INSA RENNES

Cinquième année - Informatique

Base de données Fédérée

2015 - 2016

Réalisé par :

NOUR ROMDHANE
BESNARD SYLVAIN
BOULUAD MOHAMMED
EL OMARI ALAOUI HASSAN
MARCHAIS JULIEN
MARCOU JULIEN
SEERUTTUN–MARIE DAN

Table des matières

1 Introduction

Dans le cadre de notre troisième année d'études au département Informatique à l'INSA Rennes, nous réalisons un projet technique en équipe. Le sujet de notre projet porte sur les bases de données fédérées.
Ce système de base de données défini pour la première fois en 1985 par McLeod et Heimbigner, suite à la parution de leur article « A federated architecture for information management » [1], est de plus en plus courant en entreprise. En effet, lors d'une fusion entre deux sociétés, il est important d'unifier leur système d'information. Pendant ce processus fastidieux pour de nombreuses entreprises, il est courant de voir des bases de données possédant des systèmes de gestion différents. Ces systèmes d'information possèdent des bases de données à la structure éloignée. Il faut donc un outil permettant d'interroger des bases de données structurellement différentes, tout en affichant un résultat unifié. L'objectif de la création d'une base fédérée est de donner aux utilisateurs un outil permettant d'interroger plusieurs systèmes à priori hétérogènes et de retourner un résultat unifié.
Sheth et Larson, lors de leur article « Federated database systems for managing distributed, heterogeneous, and autonomous databases » [2] paru en 1990, divisent ces systèmes fédérés en deux catégories : les systèmes faiblement couplés (à l'aide des mécanismes des bases de données multibases et réparties) et les systèmes fortement couplés (réalisant l'intégration à l'aide d'un SGBD fédéré).
Notre solution vise à créer et à comprendre l'utilisation d'une base de données fédérée faiblement couplée entre une base de données relationnelle (gérée en *SQL*) et une base de données *noSQL*. Nous avons décidé de ne pas développer la partie qui vise à séparer la requête générale (XQuery) en sous requêtes (médiateur) mais plutôt de nous concentrer sur la partie traduction de langage et interface des bases de données. Ce rapport vise à expliquer nos choix techniques et notre réalisation concernant ce sujet.
Pour ce faire, dans une première partie, nous expliquons les principes et le fonctionnement d'une base de données fédérée. Ensuite, dans une seconde partie, nous discutons des choix techniques de notre solution et de son architecture.

2 Contexte du projet

2.1 Introduction à l'Architecture fédérée

2.1.1 Principe

Dans un système d'information d'une entreprise, différents systèmes de gestion de bases de données répartis géographiquement sont utilisables pour stocker, gérer et rechercher les données.

L'architecture fédérée offre une vue commune des données stockées dans différentes bases de données et facilite l'accès aux utilisateurs. En effet, l'objectif de cette architecture est de donner la possibilité d'envoyer des requêtes réparties à plusieurs sources de données dans une seule instruction.
Cela offre un grand avantage en conservant les données où elles résident, plutôt que de les déplacer dans une base de données unique.

Cette architecture doit assurer l'intégration des données. En effet, elle traite l'hétérogénéité sémantique qui est à l'origine des conceptions indépendantes des différentes bases de données. Aussi, elle fait la traduction de tous les schémas dans un modèle commun (dit canonique ou pivot) pour s'adapter à l'hétérogénéité syntaxique, due à l'utilisation de modèles différents dans les bases de données composantes. La figure 2.1 illustre cette notion.

