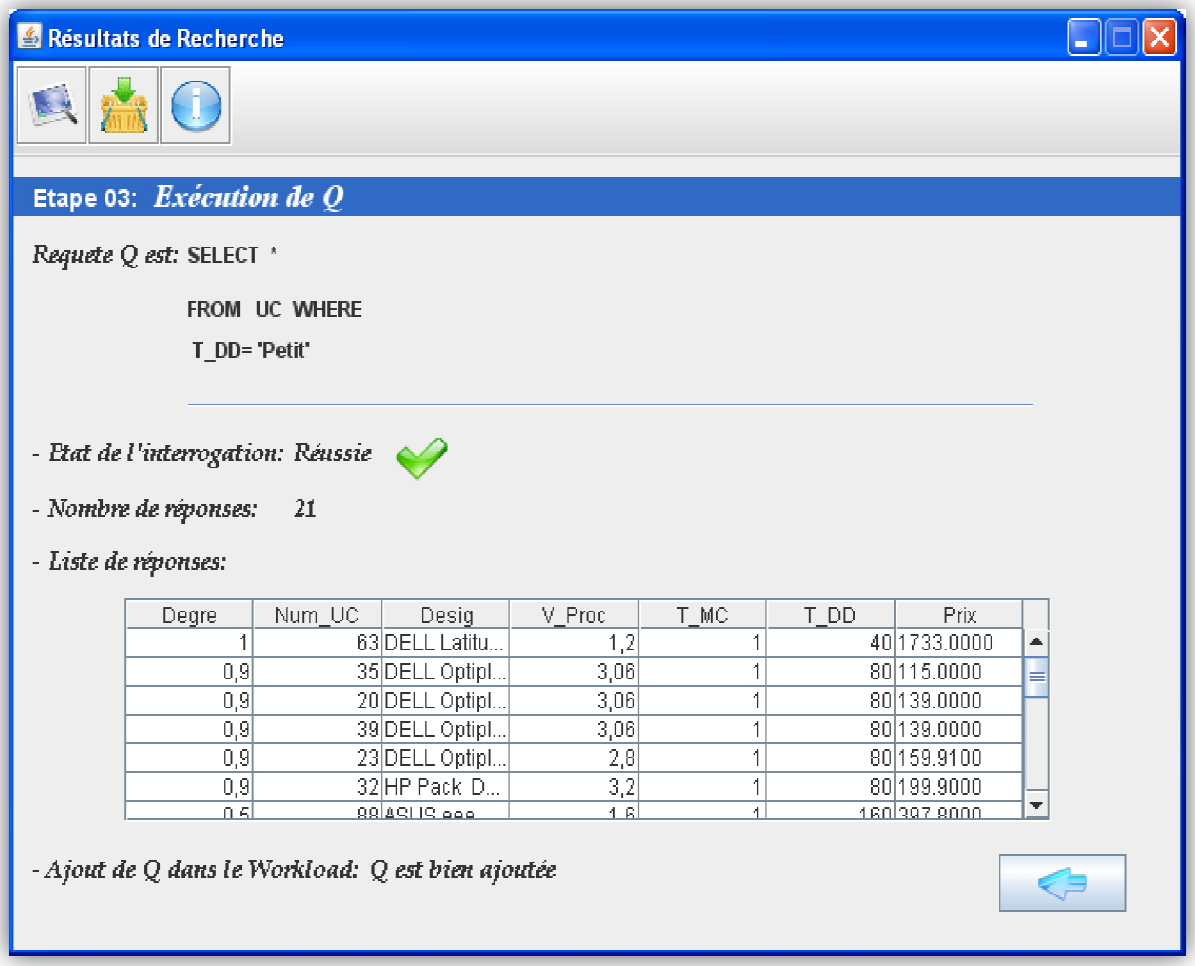


Figure 4.7. Exemple d'une requête atomique

Figure 4.8. Exemple d'une requête composée

Après avoir présenté l'implémentation de l'interrogation flexible de BD, nous présentons dans ce qui suit les deux étapes restantes qui sont : la partie la plus importante de notre travail, où nous concentrons notre effort sur l'implémentation de l'approche que nous avons proposée et la phase de l'évaluation des performances. Dans ce contexte, nous présentons d'abord le traitement des réponses vides pour une requête atomique puis pour une requête composée.

5.1. Expérimentation sur une requête atomique

Nous présentons dans cette sous section, l'implémentation de notre approche pour requêtes atomiques.

Prenons la requête *Q* suivante :"Trouver de préférence les unités centrales de tailles de la mémoire centrale très petite". Cette requête retourne un ensemble vide de réponses. Pour répondre à *Q*, nous appliquons notre module TRV en appuyant sur le bouton « Lancer l'algorithme TRRV », qui affiche par la suite le nombre des requêtes approximatives à Q. Ces

requêtes sont affichées avec leurs nombre de réponses et proximité sémantique qui représentent les critères d'ordonnancement de ces requêtes approximatives (Voir la figure 4.9).

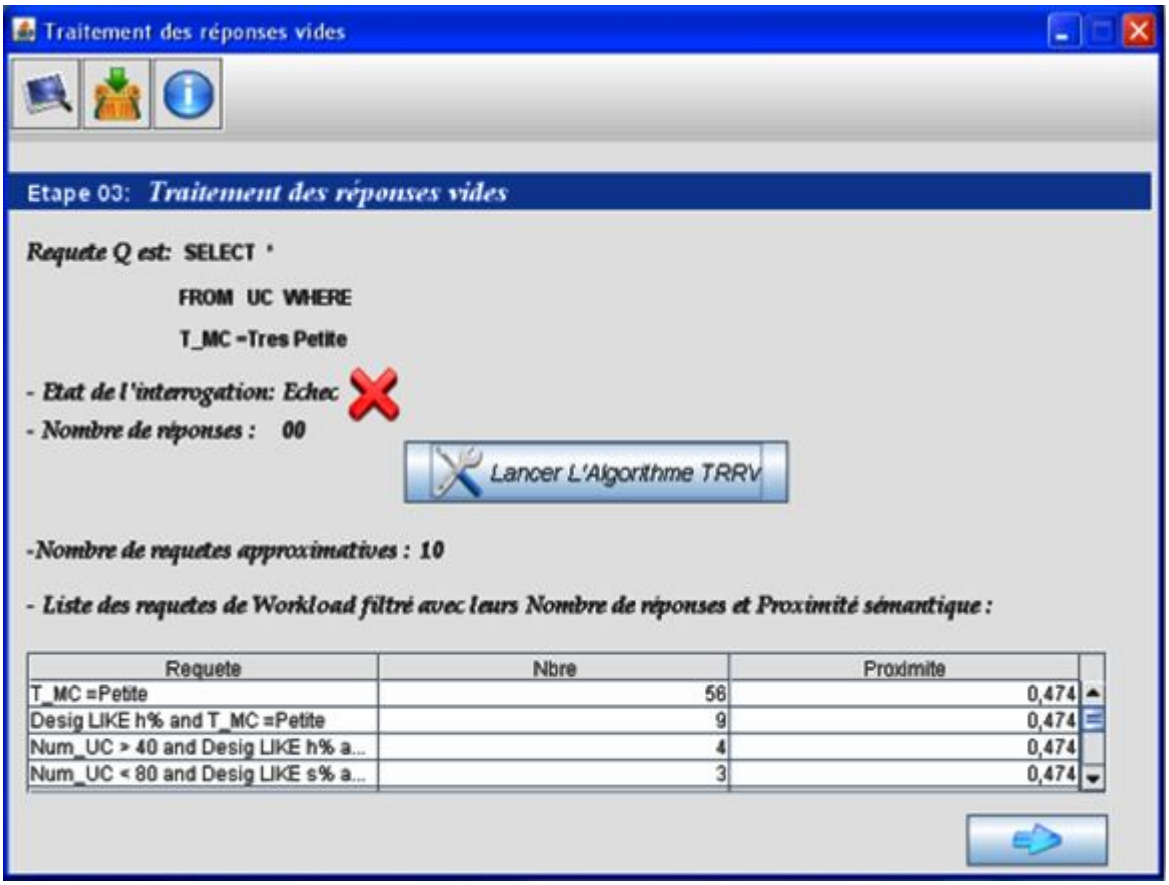
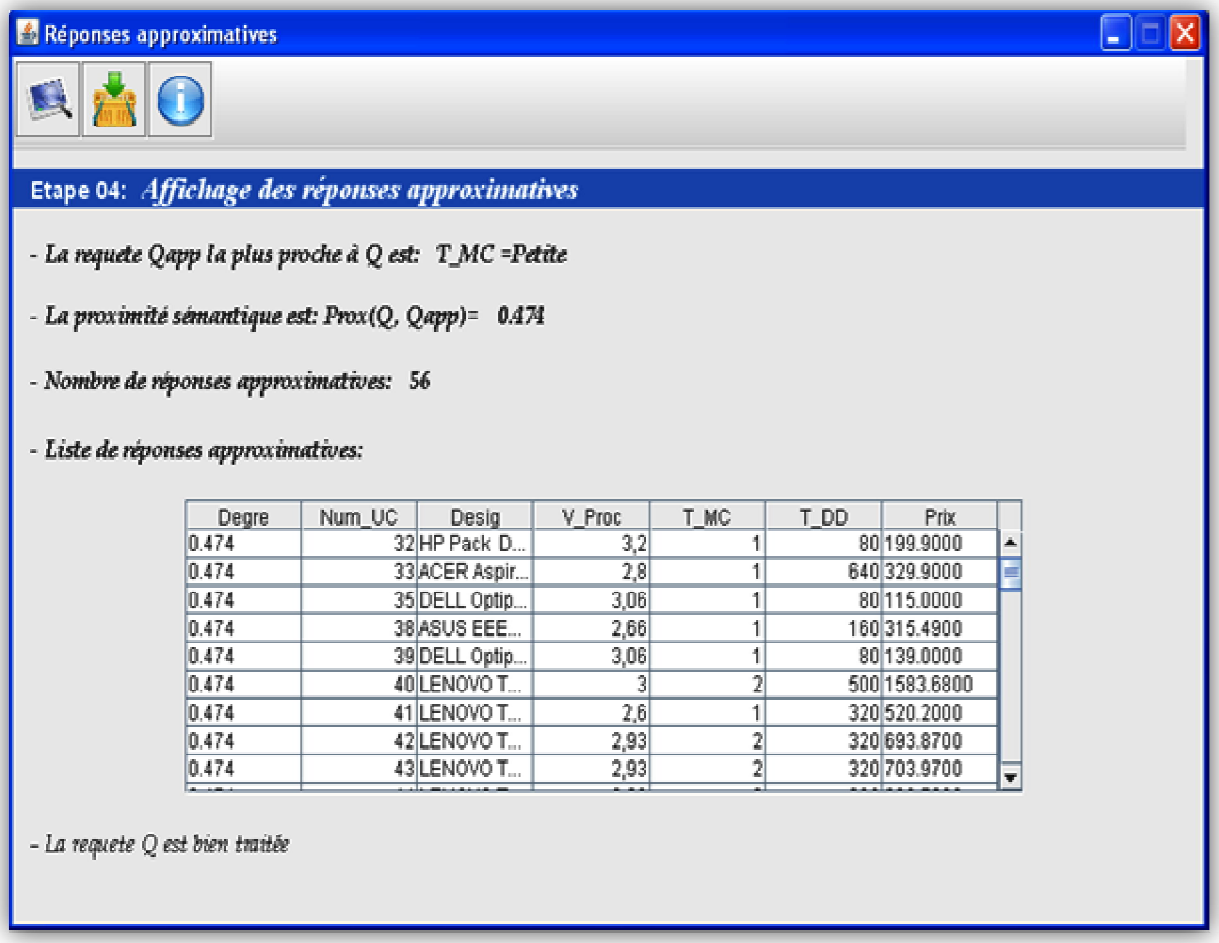


Figure 4.9. Traitement d'une requête atomique

La figure ci-dessous montre la requête Q_{app} la plus proche sémantiquement parlant à la requête Q avec des informations propre à Q_{app} à savoir la requête Q_{app}, la proximité sémantique et le nombre de réponses. Une liste de réponses approximatives à Q sont alors affichée avec un degré d'appartenance (ou de pertinence) unique qui est égale à la valeur de proximité entre Q et Q_{app}.



Degre	Num_UC	Desig	V_Proc	T_MC	T_DD	Prix
0.474	32	HP Pack D...	3,2	1	80	199.9000
0.474	33	ACER Aspir...	2,8	1	640	329.9000
0.474	35	DELL Optip...	3,06	1	80	115.0000
0.474	38	ASUS EEE...	2,66	1	160	315.4900
0.474	39	DELL Optip...	3,06	1	80	139.0000
0.474	40	LENOVO T...	3	2	500	1583.6800
0.474	41	LENOVO T...	2,6	1	320	520.2000
0.474	42	LENOVO T...	2,93	2	320	693.8700
0.474	43	LENOVO T...	2,93	2	320	703.9700

Figure 4.10. Réponses approximatives à une requête atomique

5.2. Expérimentation sur une requête composée

Nous montrons dans cette sous section le traitement d'une requête composée à réponses vides.

