Projet Base de Données Avancées - Thésaurus

Julien Deguilhem, Maïlys Denis, Sébastien Gautheron,

Johann Mitrail, Anthony Rossi, Colin Vidal

Master 1 Informatique - Université Montpellier 2

Table des matières

1 Introduction

Dans le cadre du Master 1 Informatique à l'Université Montpellier 2, nous avons été amenés à réaliser un projet de groupe pour le cours de base de données avancées dirigé par Thérèse Libourel.

Le projet proposé consiste à analyser et implémenter un système de gestion de vocabulaire partagé. Un vocabulaire partagé, aussi appelé Thésaurus, est une liste de termes organisés selon des liens qui les unissent. Les termes sont reliés entre eux par diverses relations comme la synonymie, la généralisation, la spécialisation, l'association...

Le projet qui nous a été assigné, "le thésaurus", est pertinent par rapport au sujet de la base de données parce qu'au final un thésaurus est un outil linguistique qui permet de voir les relations entre des concepts concernant un domaine de connaissance. Ainsi l'une de notre mission est de trouver la structure optimale qui mettra en avant les relations sémantiques et d'équivalence.

L'équipe de projet est constitué de six étudiants : Julien Deguilhem, Maïlys Denis, Sébastien Gautheron, Johann Mitrail, Anthony Rossi et Colin Vidal.

Pour le thésaurus, nous avons choisi de travailler sur le thème des animaux. C'est un thème vaste qui permet de créer beaucoup de lien entre tous les animaux, en partant du nom latin de l'animal jusqu'à créer des liens de synonymie, d'espèces, de races, de cousins...

2 Cahier des charges

Le projet a débuté début novembre 2013 et doit se terminer début janvier 2014.

A l'issue de cette période l'équipe de projet présentera le suivi du projet

à travers une soutenance orale et rendra un rapport de projet concis.

2.1 Fonctionnalités

L'application sera un site web ergonomique et simple d'utilisation.

L'application demandée devra permettre à un utilisateur de rechercher un mot dans le thésaurus et pouvoir avoir la liste des termes qui lui sont liés. Il pourra voir les termes synonymes, les généralisations et spécialisations du mot recherché.

Puis, l'utilisateur pourra naviguer de termes en termes grâce à la liste des mots retournés lors de la recherche d'un précédent mot.

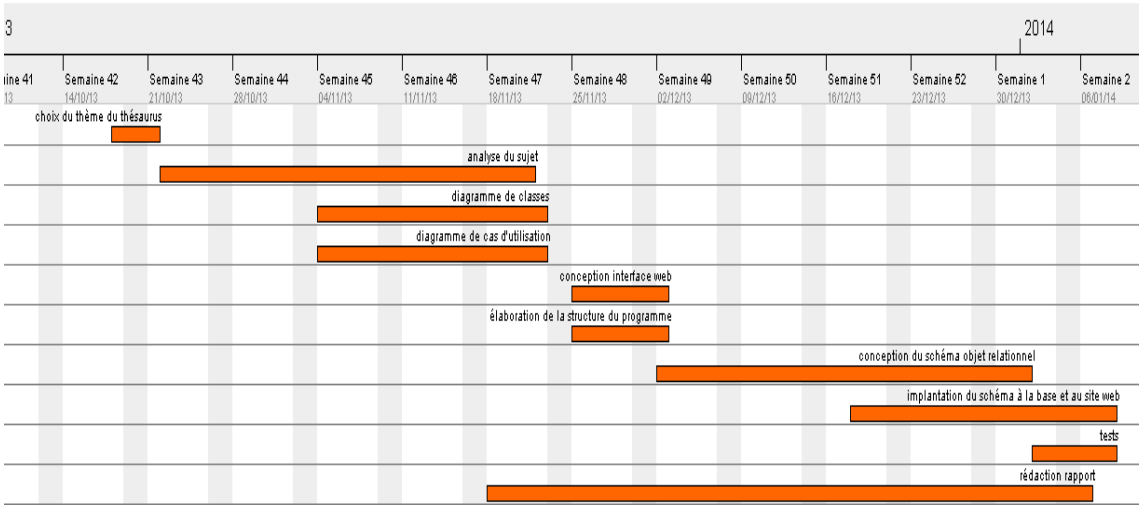
De plus, l'utilisateur aura la possibilité d'ajouter, d'éditer ou de supprimer des synonymes, des spécialisations ou des généralisations à un mot.