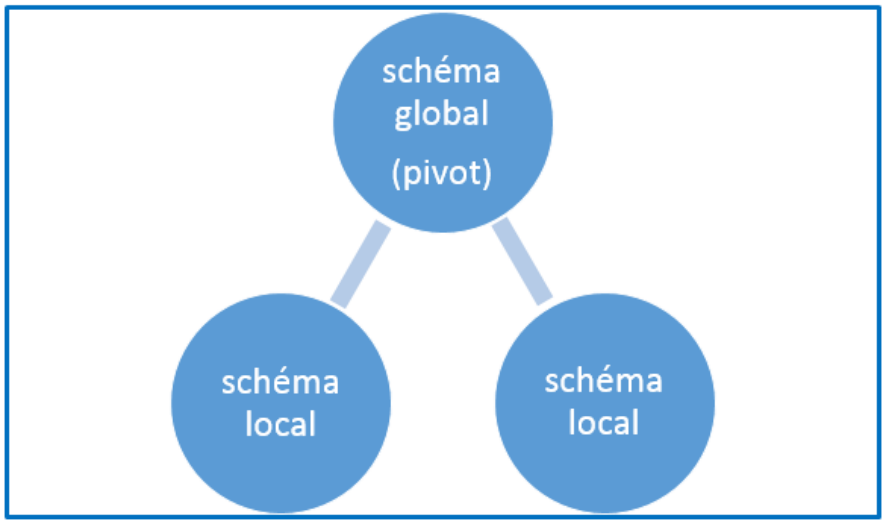

FIGURE 2.1 – Schéma global

Le fonctionnement de l'architecture peut être séparé en 3 parties (détaillées dans la figure 2.2) que nous abordons en détail dans les sous chapitres qui suivent.

Lors de la réception d'une requête (Q), le schéma de médiation la divise en sous requêtes adressées à la bonne source (q1, q2). Ensuite, une traduction en langages compréhensibles par les sources locales est faite au niveau source par un adaptateur.

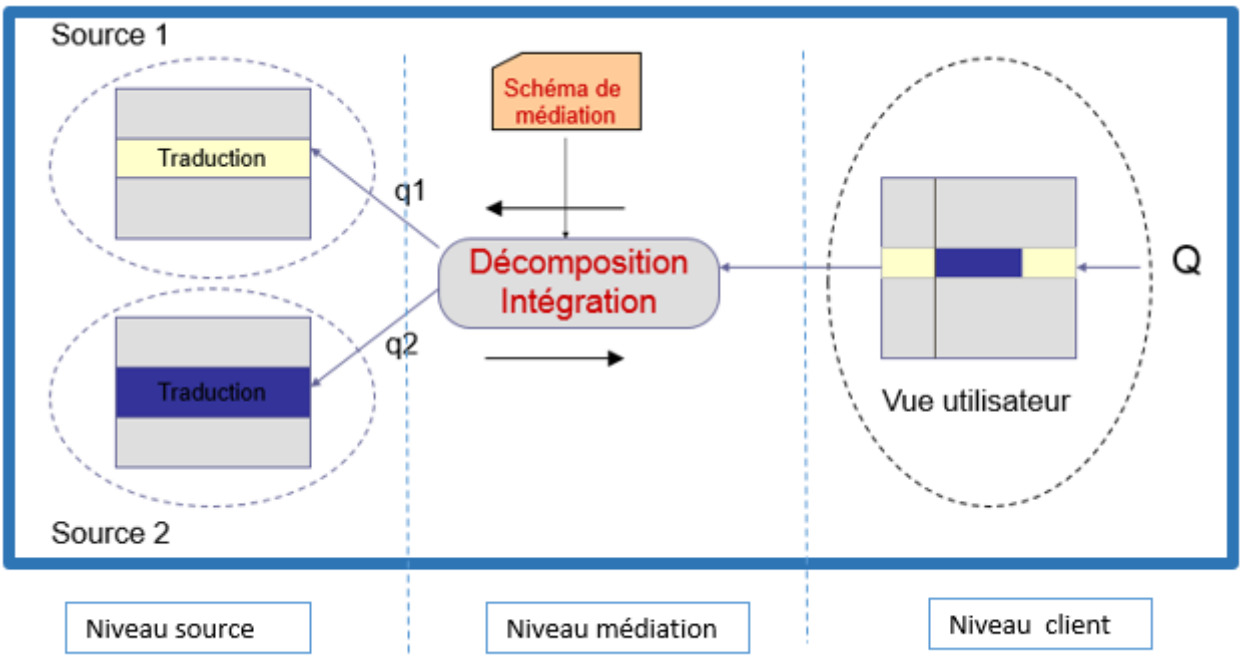

FIGURE 2.2 – Schéma de traitement de requête [3]