Prenons la requête *Q* suivante :"Trouver de préférence les unités centrales moins chère et de tailles de la mémoire centrale très petite". Cette requête retourne un ensemble vide de réponses. Pour répondre à *Q*, nous appliquons notre solution pour les requêtes composées, un ensemble des requêtes approximatives à *Q* sont affichées avec leurs nombre de réponses et proximité sémantique (Voir la figure 4.11).

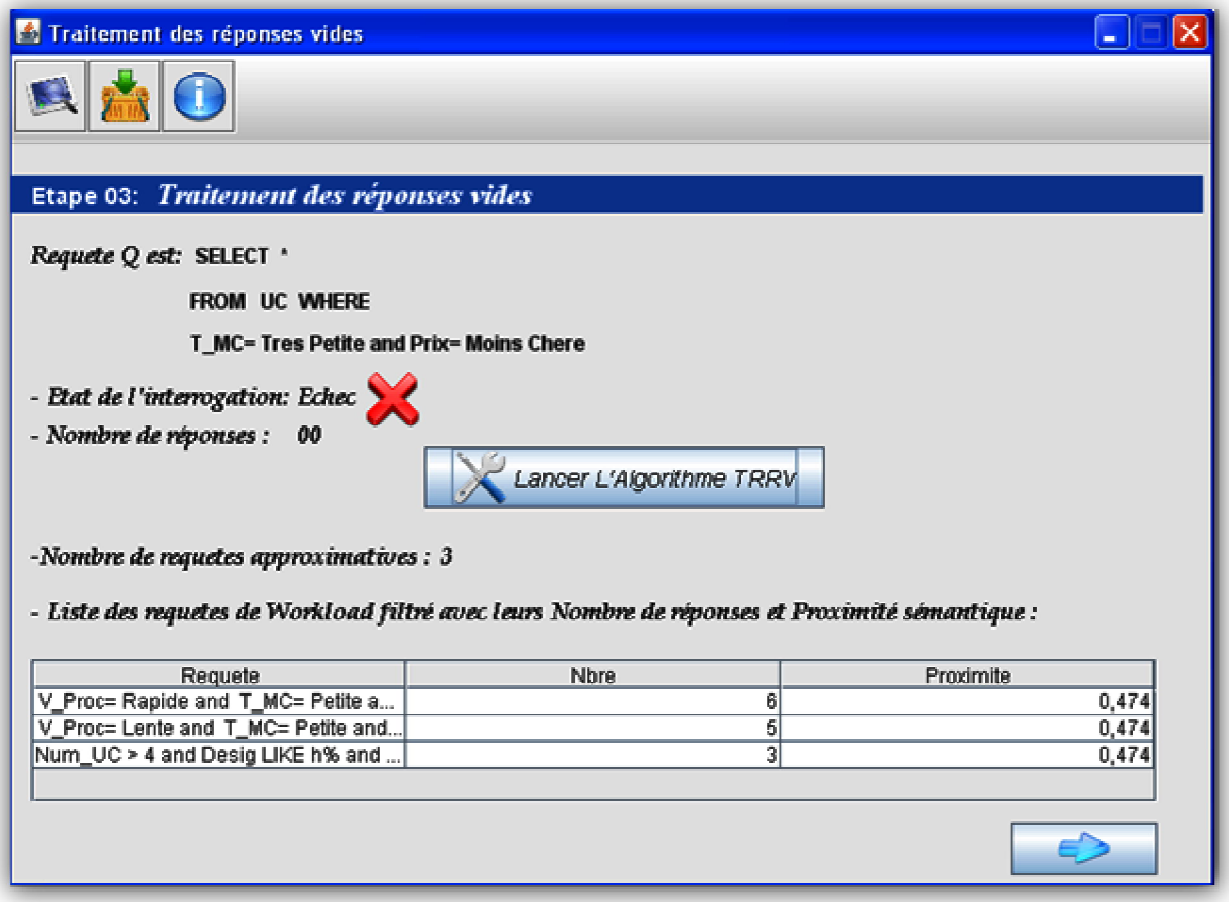
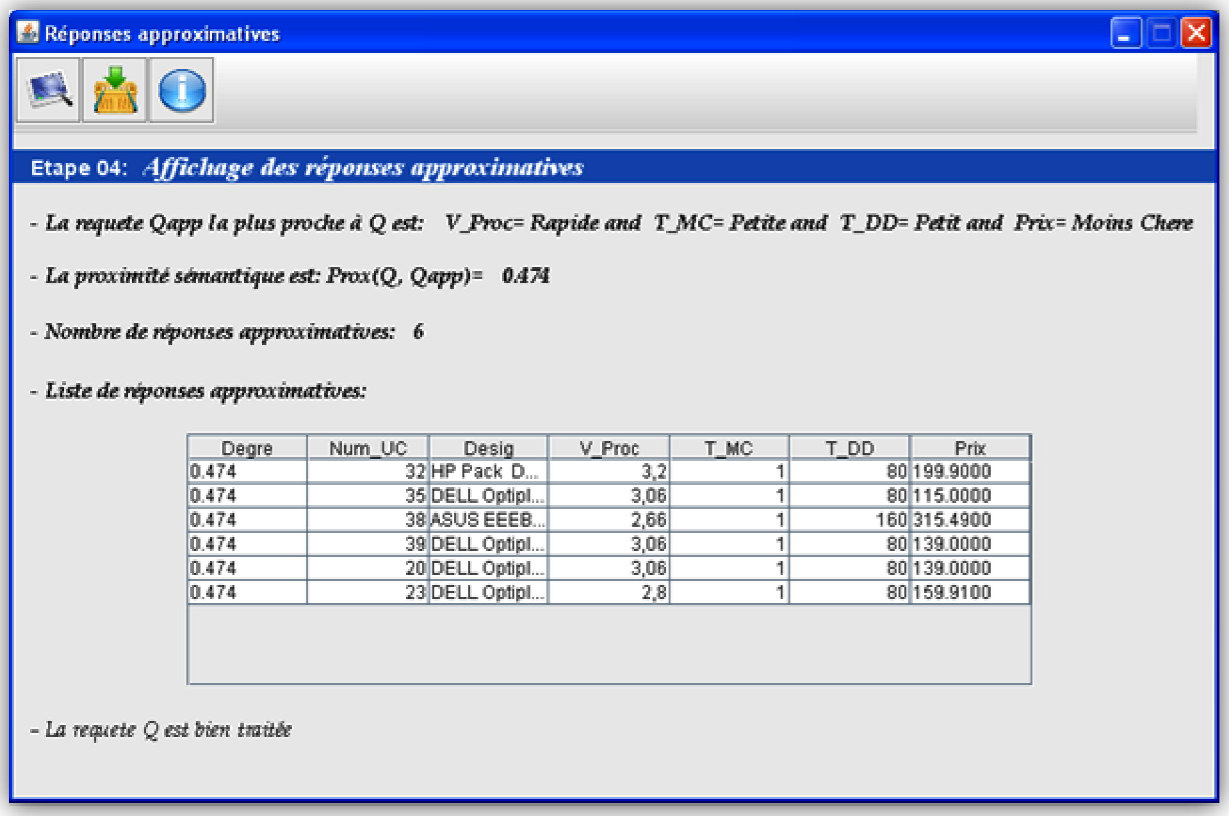


Figure 4.11. Traitement d'une requête composée

Une fois la requête traitée, le module TRV donne la liste des réponses approximatives de la requête la plus proche à la requête *Q* (Voir la figure 4.12).



Degre	Num_UC	Desig	V_Proc	T_MC	T_DD	Prix
0.474	32	HP Pack D...	3,2	1	80	199.9000
0.474	35	DELL Optipl...	3,06	1	80	115.0000
0.474	38	ASUS EEEB...	2,66	1	160	315.4900
0.474	39	DELL Optipl...	3,06	1	80	139.0000
0.474	20	DELL Optipl...	3,06	1	80	139.0000
0.474	23	DELL Optipl...	2,8	1	80	159.9100

Figure 4.12. Réponses approximatives à une requête composée

Pour montrer les bonnes performances et valider notre approche, la sous section suivante présente une évaluation de l'approche.

5.3. Evaluation des performances

Pour évaluer et comparer les systèmes informatiques, on utilise les critères de performance habituels, c'est-à-dire les mesures du temps de réponse et d'espace mémoire utilisé: plus le temps de réponse est court et l'espace occupé par le système est faible, meilleur est le système. Cependant, d'autres mesures de performances ont été introduites, dans le but d'évaluer l'efficacité de système. Parmi elles, nous pouvons citer la facilité d'utilisation du système et la capacité du système à atteindre ses objectifs.

Afin de correctement illustrer le comportement de notre approche, nous mesurons le temps de réponse c.à.d, le temps pour trouver la requête la plus proche à une requête à réponses vides.

Dans ce cadre, nous distinguons deux cas :

1) Le workload est fixe et la requête à réponse vide variée (1, 2, 3, 4 prédicats).

2) La requête est fixe et la taille du workload variée.

Nous commençons par le premier cas, en fixant par exemple la taille du workload à 100 éléments (c.à.d, 100 requêtes flexibles dont les réponses sont non vides). Le tableau suivant contient les résultats.

Nombre de prédicats	Temps moyen
1	8s 20ms
2	6s 14ms
3	1s 20ms
4	1s 55ms

Tableau 4.8. Résultats expérimentaux concernant la variation du temps de traitement de réponses vides

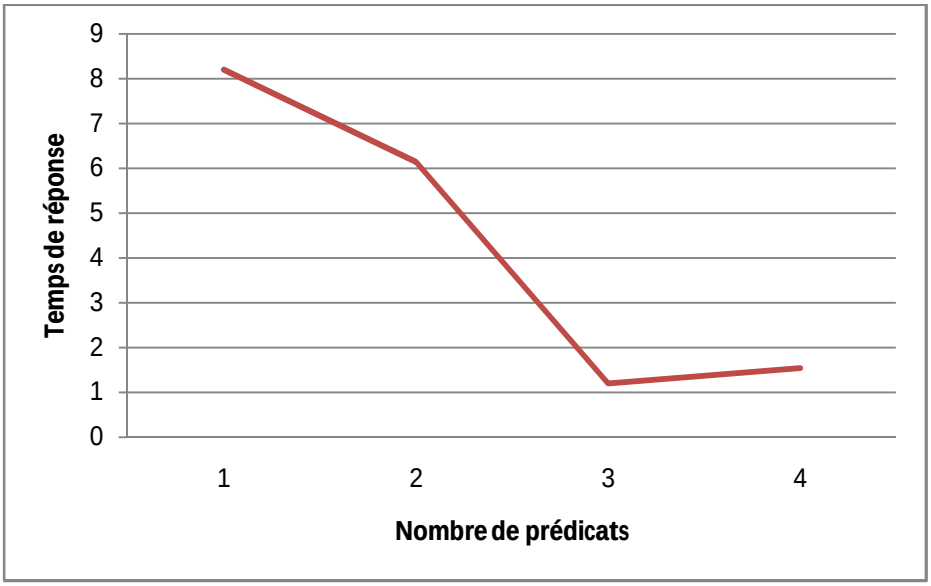


Figure 4.13. Variation du temps de réponse en fonction du nombre de prédicats

A partir des résultats obtenus, nous remarquons que le temps de traitement des réponses vides varié d'un prédicat à d'autre, par exemple pour une requête à un seul prédicat (requête atomique) le temps de réponse est 8s 20ms, alors que pour une requête à 3 prédicats (requête composée) le temps de réponse moyen devient 1s 20ms et pour 2 prédicats on trouve 6s 14ms.

L'analyse de ces résultats montre qu'il y'a d'autre critère plus que le nombre de prédicats, influe sur le temps de réponse. Cet analyse nous a permis à refaire l'évaluation à nouveau afin de dégager le second critère selon lequel le temps de réponse varié. Le tableau suivant contient les résultats.