2.2 Fiabilité

L'application avertit l'utilisateur dans le cas où le mot recherché n'est pas dans la base de données et après l'ajout, la suppression ou l'édition d'un mot.

Lors de l'ajout, de la suppression ou de l'édition de relations à un mot, l'application est en mesure de refuser si la modification compromet le bon fonctionnement de la base de données.

3 Gestion de Projet



Pour le projet, nous avons commencé par analyser le sujet. Nous avons confronter nos différents diagramme de classes et nous avons confronté nos idées durant près d'un mois pour produire le diagramme de classes et de cas d'utilisation.

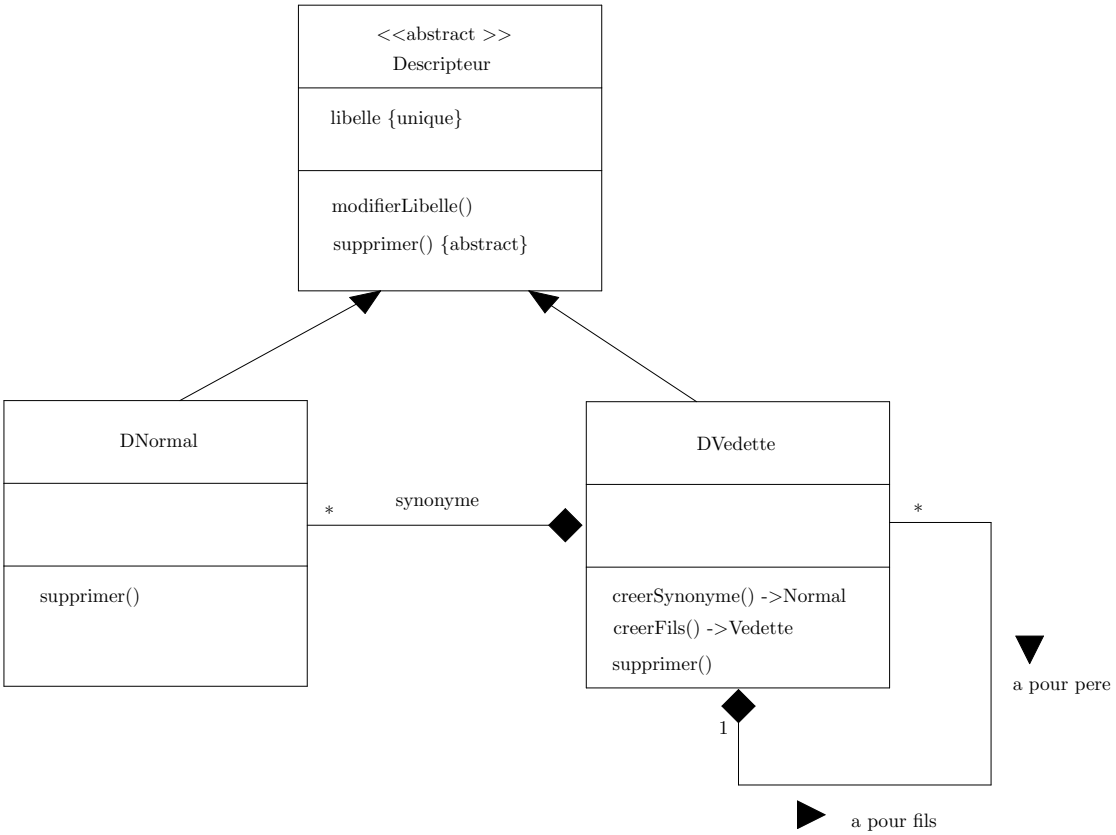
Une fois l'analyse terminée, nous avons décidé de nous répartir les tâches selon les envies de chacun et les besoins du projet.