Niveau client

Le niveau client présente l'interface client, à partir de laquelle il envoie sa requête, qui peut être une application web ou un navigateur web par exemple.
Ce niveau a deux rôles principaux :

— Il doit assurer la transmission des requêtes au niveau de médiation.
— Il doit présenter les résultats des requêtes à l'application ou au navigateur.

Niveau médiation[4]

Ce niveau a pour but de faire le lien entre les données brutes, stockées sous différentes formes, et le client, où les données doivent pouvoir être affichées de manière structurée. Ce niveau est composé de deux éléments : le médiateur et le facilitateur.

Le médiateur s'occupe de la distribution des sources. Il possède les schémas de chacune des sources et est ainsi capable de décomposer chaque requête en sous requêtes adaptées à chacune des sources, et de les envoyer dans un langage pivot. Il est capable de recevoir des requêtes en langage pivot et de les traiter en les déléguant aux Wrappers de chaque source. Il a aussi pour rôle de résoudre les conflits de schémas. Quand les sources envoient les résultats de la requête, le médiateur permet alors de les recomposer. Le système de médiation possède aussi une mémoire cache permettant de se souvenir des requêtes déjà reçues et des pages souvent accédées.

Le niveau médiation comprend également en option un facilitateur : il s'agit d'un composant d'interface permettant de transcrire les réponses aux requêtes dans le format souhaité par le niveau client. Il peut également offrir des services de conversion syntaxique et de format.

Plusieurs schémas de médiation sont disponibles :
— Enosys : Médiateur XQuery
— Liquid Data (BEA) : Dérivé de Enosys, il offre des Vues XML/XQuery
— OLE/DB.NET : Extension de OLE/DB à XML, il permet une interrogation *SQL* pour XML

Niveau source

Le but de ce niveau est d'exporter les données stockées localement vers le niveau de médiation à l'aide d'un Wrapper (aussi appelé adaptateur).

Le Wrapper fait le lien entre la représentation locale des informations et leur représentation dans le modèle de médiation. Il a pour rôle de recevoir les requêtes dans le langage pivot et de les traduire dans le langage de requête local. Il regroupe ensuite les résultats et effectue la traduction inverse (du langage local au langage pivot).

2.1.2 Réalisations actuelles

Actuellement, IBM a conçu une norme pour les systèmes de gestion de bases de données fédérées nommée *DB2 UDB Federated System*, elle est supportée par MS SQL Server, Oracle et Sybase.
IBM DB2 offre un accès à plusieurs sources comme *XML*, *SQL* et plus. L'accès est unifié à des vues *SQL/XML* grâce à des tables virtuelles.
L'interrogation est alors en *SQL/XML*, en plus de l'utilisation des fonctions XPath avec *SQL*.
Ce système n'est pas encore commercialisé car il ne supporte pas parfaitement l'écriture et est donc, en général, en lecture seule.

2.2 Problématique

Notre projet consiste à réaliser un petit système d'information d'entreprise, capable d'interroger des bases de données hétérogènes.

Nous avons choisi de ne pas implémenter le schéma de médiation. En effet, nous supposons que la séparation de la requête initiale *XQuery* en sous requêtes adressées aux bases de données correspondantes est faite.
Nous avons décidé d'implémenter le niveau source, c'est à dire les Wrappers qui seront capables de traduire les requêtes en langage pivot (*XQuery*) vers des requêtes locales.
Nous avons choisi d'interroger deux types de base de données : *SQL* et *noSQL* dans l'environnement Windows.
Pour avoir un rendu visuel, facilitant l'utilisation de notre système, nous avons créé une interface graphique permettant la visualisation des différentes étapes et la gestion des entrées/sorties (requêtes et résultats).