Nombre de prédicats	**Nombre de requêtes approximatives**	**Temps moyen**
1	15	8s 20ms
2	10	6s 14ms
3	3	1s 20ms
4	3	1s 55ms

Tableau 4.9. Variation du temps de réponse en fonction du nombre de prédicats et du nombre de requêtes approximatives

Dans ce tableau, nous observons que le temps de réponse augmente selon les deux critères suivants: le nombre de prédicats contenants la requête à réponses vides et le nombre de requêtes approximatives à cette requête. Par exemple, pour une requête atomique, le nombre de requêtes approximatives est 15 avec un temps de réponse de 8s 20ms, alors que pour une requête à deux prédicats avec 10 requêtes approximatives, le temps de réponses est diminue et devient 6s 14ms (influence de facteur de nombre de requêtes approximatives). Dans les deux dernières requêtes (à 3 et 4 prédicats respectivement) qui possèdent le même nombre de requêtes approximatives (3 requêtes approximatives), le temps de réponse est varié selon la requête qui possède le plus grand nombre de prédicats, et donc la requête à 4 prédicats prend beaucoup temps (1s 55ms) par rapport à la requête contenant 3 prédicats (influence de facteur de nombre de prédicats).

Dans le deuxième cas où la taille du workload varie et la requête à réponse vide est fixe. On prend comme requête à réponses vides, la requête à deux prédicats suivante :"Trouver de préférence les unités centrales rapides et de tailles de la mémoire centrale très petite". Le tableau suivant contient les résultats.

Taille du Workload	Temps de réponse
10	2s 20ms
50	4s 16ms
100	11s 82ms
200	20s 75ms
500	55s 32ms
1000	1m 51s 4ms
1500	2m 56s 36ms
10000	3m 6s 28ms

Tableau 4.10. Résultats expérimentaux concernant la variation du temps de réponse en fonction de la taille du workload

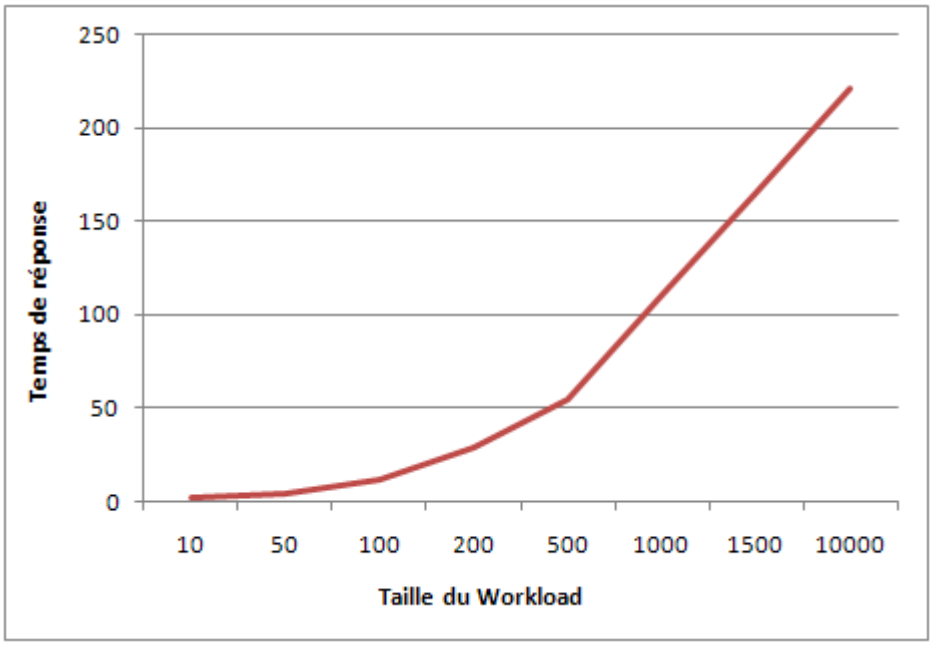


Figure 4.14. Variation du temps de réponse en fonction de la taille du workload

Nous pouvons observer que le temps de réponse augmente en fonction de la taille du workload ; plus la taille du workload est grande, plus le temps de réponse est long.

A partir de l'analyse effectuée dans ces deux cas, nous pouvons conclure que lorsqu'on fixe la taille du workload et on change la requête (modifier le nombre de prédicats), le temps de réponse augmente en fonction du nombre de prédicats et du nombre de requêtes

approximatives, alors que lorsqu'on fixe la requête (par exemple, 2 prédicats) et on modifie la taille du workload, le temps de réponse augmente en fonction de la taille du workload.

Concernant d'autres critères d'efficacité, notre approche offre une interface facile à utilisée ; l'utilisateur n'a qu'à sélectionner les attributs et les prédicats qui lui convient pour formuler ses requêtes, donc, le système limite les risques d'erreurs de la part de l'utilisateur. Ainsi, la capacité du système à fournir des réponses répondant aux besoins de l'utilisateur (préférences) et surtout la capacité du système à atteindre notre but de ce mémoire de magister qui est le traitement des requêtes flexibles à réponses vides.

Les résultats des expérimentations montrent que la prise en compte d'une proximité sémantique améliorait notablement le taux de succès des interrogations flexibles dans une BD et le temps d'exécution des requêtes.

L'utilisation de la distance de Hausdorff comme outil pour mesurer la proximité sémantique entre requêtes, améliore le temps d'exécution et donne un meilleur résultat par rapport aux autres approches (utilisation de distance de Jaccard, distance euclidienne,…etc.) à cause de ses propriétés intéressantes à savoir qu'elle représente une métrique et elle permet d'estimer à quel point deux ensembles se ressemblent, de plus, sur le plan calculatoire cette mesure se réduit toujours au calcul de la distance entre intervalles classiques.

En conséquence, nous déduisons que notre solution pour le traitement de PRV est plus performante et efficace par rapport aux autres approches guidées par un Workoload of past queries.

6. CONCLUSION

Dans ce chapitre, nous avons présenté la mise en œuvre et l'implémentation de l'approche que nous avons proposée dans le chapitre précédent, pour le traitement des requêtes flexibles à réponses vides. Nous avons montré aussi, l'implémentation de l'interrogation flexible de BD et les résultats des expérimentations effectuées pour évaluer les performances de notre approche.

Les résultats montrent que notre approche offre plusieurs avantages par rapport aux autres approches guidées par un "Workload of past queries", à savoir qu'elle limite dans une large mesure le risque d'obtention d'une réponse vide et offre à l'utilisateur des informations

sur le degré de similarité entre une requête à réponse vide et une requête de Workload la plus proche sémantiquement parlant à cette requête, ainsi qu'elle améliore le temps d'exécution d'une requête et le temps de traitement des réponses vides.

En conséquence, elle nous semble pertinente d'utiliser notre approche dans les domaines qui nécessitent l'interrogation et l'exploitation intelligente des données tels que : le Web sémantique, Systèmes d'information, Technologies WEB et le Data Mining.

Conclusion Générale

Le travail présenté dans ce mémoire se situe dans le contexte de traitement des requêtes flexibles à réponses vides.
Une interrogation flexible des données au moyen de la théorie des sous ensembles flous, permet de limiter dans une large mesure le risque d'obtention d'une réponse vide c'est-à-dire il n'existe aucun item de la base de données interrogée qui satisfait un tant soit peu la requête de l'utilisateur. Cependant, ce risque n'est pas totalement éliminé, et plusieurs approches ont été proposées dans la littérature pour traiter ce problème. La plupart de ces approches sont basées sur le mécanisme de relaxation qui agit seulement sur les conditions impliquées dans la requête qui a échoué.

Notre travail alors, consiste à proposer une approche pour le traitement des requêtes à réponses vides ; son principe est de proposer des réponses approximatives à la requête à réponse vide (échouée) en exploitant un workload des requêtes précédemment exécutées par le système et dont les réponses sont non vides.
Les réponses approximatives à la requête échouée sont les réponses de la requête du workload, la plus proche sémantiquement parlant à cette requête échouée, en estimant pour celles-ci la proximité sémantique qui représente le degré d'appartenance de ces réponses approximatives. Cette mesure de proximité sémantique utilise une distance particulière dite la distance de Hausdorff appliquée sur les prédicats contenants ces requêtes. Ainsi, qu'un algorithme de recherche (de la requête la plus proche) a été développé et son implémentation est complètement réalisée.

Les résultats des expérimentations montrent que la prise en compte d'une proximité sémantique améliorait notablement le taux de succès des recherches dans une BD et le temps d'exécution des requêtes. En conséquence, elle nous semble pertinente d'utiliser notre approche dans les domaines qui nécessitent l'interrogation et l'exploitation intelligente des données tels que : le Web sémantique, la recherche d'information et le Data Mining.

Ce travail est réalisé après une étude faite sur les différentes approches proposées pour traiter les requêtes à réponses vides. Nous avons représenté une nouvelle approche fondée sur une mesure de proximité sémantique entre requêtes qui donne un meilleur résultat par rapport

aux autres approches. Néanmoins, notre approche ouvre des perspectives futures. A court terme, les plus importantes sont les suivantes :

→ Appliquer notre approche sur plusieurs tables relationnelles, dans divers domaines de connaissances et sous diverses formes (numériques, textes, images, vidéo,… etc).

→ Étendre l'approche à des attributs à des domaines non métrique (comme attribut de couleur).

→ Améliorer l'application en utilisant des bases de données réparties et distribuées à travers un réseau local ou via Internet.

→ Utiliser notre approche dans l'interrogation d'une base de données très importante dont on ne connait pas précisément le contenu comme le Web, afin d'éviter de reformuler plusieurs fois des requêtes et assurer leurs succès.

RÉFÉRENCES BIBLIOGRAPHIQUES

[1] G. Salton and M. McGill: *Introduction to modern information retrieval*. McGraw-Hill Int. 1984.

[2] H. Tebri. *Formalisation et spécification d'un système de filtrage incrémental d'information*. Thèse de doctorat, Toulouse : Université Paul Sabatier, 2004.

[3] Claude Delobel et Michel Adiba : *Bases de données et systèmes relationnels*. Dunod. 1982.

[4] Date, Ch: *Introduction to Database Systems*. Addison-Wesley .2004.

[5] Ullman J.D: *Principles of Database and Knowledge-Base Systems*, Computer Science. 1989

[6] P. Bosc, A. Hadjali, O. Pivert:, *Empty versus overabundant answers to flexible queries*, Fuzzy Sets and Systems Journal, Vol. 159 (12), pp. 1450-1467. 2008

[7] VOM HOFE Alain: *Effective uses and usefulness judments of web boolean querying*, Revue d'intelligence artificielle, vol. 14 (243.), pp. 113-126. 2000

[8] Daniel Rocacher, Ludovic Liétard : *Préférences et quantités dans le cadre de l'interrogation flexible : sur la prise en compte d'expressions quantifiées.* IRISA/ENSSAT, IRISA/IUT Lannion

[9] L.A.Zadeh: *The concept of linguistic variable and it application to approximate reasoning.* Information Science, Vol. 08 , pp. 199-249.1975

[10] Patrick Bosc, Ludovic Liétard and Henri Prade: *An Ordinal Approach to the Processing of Fuzzy Queries with Flexible Quantifiers.* Applications of Uncertainty Formalisms, LNAI 1455, pp. 58-75. 1998

[11] Motro A., FLEX : *Tolerant and Cooperative User Interface to Database*, IEEE Transactions on Knowledge and Data Engineering, vol. 4, pp. 231-246. 1990

[12] Ounelli H., Hachani N: *Pragmatic Approach for Flexible Querying of SQL Data Base*, ADBIS, Dresden, Germany. 2003

[13] JAN CHOMICKI: *Preference Formulas in Relational Queries*. ACM Transactions on Database Systems, Vol. 28, pp. 1–40. 2003

[14] L. Ludovic, R. Daniel, and S-E Tbahriti: *Towards an Extended SQLf: Bipolar Query Language with Preferences*. World Academy of Science, Engineering and Technology 26. 2007

[15] P. Bosc, O. Pivert: *On the evaluation of simple fuzzy relational queries: principles and Measures*, Kluwer Academic Publishers, pp. 355-364. 1993

[16] Patrick Bosc, Allel Hadjali, Olivier Pivert: *Une Approche Coopérative pour le Traitement des Requêtes Flexibles à Réponse Vide ou Pléthorique.* Fuzzy Sets and Systems Journal. 2008

[17] Parke Godfrey: *Minimization in cooperative response to failing database queries.* Army Research Laboratory (USA). 1994

[18] TERRY GAASTERLAND, PARKE GODFREY, JACK MINKER: *Relaxation as a Platform for Cooperative Answering*. Journal of Intelligent Information Systems, Vol. 1, pp. 293-321. 1992

[19] Chu, W. W., Chen, Q: *A structured approach for cooperative query answering*. IEEE Transactions on Knowledge and Data Engineering, Vol. 6, pp. 738–749. 1994

[20] Muslea.I: *Machine learning for online query relaxation*. ACM SIGKDD International Conference on Knowledge Discovery and Datamining, pp. 246–255. 2004

[21] P. Bosc, A. Hadjali, O. Pivert*: Incremental controlled relaxation of failing flexible queries*, Journal of Intelligent Information Systems, Vol. 3, pp. 261-283. 2009

[22] V. Raghavan, H. Sever: *On the Reuse of Past Optimal Queries*. ACM SIGIR, pp. 344-350. 1995