Anthony Rossi est responsable du design qui sera donc l'interface de notre thésaurus. Colin Vidal prend en charge la mise en place du serveur et de la base de données, ainsi que de la modélisation de la base de données et son implémentation SQL et PHP. Maïlys Denis est responsable de la rédaction du rapport, organisation du projet et résumé des réunions. Johann Mitrail et Sébastien Gautheron se sont occupés d'aider les personnes qui en avaient besoin lors du projet, par exemple lors de la création du design, ou encore sur le rapport (création de diagrammes, etc). Enfin, Julien Deguilhem et Maïlys Denis s'occupent de l'intégration de l'interface HTML du programme avec le code PHP, ainsi de l'interface homme-machine, et du contrôle de qualité du

produit fini.

4 Modélisation

4.1 Modèle sous forme d'arbre



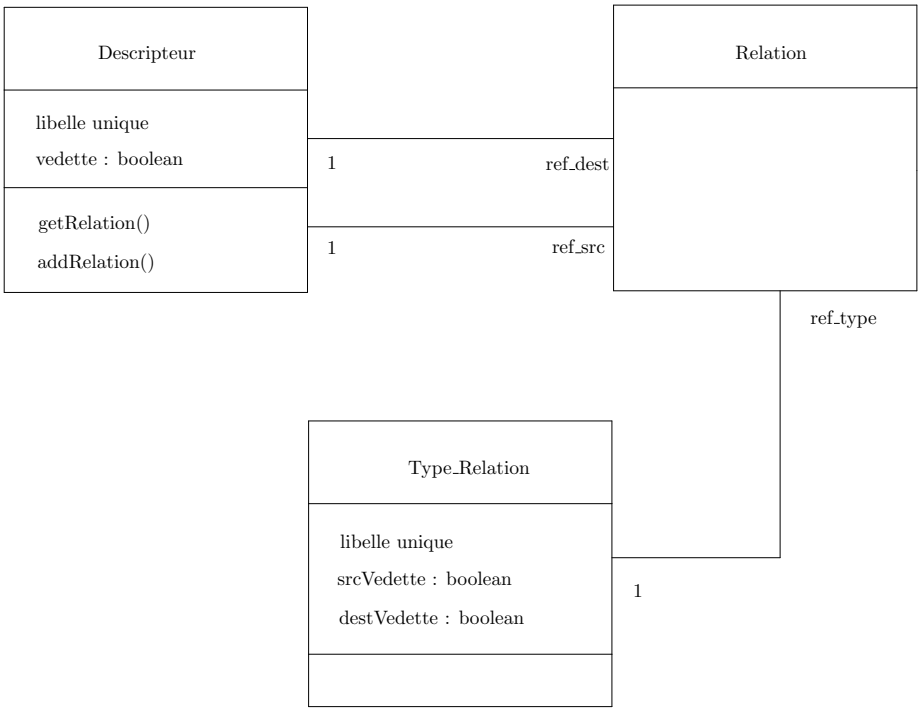
Dans un contexte de thésaurus traitant des animaux, notre première approche fût de concevoir un modèle objet relationnel qui prenne en compte par sa conception (c'est à dire sans avoir recours à des contraintes particulières) la notion de hiérarchie entre les termes, illustrant implicitement les relations de généralisations et spécialisations entre des termes vedettes. Afin d'illustrer

d'autres aspects relationnels entre les termes, nous pouvons ajouter des relations au travers de nouvelles classes, comme l'illustre la classe de Synonymie, qui regrouperait alors des termes non vedettes.