2.3 Architecture du projet

La figure 2.3 résume l'architecture implémentée dans notre projet : les parties en bleu mettent en valeur le travail effectué.

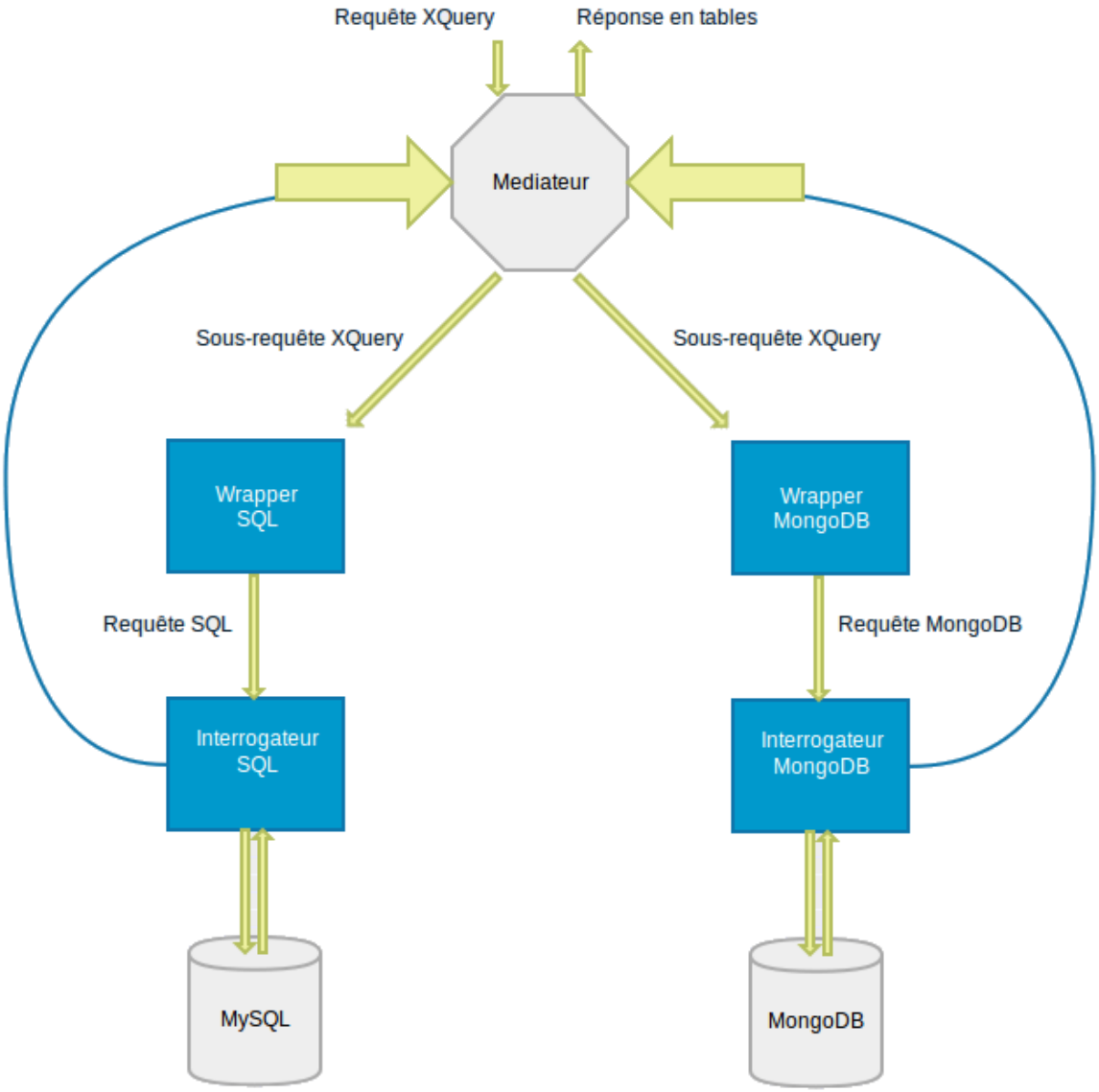

FIGURE 2.3 – Architecture global de notre solution

Les Wrappers *SQL* et *noSQL* permettent de passer respectivement d'une requête *XQuery* à une requête *SQL* ou d'une requête *XQuery* à une requête *noSQL* (*MongoDB* dans notre cas). Les interrogateurs questionnent les bases de données et retournent les réponses de leurs requêtes au médiateur qui unifie et affiche le résultat.