[23] A. Motro: *VAGUE: A user interface to relational databases that permits vague queries.* ACM Transactions on Office Information Systems, Vol. 6, pp. 187-214. 1988

[24] G.Amato, F.Rabitti, P.Savino, P.Zuzela: *Approximate similarity search in metric data by using region proximity.*

[25] Nambiar, U., Kambhampati, S: *Answering imprecise database queries: A novel approach*. ACM Workshop on Web Information and Data Management (WIDM). 2003

[26] A. Bidault, C. Froidevaux, B. Safar: *Similarity Between Queries in a Mediator.* ECAI, pp. 235-239. 2002

[27] O.R. Zaïane, A. Strilets: *Finding Similar Queries to Satisfy Searches Based on Query Traces*. OOIS Workshops, pp. 207-216. 2002

[28] S. Acharya, P. B. Gibbons and V. Poosala: *Aqua: A Fast Decision Support System Using Approximate Query Answers*. Demo Abstract in VLDB. 1999

[29] Y. E. Ioannidis and V. Poosala: *Histogram-Based Approximation of Set-Valued Query Answer*s. Very Large Data Bases. 1999

[30] K. Chakrabarti, M.N. Garofalakis, R. Rastogi, K. Shim: *Approximate Query Processing Using Wavelets*. VLDB, pp. 111-122. 2000

[31] P. Bosc, C. Brando, A. Hadjali, H. Jaudoin, O. Pivert: *Semantic proximity between queries and the empty answer problem*. IFSA. 2009

[32] V. Cross, T. Sudkamp: *Similarity and Compatibility in Fuzzy Set Theory: Assessment and Applications*. Studies in Fuzziness and Soft Computing, N°. 93. 2002

[33] D.W AHA, D KIBLER, M.K ALBERT. *Instance-Based Learning Algorithms.* Machine Learning journal. Vol. 6, pp. 37-66. 1991

[34] T. MITCHELL. *Generalization as Search*. Artificial Intelligence Journal. Vol. 18, pp. 203-226. 1982

[35] G. Hirst, D. Onge. *Lexical chains as representations of context for the detection and correction of malapropisms*. MIT Press .1998

[36] I. Bloch. *On fuzzy distances and their use in image processing under imprecision*. Pattern Recognition, Vol. 32, pp. 1873-1895. 1999

[37] Zsolt Csaba Johanyák, Szilveszter Kovács. *Distance based similarity measures of fuzzy sets*. SAMI. 2005

[38] Sang H. Lee, Hyunjeong Park, Wook Je Park. *Similarity Computation between Fuzzy Set and Crisp Set with Similarity Measure Based on Distance.* Computer Science, Vol. 4993, pp 644-649. 2008

[39] Ting-YuChen. *A note on distances between intuitionistic fuzzy sets and/or interval-valued fuzzy sets based on the Hausdorff metric*. Fuzzy sets and systems, Vol. 158, N°. 22, PP. 2523-2525. 2007

[40] B.B. Chaudhuri, A. Rosenfeld. *A modified Hausdorff distance between fuzzy sets*. Information Sciences, Vol. 118, pp. 159-171. 1999

[41] D. Dubois, H. Prade. *On distances between fuzzy points and their use for plausible reasoning*. Systems, Man and Cybernetics, pp. 300-303. 1983

[42] J. Kacprzyk, S. Zadrozny. *Computing with words in intelligent data base querying: standalone and Internet-based applications*. Information Sciences, Vol. 134, pp. 71–109. 2001

[43] Voxman, W. *Some remarks on distances between fuzzy numbers*. Fuzzy Sets and Systems, Vol. 100, pp. 353–365. 1998

[44] Chandan Chakraborty, Debjani Chakraborty. *A theoretical development on a fuzzy distance measure for fuzzy numbers.* Mathematical and Computer Modelling, Vol. 43, pp. 254–261. 2006

[45] Saeid Abbasbandy, Saeide Hajighasemi. *A Fuzzy Distance between Two Fuzzy Numbers.* Communications in Computer and Information Science, Vol. 81, pp. 376–382. 2010

[46] Delgado, M., Vila, M.A., Voxman, W. *On a canonical representation of fuzzy numbers.* Fuzzy Sets and Systems, Vol. 93, pp. 125–135. 1998

[47] Delgado, M., Vila, M.A., Voxman, W. *A fuzziness measure for fuzzy numbers: Applications.* Fuzzy Sets and Systems, Vol. 94, pp. 205–216. 1998

[48] O. Arieli, M. Denecker, M. Bruynooghe. *Distance semantics for database repair*. Math. Artif. Intell, Vol. 50, pp. 389–415. 2007.

[49] M.L. Puri, D.A. Ralescu. *Differentials of fuzzy functions.* Journal of Mathematical Analysis and Applications, Vol. 91, pp. 552-558. 1983.

[50] Sang H. Lee, Hyunjeong Park, and Wook Je Park. *Similarity Computation between Fuzzy Set and Crisp Set with Similarity Measure Based on Distance.* Lecture Notes in Computer Science, Vol. 4993, pp. 644-649. 2008.

[51] Aggoune aicha, Bouramoul abdelkrim, Khababa abdallah. *Modélisation des Interfaces Homme-Machine Adaptatives*. ICAI, vol. 1, pp. 274-281. 2009

[52] Aggoune aicha. *An Approach Based Design Patterns for Modeling of Adaptive Human-Machine Interfaces*. MSISI, 2010

[53] Blanchard E., Harzallah M., Briand H., Kuntz, P. *A Typology Of Ontology-Based Semantic Measures.* Workshop EMOI-INTEROP at CAISE'05. 2005

[54] Nicolas BÉCHET. *Extraction et regroupement de descripteurs morpho-syntaxiques pour des processus de Fouille de Textes.* Thèse de doctorat, Montpellier: Université Montpellier II. 2009

[55] AICHA Aggoune. *A Reusable Modeling for Adaptable HMI.* ICIST'2011, 2011.

ANNEXES

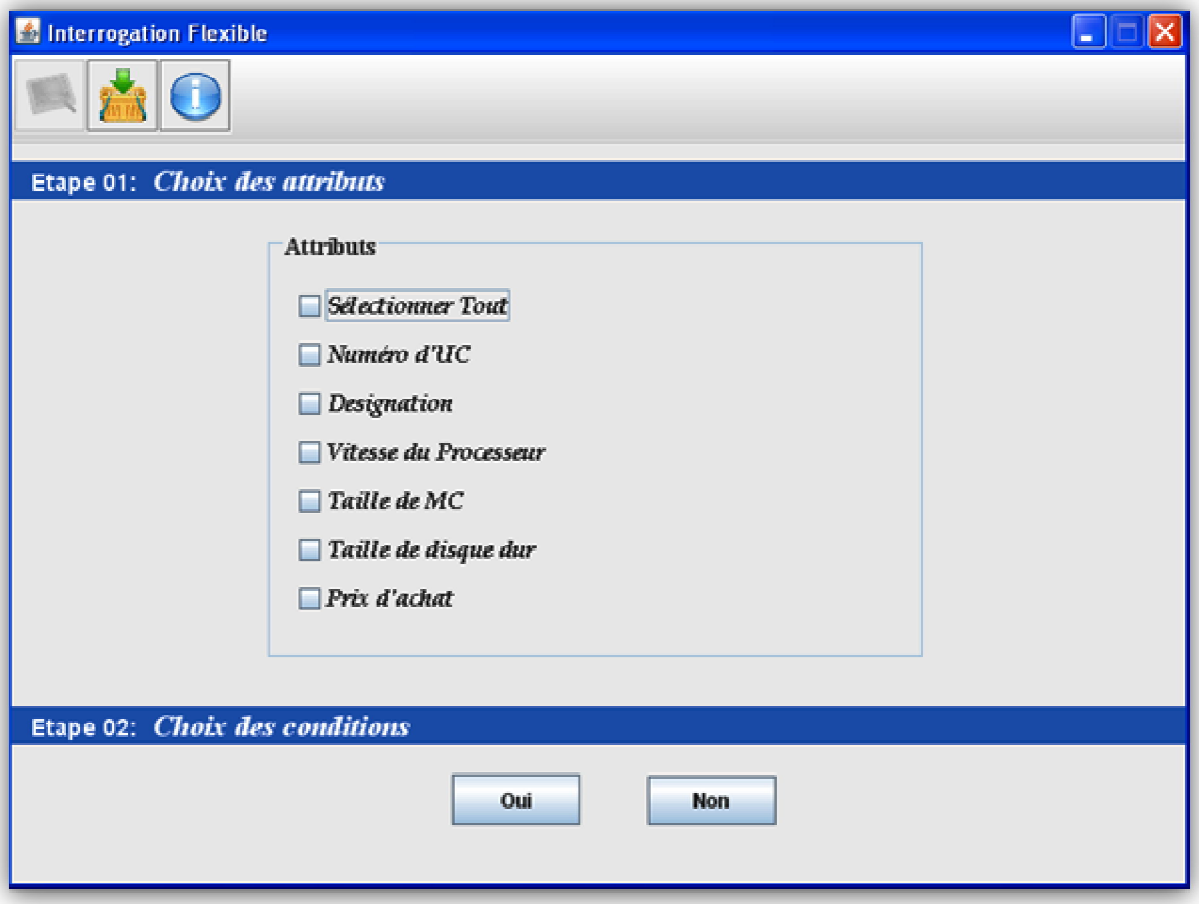
Annexe 01 : ***Présentation de l'application***

Dans cette première annexe, nous présentons, comment fonctionne notre application lors d'une interrogation flexible et plus précisément lors de traitement du problème des réponses vides.

Pour exécuter l'application, il suffit comme pour n'importe quel logiciel, une splash screen ou écran de démarrage s'affiche durant le lancement de l'application.

Ecran de démarrage de l'application

L'accès à notre application se fera à travers l'interface d'accueille, qui permet aux utilisateurs de réaliser leurs interrogations.



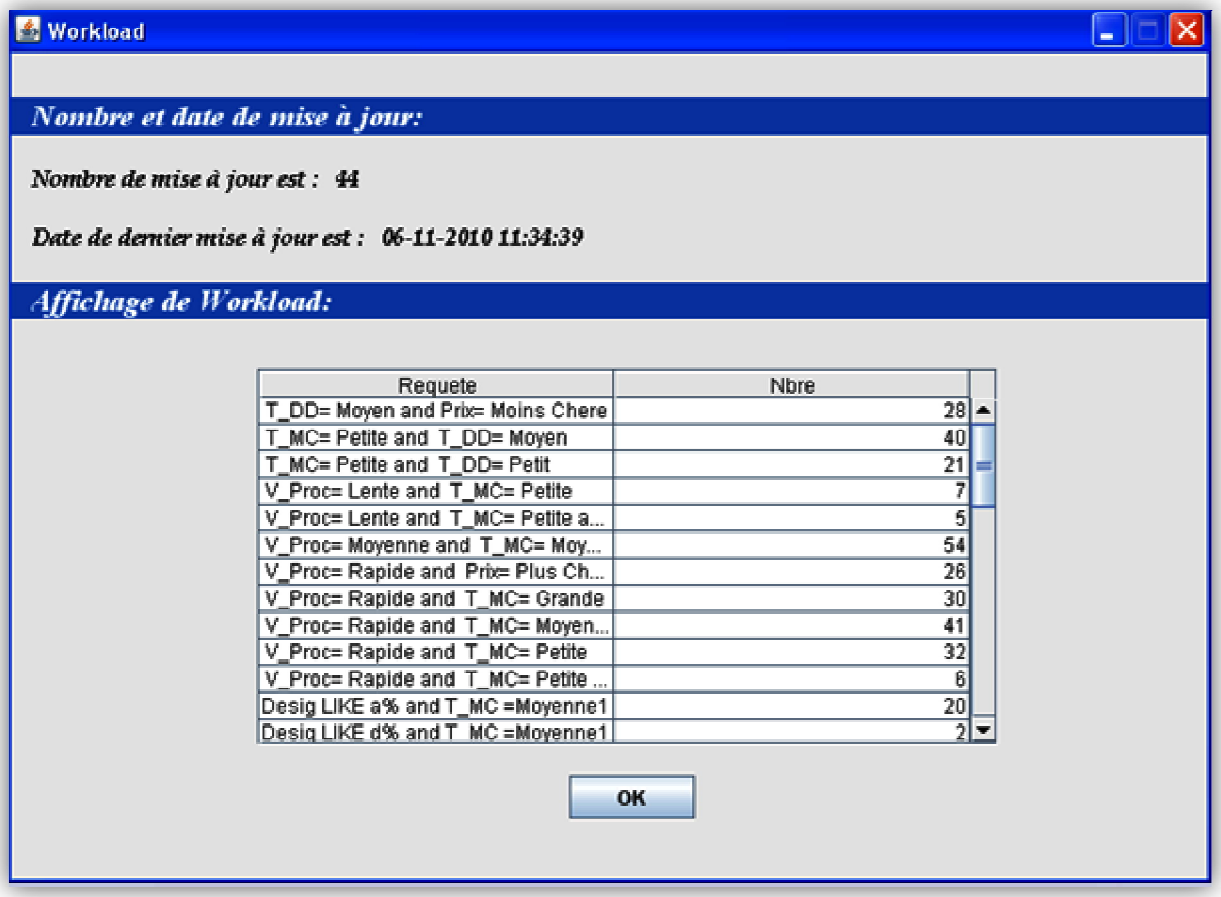
Interface d'accueille

Cette interface dispose un menu contient les items images suivants :

-  *(Nouvelle requêtes) :* pour exécuter une nouvelle requête, au début il est désactivé puisque nous somme dans la forme de réalisation de l'interrogation, qui consiste d'abord à sélectionner les attributs de la requête utilisateur.
-  *(Workload) :* contient l'ensemble des requêtes précédemment exécutées par le système et dont les réponses sont non vides.
-  *(A-propos) :* un résumé sur l'application.