Le modèle est intéressant dans notre situation puisque léger à mettre en oeuvre : pas besoin d'utiliser de triggers ou de contraintes complexes pour assurer l'intégrité de la base. En revanche il en est pas moins extrêmement statique, et très spécialisé. On peut imaginer des domaines où une relation de hiérarchie n'est pas nécessaire, où encore un domaine où l'héritage multiple est indispensable, ce que ne peux fournir ce modèle. Par ailleurs, l'ajout de différents types de relations est relativement pénible, puisqu'il impose de modifier le modèle, et probablement les autres couches du programme.

De plus, la contrainte d'utiliser l'objet relationnel pose différents problèmes au niveau du stockage des données. Il n'est pas possible d'intégrer dans des tables imbriquées les noeuds que contiennent un noeud père (relation de spécialisation) car cela rendrait la recherche d'un terme vedette très complexe (il faudrait parcourir chaque branche de l'arbre à partir de la racine, sans index possible). L'utilisation de références d'objet s'avère alors utile, mais le problème de suppression de noeuds se pose alors : il n'est pas possible sous Oracle d'avoir une suppression automatique d'un tuple étant référencé dans un autre tuple qu'on supprime, contrairement au relationnel, où les clés étrangères et l'option FOREIGN KEY permet d'assurer cette fonction. On serait alors obligé d'avoir recourt à des déclencheurs particulièrement pénibles à mettre en oeuvre (et sujet à être modifié si l'on rajoute de nouvelles relations).

4.2 Modèle sous forme de graphe orienté (ou réseau lexical)



Afin de trouver un palliatif aux problèmes de souplesse que pose le modèle précédent, nous nous sommes inspirés de la vision des réseaux lexicaux telle qu'on peut la retrouver sur JeuxDeMots [1]. Il s'agit d'un graphe orienté, où les noeuds représentent des termes et les arcs des relations, valués et orientés. Dans le cadre du thésaurus, seule l'orientation de ces relations nous intéresse : une relation de type spécialisation entre $A \rightarrow B$ est dite *sortante* de A, et entrante de B.

L'intérêt de ce modèle est qu'il offre une souplesse extrême puisqu'on

1. http ://www.jeuxdemots.org réalisé par Mathieu Lafourcade

peut aisément manipuler toute sorte de relations entre tous les noeuds de la base de données, permettant de s'adapter facilement à de nombreux types de thésaurus. Du point de vue de l'implémentation, deux classes (une classe contenant les termes, une autre les relations) suffisent.

Afin d'assurer des contraintes entre différentes relations (par exemple, le fait que la relation de synonymie puisse s'exprimer entre un descripteur vedette et un non vedette uniquement), nous avons décidé de concevoir une classe supplémentaire, qui répertorie l'ensemble des relations possibles du modèle, et qui indique par deux booléens si le terme source de la relation doit être vedette, et si le terme de destination de la relation doit l'être également. Cet ajout permet d'adapter le modèle à de nouveaux contextes (nouvelles relations, contraintes entre relations) sans modifier le schéma, simplement en ajoutant des tuples dans la classe répertoriant les relations.

Cela nécessite un déclencheur assez lourd qui consiste lors de chaque ajout ou modification de relation à vérifier l'état des termes (vedettes ou non) avant d'effectuer la modification dans la base de données. Néanmoins, ce déclencheur n'a pas besoin d'être modifié ou adapté en fonction du contexte, ce qui est un atout vis à vis du modèle précédent. Enfin, bien que cela relève plus de l'implémentation que de la modélisation, la classe de relations est représenté en base de données par le triplet de références d'un descripteur source, descripteur destination et d'un type de relation. Oracle ne permettant pas de placer une clé primaire sur des champs qui sont des références, nous devons ajouter un second déclencheur qui assurera la contrainte d'unicité de la table avant chaque modification.

4.3 Modèle retenu

En raison de sa souplesse face à différentes situations (non nécessité de modifié le schéma en case de modification du modèle, déclencheurs qui n'ont pas besoins d'être spécialisés), nous avons décidé de retenir le second modèle.