Dans le prochain chapitre, nous expliquons les techniques utilisées pour la réalisation des différentes étapes de la création de notre système.

3 Réalisation technique

3.1 Wrapper

Une des grande parties de ce projet a été l'élaboration d'un Wrapper permettant de transformer du XQuery en *SQL* ou *noSQL*. Cette partie du rapport explique nos choix techniques et notre réalisation concernant le Wrapper.

3.1.1 Contraintes initiales

Comme il est difficile de transformer toutes les formes de requêtes *XQuery* vers des requêtes *SQL* ou *noSQL*, nous avons décidé de restreindre l'ensemble de requêtes à analyser au type *FLWOR*. En effet, ce type de requête *XQuery* peut être traduit en une requête *SQL* de type «*Select Where*» en utilisant éventuellement des jointures, et une requête *noSQL* de type «*db.table.find({ Where })*». Faute de temps, nous nous sommes concentrés sur les parties *For* et *Where* en prenant compte des jointures.
Dans un fichier XML, la clause *For* a pour but de sélectionner un ou un ensemble d'élements. Comme le langage XML ne peut pas être totalement représenté en base de données relationnelle ou non relationnelle (*NoSql*), nous avons supposé que tous les éléments qui viennent après «*document("nom")*» sont des tables (collections) hormis le dernier qui peut-être soit une table, soit une colonne (champ).

3.1.2 Outils utilisés

JavaCC

Java Compiler Compiler (*JavaCC*) est un générateur d'analyseur syntaxique. Nous avons choisi de l'utiliser car il est adapté pour les applications Java.
En plus du générateur d'analyseur lui-même, *JavaCC* fournit d'autres capacités standards liées à la génération de l'analyseur, comme la construction de l'arbre (via un outil appelé JJTree inclus avec *JavaCC*), les actions et le débogage.

JavaCC prend en entrée un fichier d'extension «jj» qui contient entre autres les descriptions des règles de la grammaire et il produit un parser descendant (dans ce fichier).
JavaCC produit 7 fichiers Java en sortie, la figure 3.1 montre les différentes classes générées :