Cette interface permet à l'utilisateur de réaliser son interrogation à l'aide des composants graphiques afin d'être aidé dans l'élaboration de sa requête.

La fenêtre suivante permet aux utilisateurs à voir le Workload des requêtes.



Workload de l'application

La figure suivante représente l'interface, lors de sélection de l'item *A-propos :*

A-propos de l'application

Notre table SQL est crée à l'aide de SGBD Access 2007, comme illustré dans la figure suivante :

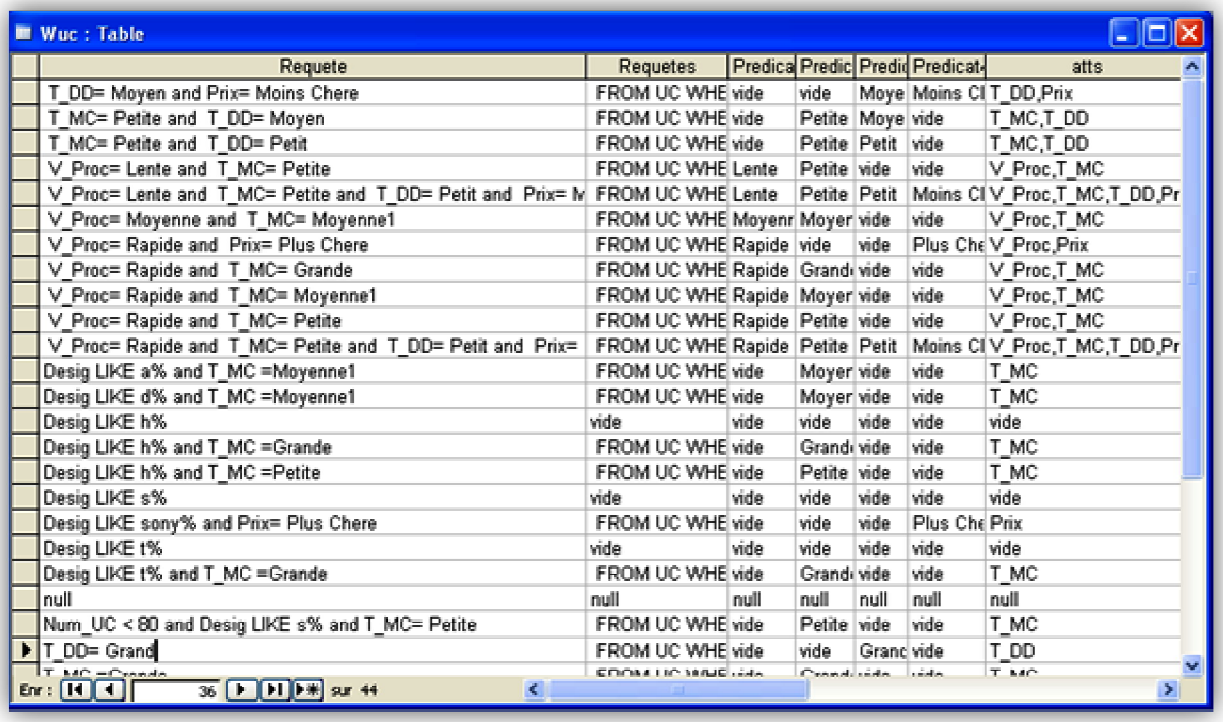
UC : Table

Num_UC	Desig	V_Proc	T_MC	T_DD	Prix
1	PACKARD Bell iMedia i4573FR	2,93	4	1000	469,90 €
2	COMPACK Presario CQ5325FR	2,7	3	500	287,90 €
3	HP Pavilion p6340FR	2,7	4	1000	439,00 €
4	ACER Aspire AZ5610-UF9p	2,66	4	750	718,50 €
5	COMPACK Presario CQ5315M	2,66	2	320	349,00 €
6	HP Pavilion p6356F-mR	2,66	4	1000	585,90 €
7	HP Pavilion Elite HPE-110fr	3,2	6	1000	772,90 €
8	ASUS EEEBox EB1501	2,66	2	320	377,49 €
9	HP Pavilion p6346FR-m	2,7	4	640	489,00 €
10	HP Pavilion p6345FR	2,93	4	750	489,00 €
11	HP TouchSmart 600-1140FR	2,13	4	1000	1 132,90 €
12	ACER Aspire G7711-8WFB	2,6	6	1000	1 473,00 €
13	SONY Vaio VPC-L12M1E/S	2,93	3	1000	1 182,00 €
14	SERANO D5000	2,2	2	500	309,00 €
15	HP Compaq 6005 Pro	3	2	250	999,90 €
16	HP WorkStation Wx4600	3	2	250	1 299,87 €
17	HP Compaq 505b	2,7	2	320	720,13 €
18	HP Compaq 8000 Elite	3	2	160	1 303,91 €
19	HP Elite	2,66	4	500	1 186,75 €
20	DELL Optiplex	3,06	1	80	139,00 €
21	ACER Aspire Revo R3610-020	1,6	2	250	319,90 €
22	HP Pavilion Elite HPE-130fr	3,2	8	2000	1 189,00 €
23	DELL Optiplex Gx270	2,8	1	80	159,91 €

Enr : 1 sur 100

La table SQL de la BD "UC.mdb"

Le Workload des requêtes est également stocké dans une autre table SQL nommée Wuc suivante :



Wuc : Table

Requete	Requetes	Predica	Predic	Predi	Predicat	atts
T_DD= Moyen and Prix= Moins Chere	FROM UC WHE	vide	vide	Moye	Moins Cl	T_DD,Prix
T_MC= Petite and T_DD= Moyen	FROM UC WHE	vide	Petite	Moye	vide	T_MC,T_DD
T_MC= Petite and T_DD= Petit	FROM UC WHE	vide	Petite	Petit	vide	T_MC,T_DD
V_Proc= Lente and T_MC= Petite	FROM UC WHE	Lente	Petite	vide	vide	V_Proc,T_MC
V_Proc= Lente and T_MC= Petite and T_DD= Petit and Prix= M	FROM UC WHE	Lente	Petite	Petit	Moins Cl	V_Proc,T_MC,T_DD,Pr
V_Proc= Moyenne and T_MC= Moyenne1	FROM UC WHE	Moyenr	Moyer	vide	vide	V_Proc,T_MC
V_Proc= Rapide and Prix= Plus Chere	FROM UC WHE	Rapide	vide	vide	Plus Che	V_Proc,Prix
V_Proc= Rapide and T_MC= Grande	FROM UC WHE	Rapide	Grand	vide	vide	V_Proc,T_MC
V_Proc= Rapide and T_MC= Moyenne1	FROM UC WHE	Rapide	Moyer	vide	vide	V_Proc,T_MC
V_Proc= Rapide and T_MC= Petite	FROM UC WHE	Rapide	Petite	vide	vide	V_Proc,T_MC
V_Proc= Rapide and T_MC= Petite and T_DD= Petit and Prix=	FROM UC WHE	Rapide	Petite	Petit	Moins Cl	V_Proc,T_MC,T_DD,Pr
Desig LIKE a% and T_MC =Moyenne1	FROM UC WHE	vide	Moyer	vide	vide	T_MC
Desig LIKE d% and T_MC =Moyenne1	FROM UC WHE	vide	Moyer	vide	vide	T_MC
Desig LIKE h%	vide	vide	vide	vide	vide	vide
Desig LIKE h% and T_MC =Grande	FROM UC WHE	vide	Grand	vide	vide	T_MC
Desig LIKE h% and T_MC =Petite	FROM UC WHE	vide	Petite	vide	vide	T_MC
Desig LIKE s%	vide	vide	vide	vide	vide	vide
Desig LIKE sony% and Prix= Plus Chere	FROM UC WHE	vide	vide	vide	Plus Che	Prix
Desig LIKE t%	vide	vide	vide	vide	vide	vide
Desig LIKE t% and T_MC =Grande	FROM UC WHE	vide	Grand	vide	vide	T_MC
null	null	null	null	null	null	null
Num_UC < 80 and Desig LIKE s% and T_MC= Petite	FROM UC WHE	vide	Petite	vide	vide	T_MC
T_DD= Grand	FROM UC WHE	vide	vide	Granc	vide	T_DD

Enr : 36 sur 44

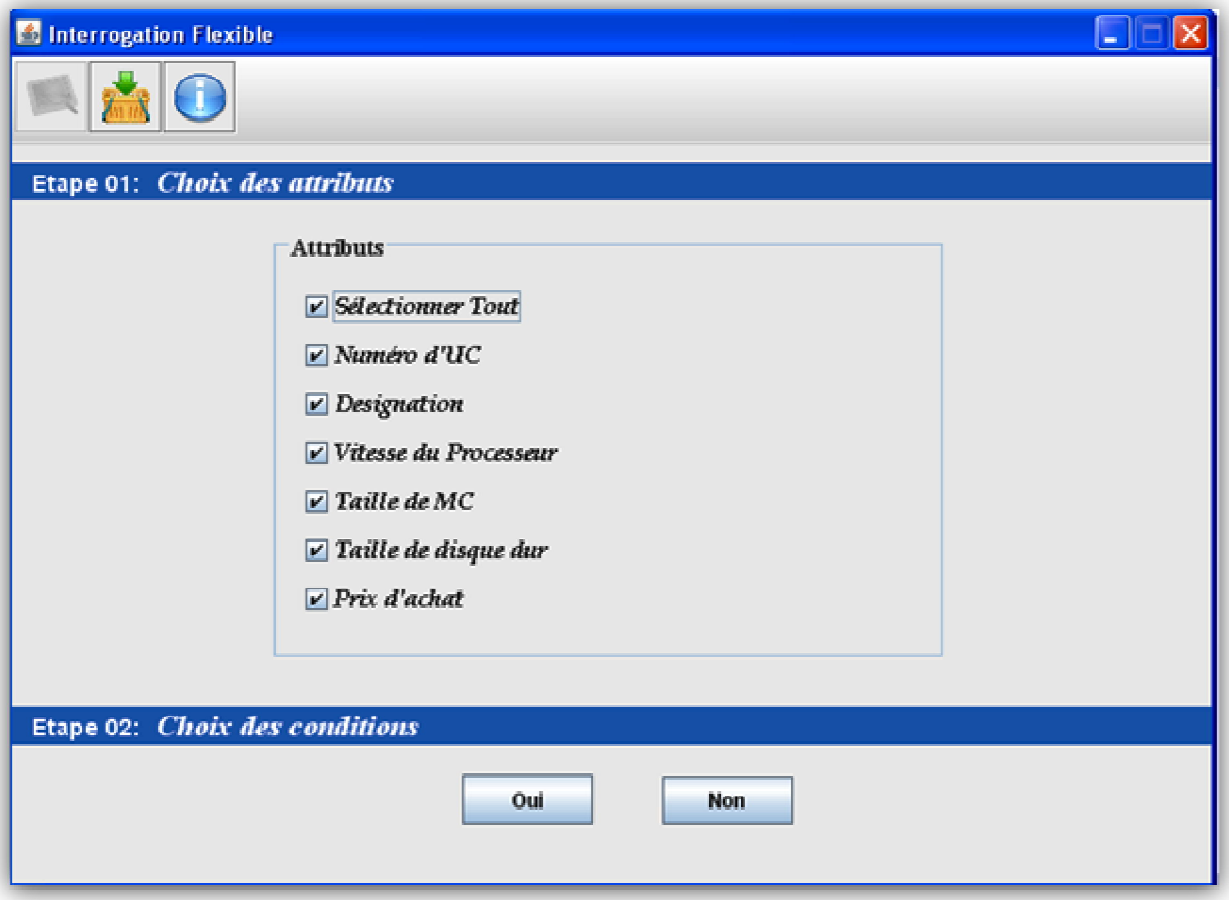
La table SQL de Workload " Wuc "

L'interrogation flexible de BD est réalisée en trois étapes suivantes :

Etape 01 "*Choix des attributs" :* permet aux utilisateurs de sélectionner les attributs qu'il souhaite voir afficher dans la réponse.