Nous voyons cependant des limites à ce modèle sur deux plans. Premièrement, dans le cas d'un thésaurus très vaste, on pourrait imaginer que deux termes ont la même étiquette, et que l'un soit vedette alors que l'autre ne l'est pas. L'étiquette étant la clé primaire d'un terme, cela est impossible. Ce problème peut par exemple se produire dans un thésaurus où *caisse* est un synonyme de *auto-mobile*, et où *caisse* serait un descripteur vedette pour illustrer le concept de *conteneur*. Ce type de problème pourrait être résolu par l'adoption comme clé primaire de l'étiquette et du booléen vedette sur un descripteur.

Par ailleurs, si le premier modèle permettait d'exprimer les relations de généralisation/spécialisation de manière synchrone (c'est à dire que si on ajoutait *chat* comme spécialisation de *félin*, ce dernier aurait automatiquement la relation de généralisation vers *chat*), ce qui est impossible dans le modèle que nous avons choisi, car il nécessiterait du code spécifique (donc moins de souplesse), et aussi parce qu'il pourrait poser des problèmes de polysémie suivant les thésaurus : *chat* représente l'animal et la discussion instantanée. Si on ajoute une relation de généralisation vers *félin*, il est impossible d'y ajouter automatiquement une relation de spécialisation de *félin* vers *chat*, puisque ce dernier peut représenter aussi la discussion instantanée. Ce serait sémantiquement faux. Cependant, bien que ça ne résolve pas complètement le problème, les relations ayant un sens (entrantes ou sortantes), on peut tout de même naviguer entre les termes.

Cela assure tout de même un bon compromis entre souplesse d'utilisation

et complexité du modèle.

5 Spécifications

Afin de simplifier le développement (l'installation et la configuration d'Oracle n'étant pas triviale), nous avons mis en place une architecture centralisée basée sur une machine virtuelle KVM hébergeant une distribution GNU/Linux CentOS 6.4 sur l'un de nos serveurs personnels.

5.1 Serveur de gestion de base de données

Il était nécessaire d'utiliser un SGBD gérant l'objet relationnel pour ce projet. À l'heure actuelle, les deux SGBD éprouvés et supportant cette norme sont Oracle et PostgreSQL. Notre choix s'est porté sur Oracle 11g en raison de notre formation sur ce type de système.

Notons que nous aurions préféré utilisé PostgreSQL en raison de son aspect libre et fortement utilisé en entreprise, mais la complexité d'utilisation d'un tel SGBD que l'on ne maîtrise pas aurait prit un temps disproportionné sur ce projet.

5.2 Serveur Web

Nous avons utilisé Apache 2.2 pour le service web de notre application. Le choix du langage de programmation s'est porté sur PHP (avec le module OCI pour interaction avec le SGBD Oracle), en raison de sa maîtrise par tous les membres du projet.

5.3 Serveur de gestion de version

Dans le contexte de gestion de projet, nous avons utilisé le service fourni par GitHub pour centraliser et gérer nos versions du projet afin de faciliter l'ajout, la modification et la suppression de code.

6 Implémentation

6.1 Base de Données

Nous avons implémenté en objet relationnel le schéma UML correspondant au modèle sous forme de graphe orienté. De plus, on a implémenté des Trigger PL/SQL afin que la base de données reste cohérente en fonctions des contraintes imposés entre les relations (définies dans la table types_relations) :

CHK_MATCH_RELATION : Assure que l'état (vedette ou non vedette) du descripteur source et destination de la relation correspondent bien à l'état du descripteur source et destination imposé par le type de relation choisi.

CHK_UNIQUE_RELATION : Vérifie que la relation à créer n'est pas déjà existante (étant donné qu'il est impossible de placer une contrainte d'unicité sur des références).

À l'heure actuelle, la table types_relations ne comportent que trois tuples correspondant à trois types de relations possibles : Synonyme, Généralisation et Spécialisation du mot courant. D'autres relations peuvent être aisément ajoutées par la suite car le diagramme de classe et l'infrastructure web ont été pensé ainsi.