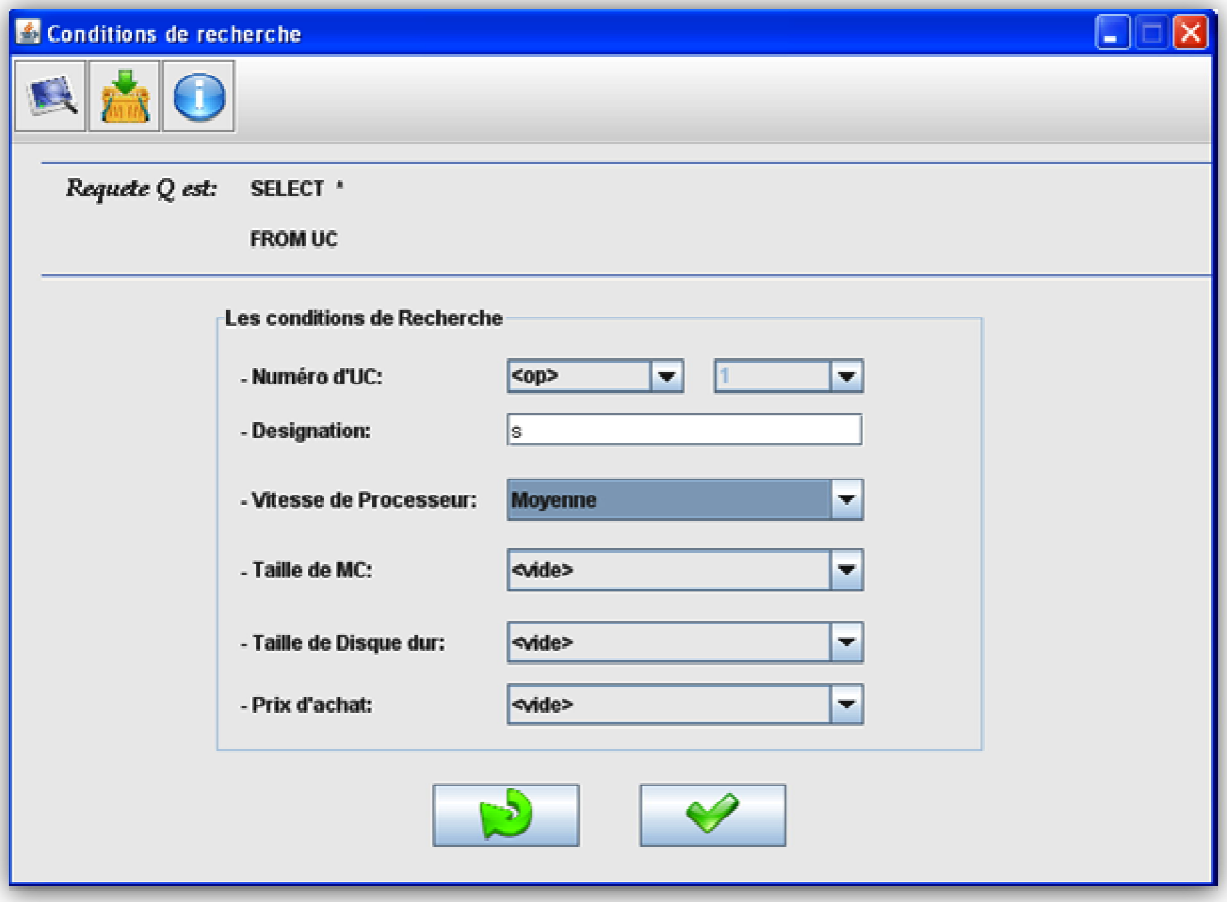
Etape 02 "*Choix des conditions" :* permet aux utilisateurs d'effectuer la (ou les) condition (s) de la requête.

Etape 03 "*Exécution de la requête" :* afficher les réponses de la requête d'utilisateur.



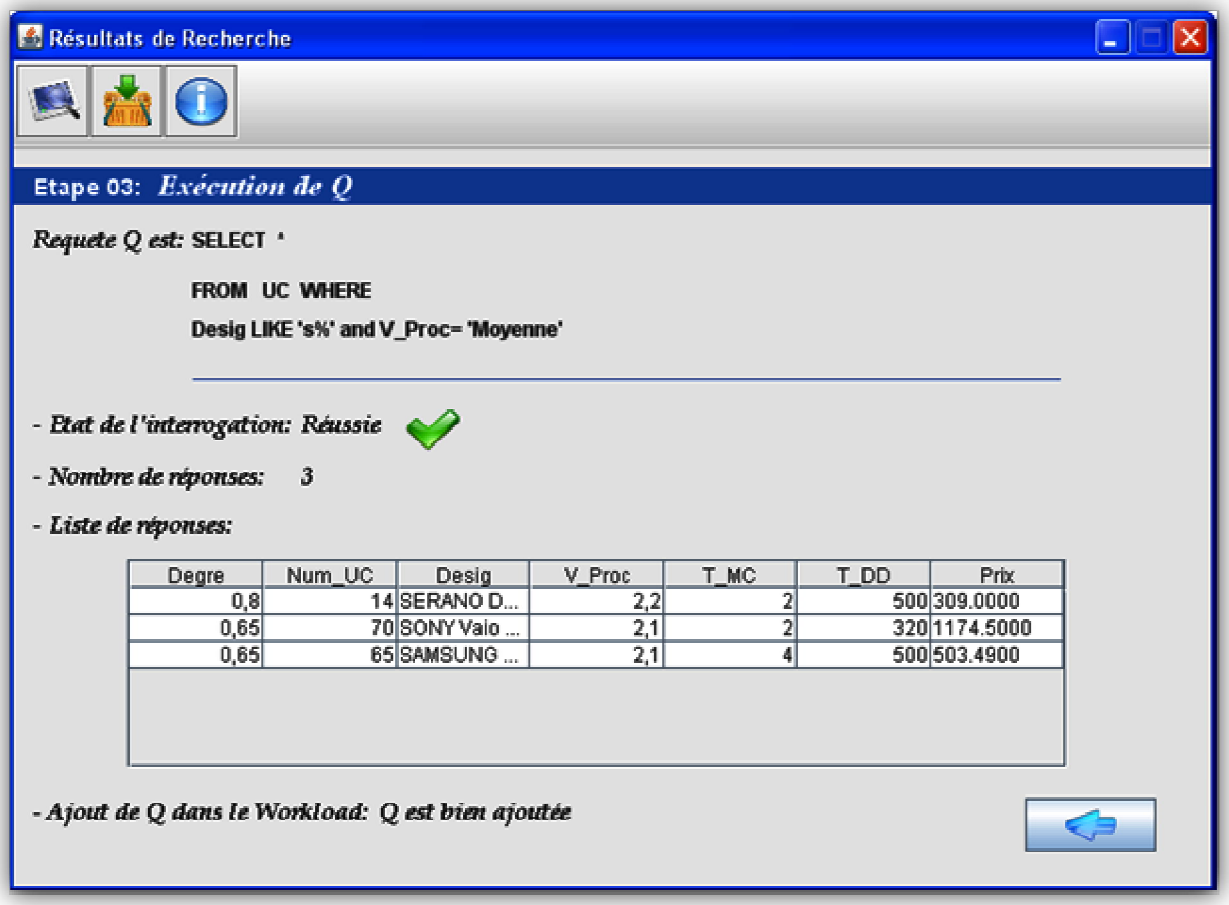
Etape 01 "*Choix des attributs*"

La figure suivante illustre la 2ème étape de l'interrogation flexible de BD



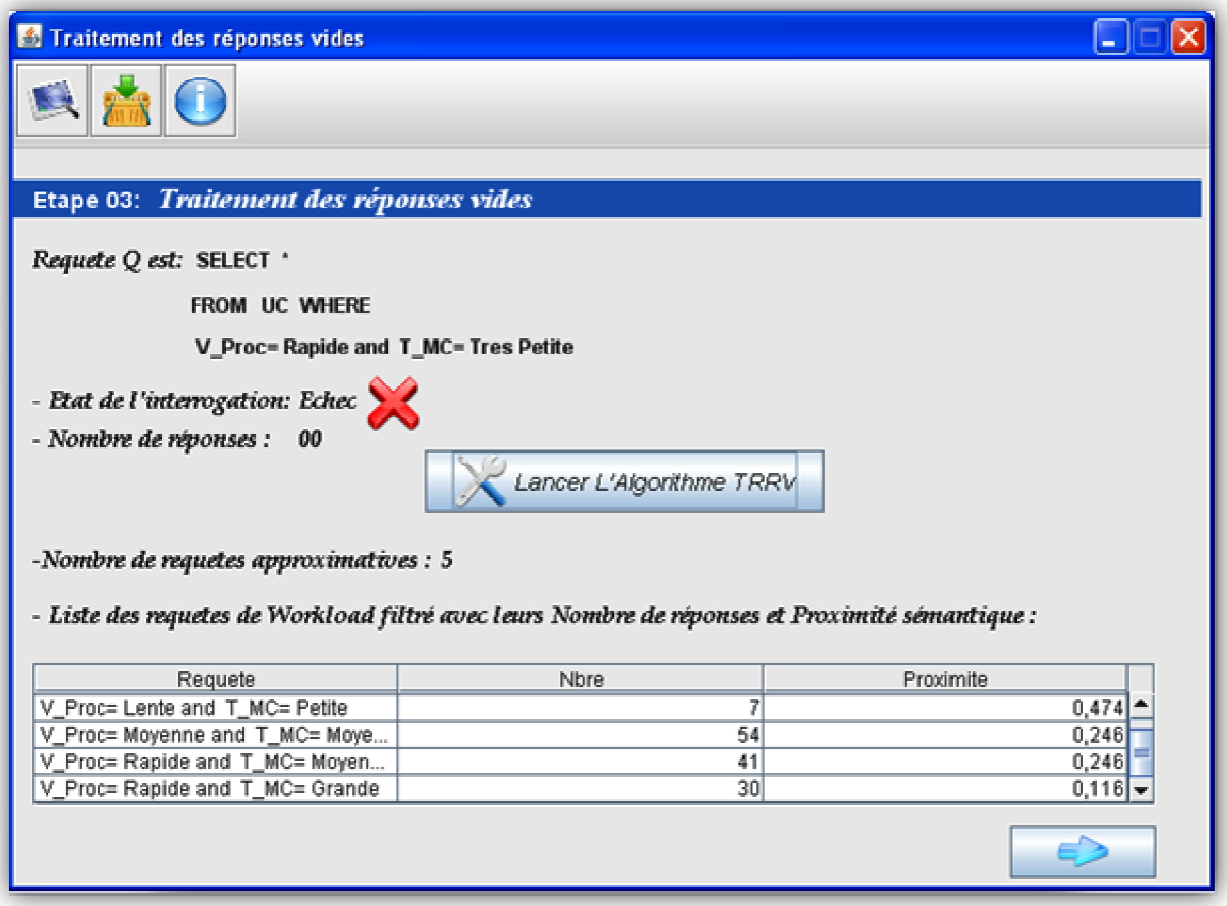
Etape 02 "*Choix des conditions*"

Lorsque l'utilisateur confirme l'envoie de la requête en appuyant sur le bouton valider, le système affiche la liste des réponses comme il a été montré dans la 3ème étape suivante.



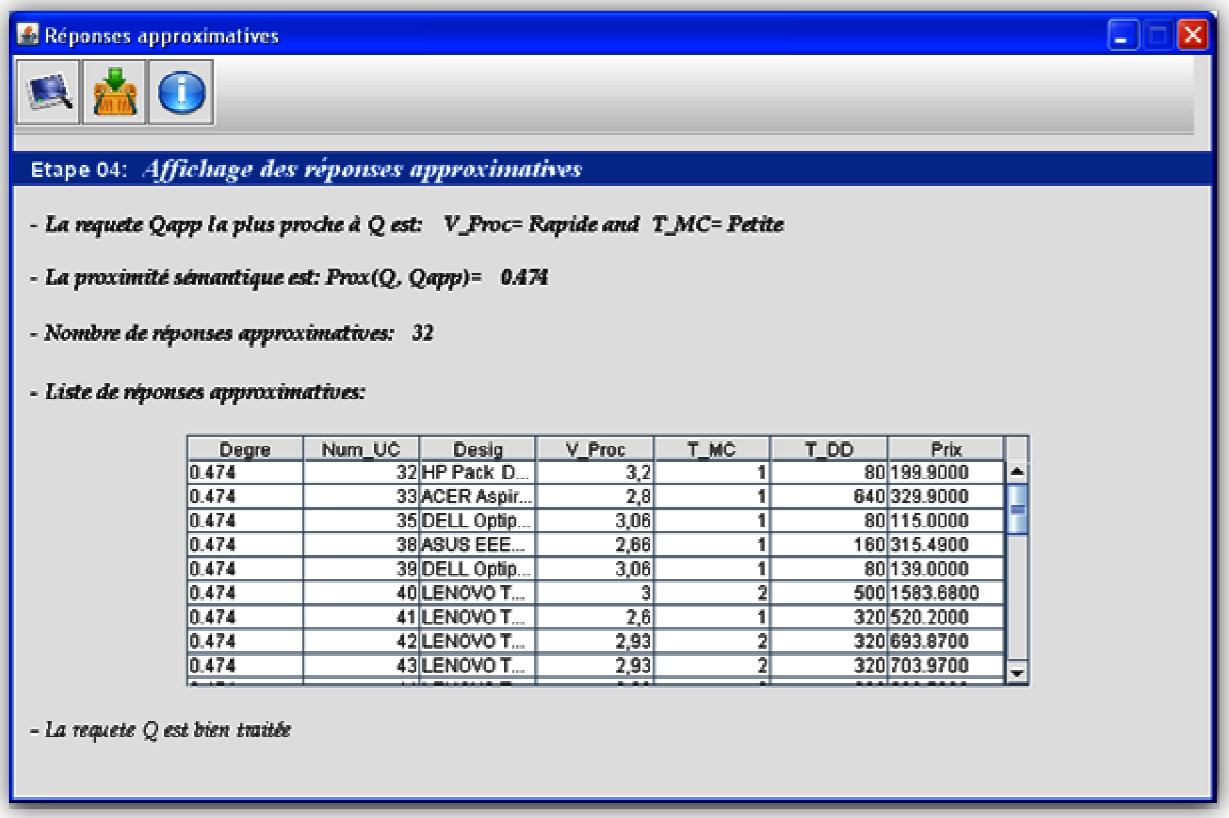
Etape 03 "*Exécution de la requête*"

Dans le cas où le problème de réponses vides est détecté, l'étape 03 prend comme nom " *Traitement des réponses vides*". Nous appliquons notre module TRV en appuyant sur le bouton « Lancer l'algorithme TRRV », qui affiche par la suite le nombre des requêtes approximatives à la requête à réponses vides. Ces requêtes sont affichées avec leurs nombre de réponses et proximité sémantique qui représentent les critères d'ordonnancement de ces requêtes approximatives.



Etape 03 " *Traitement des réponses vides"*

Une quatrième étape est donc accomplie, qui permet d'afficher les réponses approximatives à la requête à réponses vides.



Degre	Num_UC	Desig	V_Proc	T_MC	T_DD	Prix
0.474	32	HP Pack D...	3,2	1	80	199.9000
0.474	33	ACER Aspir...	2,8	1	640	329.9000
0.474	35	DELL Optip...	3,06	1	80	115.0000
0.474	38	ASUS EEE...	2,66	1	160	315.4900
0.474	39	DELL Optip...	3,06	1	80	139.0000
0.474	40	LENOVO T...	3	2	500	1583.6800
0.474	41	LENOVO T...	2,6	1	320	520.2000
0.474	42	LENOVO T...	2,93	2	320	693.8700
0.474	43	LENOVO T...	2,93	2	320	703.9700

Etape 04 " *Affichage des réponses approximatives"*

Annexe 02 : *codes source de l'application*

Dans cette annexe, nous présentons quelques codes source utilisés pour le fonctionnement de notre application.

Notre application possède 16 fichiers JAVA qui sont :

1. *Welcome.java* : pour le fonctionnent de l'interface de l'écran de démarrage

```
/*
 * FrameWelcome.java
 *
 * Created
 */
/**
 *
 * @author  Aggoune-ai
 */


public class Welcome extends javax.swing.JFrame {

    Thread monThread;
    int rappidite;

    private static Welcome Welcome;
    /** Creates new form FrameWelcome */
    public Welcome() {
        initComponents();
        this.setLocation(300, 100);
        this.setLocationRelativeTo(null);

    }
```

2. *Principale.java* : pour le fonctionnent de l'interface principale de l'application.

```
19    * @author Aggoune-ai
20    */
21   public class Principale extends javax.swing.JFrame {
22   public static String all,query,c,at, att,q ="";
23   public String host = "jdbc:odbc:DRIVER=Microsoft Access Driver (*.mdb); DBQ=tools\\UC.mdb; ";//PATH
24   public BDD dataBase;
25   public String pilote = "sun.jdbc.odbc.JdbcOdbcDriver";//DRIVERR
26   // fonction de fermeture d'une fenetre à un instant
27   public void hideFrame() {...}
30    public void Vérifier() {...}
40
41       /**...*/
42   public Principale() {
43         initComponents();
44         enableEvents(AWTEvent.WINDOW_EVENT_MASK);
45         try{
46          pgm();
47
48         }
49         catch(Exception e){
50             e.printStackTrace();
51         }
52        setSize(720, 525);
53        this.setLocationRelativeTo(null);
54
55       }
56   public void pgm() throws Exception{
```

3. *Condition.java :* pour le fonctionnent de l'interface de choix des conditions de la requête utilisateur.

```
44      /** Creates new form Conditions */
45      @SuppressWarnings("static-access")
46      public Conditions() {
47          initComponents();
48          setSize(720, 525);
49          attribut=new Principale();
50          @SuppressWarnings("static-access")
51          String question=attribut.query;
52          attr=attribut.Q;
53          aa=question;
54          jLabel3.setText(question);
55          this.setLocationRelativeTo(null);
56          jComboBox5.enable(false); // désactiver le numéro d'uc
57          enableEvents(AWTEvent.WINDOW_EVENT_MASK);
58        try{
59            this.jComboBox5.enable(false);
60         pgm1();
61
62        }
63        catch(Exception e){
64            e.printStackTrace();
65        }
66      }
67      public void num(String val1, String val2){
68        if ("<".equals(val1) && "1".equals(val2)){
```

4. *SansCondition.java :* pour le fonctionnent de l'interface de l'interrogation de BD sans conditions, en d'autre terme, elle permet à l'utilisateur de faire une consultation de données de la BD.

```
31
32      public SansConditions() {
33          initComponents();
34          setSize(720, 525);
35       this.setLocationRelativeTo(null);
36          attribut=new Principale();
37          @SuppressWarnings("static-access")
38          String question=attribut.query;
39          @SuppressWarnings("static-access")
40          String requete=attribut.Q;
41          jLabel3.setText(question);
42          attribut.dataBase = new BDD(pilote,host);
43          attribut.dataBase.executeQuery(requete);
44          jTable1.setModel(attribut.dataBase);
45          String aaa=question+" " +jLabel2.getText();
46          w[0]=aaa;
47          w[1]=requete;
48
49      }
50
```

5. *Workload.java :* pour le fonctionnent de l'interface de Workload

```
28    static Locale locale = Locale.getDefault();
29    static Date actuelle = new Date();
30    static DateFormat dateFormat =new SimpleDateFormat("dd-MM-yyyy HH:mm:ss");
31
32
33        /** Creates new form Workload */
34        @SuppressWarnings({"static-access", "static-access", "static-access"})
35
36
37        public Workload() {
38            initComponents();
39            setSize(720, 525);
40            this.setLocationRelativeTo(null);
41             BD=new BDD(pilote,host);
42
43             re1="INSERT INTO Wuc VALUES ('"+v.w[0]+"', '"+v.w[1]+"','"+ccc.pr[0]+"','
44              rep="SELECT Requete, Nbre FROM Wuc";
45             BD.executeUpdate(re1);
46             BD.executeQuery(rep);
47             jTable1.setModel(BD);
48             jLabel4.setText(""+BD.N);
49             jLabel7.setText(dateFormat.format(actuelle));
50        }
51
```

6. *Vérificateur.java* : consiste à exécuter la requête utilisateur.

```
44        public Vérificateur() {
45            initComponents();
46             setSize(720, 525);
47            this.setLocationRelativeTo(null);
48            dataBase = new BDD(pilote,host);
49            String requete;
50            int n;
51             pr=new Principale();
52    if (c.etat.equals("off")){ // interrgation classique
53     requete=c.attr +" "+jLabel5.getText()+" "+ c.cond;
54            jLabel3.setText(pr.query);
55            jLabel6.setText(c.cond);
56            dataBase.executeQuery(requete);
57            jTable1.setModel(dataBase);
58            nombre=dataBase.N;
59            n=dataBase.rows.size();
60            jLabel12.setText("" + n );
61             w[0]=con.p2;
62             w[1]=con.p3;
63             w[2]=con.attri;
64      String m=tester(con.p2);
65     jLabel15.setText(m);
66
67    }
68    else{
69    // interrogation floue
70       requete=c.pro +" "+"FROM UC"+ " "+jLabel5.getText()+" "+ c.contr;
71      jLabel3.setText(pr.query);
72      jLabel6.setText(c.cond);
73      dataBase.executeQuery(requete);
74            jTable1.setModel(dataBase);
75            n=dataBase.rows.size();
76            jLabel12.setText("" + n );
```

Les quatre fichiers suivants sont utilisés pour l'implémentation de la distance de Hausdorff.

7. *Integrale.java* : consiste à calculer l'intégrale de Richardson qui donne un meilleur résultat par rapport aux autres méthodes.

```
10  public class Integrale {
11  double[] polynome;
12      int degre;
13  public static int pred;
14  public static double  convert=0;
15      public double integre(double borneA, double borneB, int degre, double[] polynome, double pasH, Integrable maFonction, int z){
16          // Ax^2 + Bx + C ==> degré 2 = 3 elements dans le tableau "polynome".
17          if((polynome != null && polynome.length -1 != degre) | borneA>=borneB | pasH==0 | (polynome ==null & maFonction == null))
18              return 0;
19          this.polynome = polynome;
20          this.degre = degre;
21
22          double val = 0;
23          for(double temp=borneA+pasH ; temp<borneB ; temp+=pasH)
24              if(maFonction != null)
25                  val+= maFonction.evalueMaFonction(temp,z);
26              else
27                  val+=evaluePolynome(temp);
28          val = val*pasH;
29          if(maFonction == null)
30              return ((pasH/2)*(evaluePolynome(borneA) + evaluePolynome(borneB)) + val);
31          else
32              return ((pasH/2)*(maFonction.evalueMaFonction(borneA,z) + maFonction.evalueMaFonction(borneB,z)) + val);
33      }
34
35      //Cette methode ne fait qu'évaluer le polynome en un point "x".
36      private double evaluePolynome(double x){
37          double val = 0;
38          for(int i=0 ; i < polynome.length ; i++){
39              val += polynome[i]*Math.pow(x,degre-i);
40          }
41          return val;
42      }
44      public double integreExtrapolationRichardson(double borneA, double borneB, int degre, double[] polynome, double pasH,double pasH2,
45          double I1 = integre(borneA, borneB, degre, polynome, pasH, maFonction,z);
46          double I2 = integre(borneA, borneB, degre, polynome, pasH2, maFonction,z);
47          double m = pasH2/pasH;
48          double val = (Math.pow(m,2)*I1 - I2)/(Math.pow(m,2) -1) ;
49          return val;
50      }
51
52      public static void main(String[] s){
53
54
55      }
56  }
```

8. *Integrable.java* : c'est l'interface java pour le calcul de l'intégrale

```
10    public interface Integrable {
11    public double evalueMaFonction(double x, int z);
12    }
13
```

9. *mafct.java* : charge la fonction à intégrée, il implémente l'interface intégrable de fichier java précédent.

```
10  public class mafct implements Integrable {
11
12      public double evalueMaFonction(double x, int z) {
13          double valeurDeLaFonction = 0;
14          if (z==1){
15              // c,mc
16          valeurDeLaFonction = 2*(450*x-50*Math.pow(x,2));
17      }
18          else if (z==2){ //mc,c
19   valeurDeLaFonction = 2*(450*x+50*Math.pow(x,2));
20          }
21                  else if (z==3){ // 3 pour (mc,p) et 5 pour (p,mc)
22   valeurDeLaFonction = 2*(4450*x+150*Math.pow(x,2));
23          }
24         else if (z==4){ //4 pour (c,pc) et 6 pour (pc,c)
25   valeurDeLaFonction = 2*(4000*x+100*Math.pow(x,2));
26          }
27          else if (z==0){
28   valeurDeLaFonction = 0;
29          }
30  return valeurDeLaFonction;
31      }
32  }
```