6.2 Application web

Nous avons donc crée une application web en PHP pour l'utilisation et la gestion des données. Nous avons utilisé le module PHP OCI pour communiquer avec la base de données Oracle.

Afin de garder une approche objet, les accès à la base de données par l'application se font au travers d'envois de messages à des objets de type *Descripteur* appelant des méthodes dédiées (ajout de relations, par exemple) et facilitant l'intégration de l'interface utilisateur.

Par ailleurs, dans un but de simplification d'URL, nous avons utilisé le module de réécriture d'adresse d'Apache. Ainsi, au lieu de passer des paramètres de type GET sous forme d'une syntaxe lourde pour l'utilisateur (par exemple *http ://domain.tld/index.php ?libelle=chat*), l'utilisateur peut accéder à un libellé en tapant directement la chaine de caractères de l'animal recherché après le nom de domaine soit *http ://domain.tld/chat/*.

6.3 Interface utilisateur

Notre interface web comporte deux pages principales :

L'accueil une zone de recherche de descripteur, une zone d'ajout de descripteur vedette ou non, et une zone de présentation du projet.

L'affichage du descripteur courant une zone de recherche de descripteur, une zone d'ajout de descripteur vedette ou non, une zone d'affichage des relations du descripteur courant et pour chacune des relations une zone d'ajout de relation à ce même descripteur.

Plus de détails sur les différentes zones du site :

Zone de recherche de descripteur Formulaire qui permet à l'utilisateur de rechercher un terme, après avoir cliqué sur le bouton "recherche",

il obtiendra la zone d'affichage des relations du descripteur courant. Si le terme recherché n'est pas dans la base de données, l'utilisateur en sera averti.

Zone d'affichage d'un descripteur Cette zone comprend la liste des types de relations et des mots reliés au descripteur recherché. A chaque type de relation existe une zone d'ajout de relation, c'est à dire que l'on peut ajouter un mot qui aura cette relation avec le descripteur courant. Il est indiqué si le mot recherché est vedette ou non.

Zone d'ajout de relation Dans ce formulaire, l'utilisateur peut ajouter une relation au descripteur recherché. Chaque type de relation est composé d'un formulaire d'ajout de relation. On doit donc renseigner le descripteur au bon formulaire concernant le type de relation que nous voulons ajouté. Si l'ajout d'une relation compromet le bon usage de la base, un message d'erreur déclenché par un Trigger apparaitra. Si l'ajout d'une relation comprend un mot qui n'a pas encore été déjà ajouter dans la base de données, un message d'erreur apparait, c'est pourquoi il faut impérativement ajouter le terme voulu grâce à la zone d'ajout de descripteur avant d'ajouter une quelconque relation.

Zone d'ajout de descripteur Ce formulaire permet d'ajouter un descripteur à la base. On doit préciser si le descripteur à ajouter est vedette ou non. Si le descripteur est déjà existant, un message d'erreur déclenché par un Trigger apparaîtra.

7 Conclusion

7.1 Difficultés rencontrées

Nous avons rencontré quelques difficultés concernant la phase d'analyse du sujet en particulier l'élaboration du diagramme de classe. Notre premier diagramme de classe était fait en sorte que l'on puisse créer une hiérarchie de mots sous forme d'arbre mais aussi une liste de synonyme.

Cependant, nous l'avons repensé car l'idée de la génération d'un arbre n'est pas adapté au problème de l'utilisation de ce thésaurus dans toute sorte de domaine (et non pas exclusivement au domaine des animaux).

7.2 Résultats

A l'issue de ce projet, nous avons réussi à implémenter une base de données objet relationnel et un site web. Cette implémentation nous permet de rechercher un mot, ajouter un mot et des relations et obtenir la liste des relations qu'il possède. Cependant, par manque de temps, nous aurions aimé inclure des zones d'édition et de suppression des descripteurs, des relations et des types de relations mais aussi améliorer l'affichage des erreurs lancées par les Triggers lors d'un ajout ambigu de descripteur ou de relation.