10. *Hausdorff.java* : extrait de code pour calculer la distance de Hausdorff.

```
51          // calcule la distance de hausdorff
52          public static double H(double [] sup,double [] coupe, double s){
53        double h = 0; double mult,sum = 0;
54    int i=0;
55    while (i<coupe.length && coupe[i]!=0){
56     mult=sup[i]* coupe[i];
57     sum+=mult; i++;
58    }
59    h=sum/s;
60    return h;
61         }
62          // calcule les infs de A alpha
63         public static double [][] infH(double [][]A, double [][]B){
64        double ds[][] = new double [12][9];
65        int i=0; double inf=Math.abs(B[0][0]-A[0][0]);int k = 0;
66    while (i<B.length){
67        int j=0;
68
69            while (j<B[1].length && B[i][j]!=0){
70              int l=1;  inf=Math.abs(B[i][j]-A[k][0]);
71              while (l<A[1].length && inf!=0){
72                  double w=Math.abs(B[i][j]-A[k][l]);
73                  if (w<inf) {
74                      inf=w;
75                  }
76                  l++;
77              }
78              ds[i][j]=inf; j++;
79            }
80            k++; i++;
81
```

11. *info.java* : comporte les informations des prédicats flous modélisés au moyen des ensembles flous discrets.

```
public class info {
public final static double [][]VL={{1,1.2}, {0.6,1.6},{0.66,1.66}};
public final static double [][]VM={{0.5,2}, {0.65,2.1},{0.7,2.13}, {0.8,2.2},{1,2.3},{1,2.4}};
public final static double [][]VR={{0.53,2.53}, {0.6,2.6},{0.66,2.66}, {0.77,2.77},{0.87,2.8},
public final static double [][]MT={{1,0.128}, {1,0.256},{0.9,0.384}, {0.5,0.500},{0.5,0.512},{
public final static double [][]MP={{1,1}, {0.5,2},{0.25,3}};
public final static double [][]MM={{0.3,2}, {1,3},{0.7,4}};
public final static double [][]MG={{1,8}, {0.7,6},{0.5,4}};
public final static double [][]DP={{1,40}, {0.9,80},{0.5,160}, {0.3,250}};
public final static double [][]DM={{0.7,250}, {0.9,300},{1,320}, {0.5,500},{0.35,520},{0.2,640
public final static double [][]DG={{0.1,320}, {0.4,500},{0.5,520}, {0.8,640},{0.9,750},{1,1000


}
```

12. *getDegre.java* : implémente le système de code utilisé pour voir le degré d'appartenance d'un élément de la BD.

```
public class getDegre {
    public static String trapeze(String val1,String val2,String R2){
return "IIf([Prix] BETWEEN "+val1+" and "+ val2+ " , 1 , "+ R2+" )";
    }
    public static  String OPR(String op1,String val1,String ol,String op2,String val2,String R2){
        String R1=null,x=null;
        if (">".equals(op1) && "<".equals(op2) && "and".equals(ol)){
        R1="[Prix]/("+val2+"-"+val1+")-"+val1+"/("+val2+"-"+val1+")"; //coté gauche
        x="IIf([Prix]"+op1+val1+ol+"[Prix]"+op2+val2+ " , "+R1+ " , "+ R2+" )";
        }
        else if ("<".equals(op1) && ">".equals(op2) && "and".equals(ol)){
        R1=val1+"/("+val1+"-"+val2+")- [Prix]/("+val1+"-"+val2+")"; // coté droit
        x="IIf([Prix]"+op1+val1+ol+"[Prix]"+op2+val2+ " , "+R1+ " , "+ R2+" )";
        }
        else if ("<=".equals(op1) && ">=".equals(op2) && "or".equals(ol)){
        R1="0" ;
        x="IIf([Prix]"+op1+val1+ol+"[Prix]"+op2+val2+ " , "+R1+ " , "+ R2+" )";
        }

        else if ("<=".equals(op1) && op2==null && ol==null && val2==null){
        R1="0" ;
        x="IIf([Prix]"+op1+val1+ " , "+R1+ " , "+ R2+" )";
        }
return x;
    }
     public static String Cote(String val1,String R2){
return "IIf([Prix]>= "+val1+" ,1, "+ R2+" )";
    }
```

13. *TRRV.java* : implémente l'approche proposée, il représente le module TRV.

```
143        // mon approche
144        @SuppressWarnings("static-access")
145        public void TRVa(String x){
146              double prox;
147       String nomcol=null,query;
148       int m=0;
149        if (x.equals("V_Proc,T_MC,T_DD,Prix") ){
150        m=getcol("T_MC");
151        }
152        else    m=getcol(x);
153              nomcol=jTable2.getColumnName(m) ;
154              for (int i=0; i<dataBase.N;i++){
155                 //  System.out.print("La somme de alpha cuts de la requete "+ i+" est: ");
156                co=CoupeAlphat(gettab((String)jTable2.getModel().getValueAt(i, m)), gettab(c.pr[1]));
157                //calcule de alpha cut
158               // System.out.println(" "+som);
159                int k=0;
160                while (co[k]!=0){
161                // System.out.println(" "+co[k]);
162                 n++;
163                 k++;
164             }
165
166    // calcule l'ensemble A alpha
167               e=Hd.ens(gettab((String)jTable2.getModel().getValueAt(i, m)), co);
168                 int y=0;
169                    while(y<n){
170                     int j=0;
171                    while(j<gettab((String)jTable2.getModel().getValueAt(i, m)).length){
172     //System.out.println("la ligne "+y+" est: "+ e[y][j]);
173     j++;
174                    }
175                     y++;
176                    }
177    // calcule l'ensemble B alpha
178                 ee=Hd.ens(gettab(c.pr[1]), co);
179                 int yy=0;
180                    while(yy<n){
181                     int jj=0;
182                    while(jj<gettab(c.pr[1]).length && ee[yy][jj]!=0 ){
183
184    // System.out.println("la ligne "+yy+" de requete Q est: "+ ee[yy][jj]);
185     jj++;
186                    }
187                     yy++;
188                    }
189    // calcule de sup-inf
190                iinf=Hd.infH(ee, e);
191                int o=0;
192                    while(o<n){
193                     int oo=0;
194                    while(oo<gettab((String)jTable2.getModel().getValueAt(i, m)).length && iinf[o][oo]
195
196    //System.out.println("les infs de la ligne "+o+" de requete Q est: "+ iinf[o][oo]);
197     oo++;
198                    }
199                     o++;
200                    }
201    // calcule de H(B,A)
202                ssup=Hd.HAlpha(iinf);
```

14. *Rapprox.java* : consiste à afficher les réponses approximatives pour la requête à réponse vide.

```
16  public class Rapprox extends javax.swing.JFrame {
17      public TRRV ModuleTRV;
18      private Conditions c;
19      private BDD base;
20      static String requete;
21     private String pilote = "sun.jdbc.odbc.JdbcOdbcDriver";//DRIVER
22     private String host = "jdbc:odbc:DRIVER=Microsoft Access Driver (*.mdb); DBQ=tools\\UC.mdb; ";//PATH
23  public void hideFrame() {
24          super.hide();
25      }
26      /** Creates new form Rapprox */
27      @SuppressWarnings("static-access")
28      public Rapprox() {
29          initComponents();
30          setSize(800, 525);
31           requete=c.p1+" , "+ModuleTRV.convert+ " AS Degre "+ModuleTRV.r;
32
33           base = new BDD(pilote,host);
34          base.executeQuery(requete);
35          jTable1.setModel(base);
36           this.setLocationRelativeTo(null);
37
38           jLabel1.setText(ModuleTRV.requ);
39           jLabel2.setText(""+ModuleTRV.convert);
40           jLabel5.setText(""+base.N);
41
42
43      }
```

15. *BDD.java* : charger à exécuter l'interrogation de la BD.

```
12  public class BDD extends javax.swing.table.AbstractTableModel {
13      int nMAJ=0;
14      public static int N;
15  private final String DRIVER="sun.jdbc.odbc.JdbcOdbcDriver";
16                  Connection          connection;
17              Statement        statement;
18              ResultSet        resultSet;
19              String[]         columnNames = {};
20              @SuppressWarnings("unchecked")
21                  Vector           rows = new Vector();
22              ResultSetMetaData   metaData;
23
24          public BDD(String driverName,String url) {
25                  try {
26                      Class.forName(driverName);
27                      System.out.println("ouverture de connection");
28                      connection = DriverManager.getConnection(url);
29                       statement = connection.createStatement(
30                   ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATABLE );
31                  }
32                  catch (ClassNotFoundException ex) {
33                      System.err.println("Cannot find the database driver classes.");
34                      System.err.println(ex);
35                  }
36                  catch (SQLException ex) {
37                      System.err.println("Cannot connect to this database.");
38                      System.err.println(ex);
39                  }
```

16. *About.java* : implémente l'interface de l'item A-propos

```
16    public class About extends javax.swing.JFrame {
17
18        /** Creates new form About */
19        public About() {
20            initComponents();
21             this.setLocationRelativeTo(null);
22            //setSize(651, 270);
23
24        }
```

Résumé

L'interrogation des bases de données, en particulier celles accessibles via le Web, est devenue une tâche omniprésente pour les utilisateurs. Dans la plupart des cas, les requêtes formulées prennent en compte les préférences des utilisateurs afin de mieux personnaliser les réponses désirées. Un des formalismes permettant d'exprimer les préférences dans les requêtes est la théorie des ensembles flous. Dans ce contexte, les requêtes sont dites des requêtes flexibles dont l'avantage est de retourner un ensemble de réponses discriminées plutôt qu'un ensemble plat de réponses.

Un des problèmes auxquels les utilisateurs sont confrontés dans un processus d'interrogation, est le problème des réponses vides. C'est-à-dire, il n'existe aucun item de la base de données interrogée qui satisfait un tant soit peu la requête de l'utilisateur. Une des solutions pour traiter ce problème consiste à exploiter un ensemble de requêtes (Workload-past query) précédemment exécutées par le système et ayant retournées des réponses non vides. L'objectif de ce travail est d'étudier dans quelle mesure cet ensemble de requêtes pourrait aider à répondre d'une manière approximative, à une requête à réponse vide. Pour cela, une première étape consiste tout d'abord à proposer une mesure permettant d'estimer la proximité entre requêtes sémantiquement parlant.

Mots-clés : Bases de données, requêtes flexibles, proximité sémantique, distance de Hausdourff.

Abstract

The interrogation of the databases, in particular those accessible via the Web, became an omnipresent task for the users. In most cases, the formulated queries take into account the preferences of the users in order to better personalize the desired answers. One of the formalisms making it possible to express the preferences in the queries is the theory of fuzzy set. In this context, the queries are known as flexible queries whose advantage is to turn a set of discriminated responses rather than a set of flat responses.

One of the problems with which the users are confronted, in a process of interrogation, is the problem of the empty answers. I.e., the problem of not being able to provide the user with any data fitting his/her queries. One of the solutions to trait this problem consists in exploiting a set of queries (Workload-past query) previously executed by the system and returned the non empty answers. The objective of this work is to study how this set of queries might help to answer in an approximate way, at a query with empty answer. For this, a first step consists to begin by proposing a measure to estimate the proximity between queries semantically speaking.

Keywords: Data base, Flexible queries, Semantic proximity, Hausdourff distance.

الملخص:

أصبح استجواب قواعد البيانات ، ولا سيما تلك التي يمكن الوصول إليها عبر الويب ،مهمة دائمة الوجود بالنسبة للمستخدمين. في معظم الحالات ، الأسئلة المصاغة تأخذ بعين الاعتبار أفضلية المستخدمين من أجل تحسين تخصيص الإجابات المطلوبة. واحدة من الشكليات التي تسمح بالتعبير عن الأفضلية في الأسئلة هي نظرية المجموعات الغامضة.و في هذا السياق ، تدعا الأسئلة بالأسئلة المرنة التي تمتاز بردها بمجموعة من الإجابات المتميزة بدلا من مجموعة مسطحة من الإجابات.

من بين المشاكل التي تواجه المستخدمين في عملية الاستجواب، هي مشكلة الإجابات الفارغة. وهذا يعني، لا يوجد أي عنصر في قاعدة البيانات يرضي ولو قليلا طلب المستخدم. أحد الحلول لمعالجة هذه المشكلة هو استغلال مجموعة من الأسئلة التي سبق و أن نفذها النظام والتي تعطي إجابات غير فارغة. الهدف من هذا العمل هو دراسة إلى أي مدى هذه المجموعة من الأسئلة يمكن أن تساعد على الإجابة بطريقة تقريبية ،لطلب يخلو من الجواب. لهذا ، كخطوة أولى ومن المقرر أن نبدأ باقتراح قياس لتقدير القرب بين الأسئلة.

كلمات مفتاحيه: قاعد البيانات، الأسئلة المرنة ، القرب المعنوي، مسافة هوسدورف.