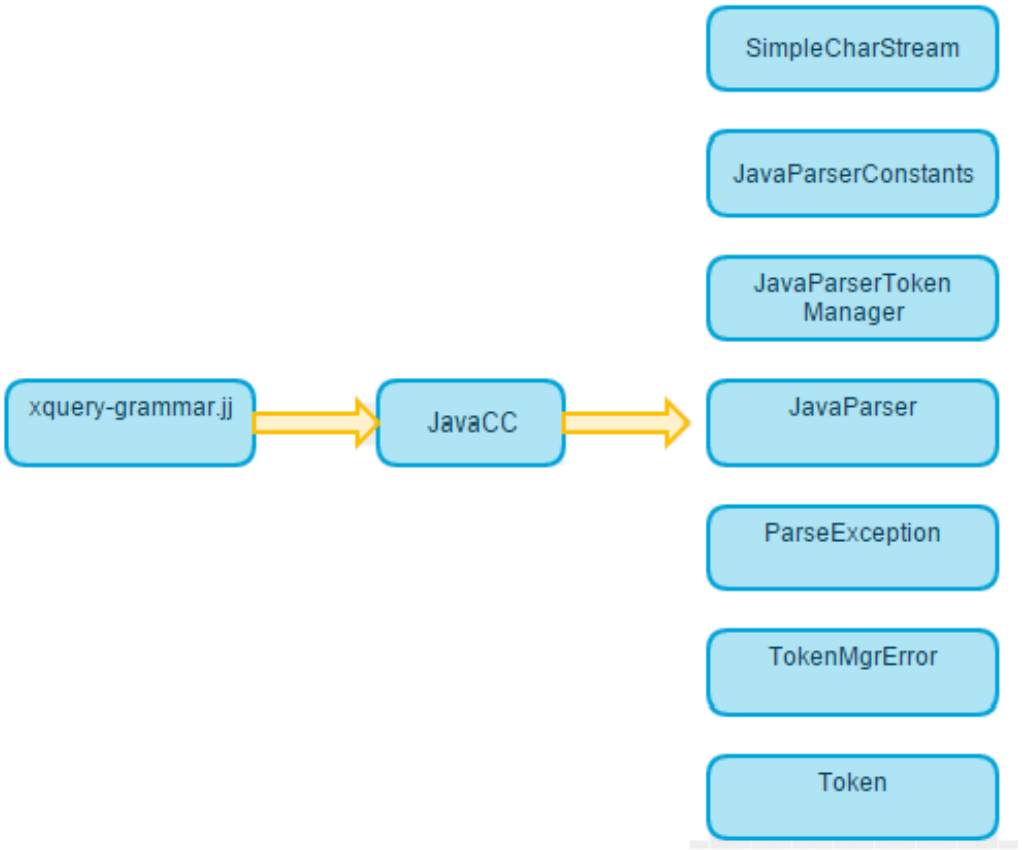

FIGURE 3.1 – javaCC

Pour notre projet, nous travaillons avec le fichier «xquery-grammar.jj» qui décrit les règles de grammaire des requêtes *XQuery* et définit les tokens.

Afin de pouvoir transposer nos requêtes *XQuery* en un autre langage, il est important de posséder la grammaire permettant d'analyser *XQuery*.
Notre choix s'est porté sur la grammaire fournie par W3C [5]. W3C est un organisme de standardisation à but non lucratif, fondé en octobre 1994 et chargé de promouvoir la compatibilité des technologies du *World Wide Web*. La figure 3.2 montre une partie de la grammaire que nous utilisons pour parser XQuery.

```
1  FLWRExpr      ->      ((ForClause |  LetClause)+ WhereClause? OrderByClause? "return")* QuantifiedExpr
2  ForClause     ->      <"for" "$"> VarName TypeDeclaration? PositionalVar? "in" Expr ("," "$" VarName TypeDeclaration? PositionalVar? "in" Expr)*
3  LetClause     ->      <"let" "$"> VarName TypeDeclaration? ":=" Expr ("," "$" VarName TypeDeclaration? ":=" Expr)*
4  WhereClause   ->      "where" Expr
```

FIGURE 3.2 – Partie de la grammaire concernant les For Let et Where

JOOQ et MongoDB API

Java Object Oriented Querying (JOOQ)[6], est un outil de création de requête *SQL* en JAVA. MongoDB API [7] est l'équivalent de JOOQ pour *noSQL*. Ces outils nous ont simplifié le développement durant la création du compilateur.

3.1.3 Méthode d'analyse

Comme cité précédemment, nous avons utilisé le fichier « *jj* » du W3C contenant la grammaire. Afin de traduire une requête *XQuery* en *SQL* et *noSQL*, nous avons ajouté des méthodes d'analyse dans chaque partie, à savoir des méthodes de reconnaissance du type d'une variable ainsi que des méthodes d'évaluation des expressions et de la gestion des ordres des conditions.

Clause For

Afin de traduire la clause *For* en *SQL(noSQL)* , nous avons associé, en utilisant deux éléments de hachage, chaque variable déclarée après *document("nom")* à une table et/ou une colonne pour pouvoir l'utiliser dans la clause *Where*. En effet, si le dernier élément qui vient après «*document("nom")* » est une colonne (champ), elle doit être liée à la table qui la précède. De plus chaque table (collection) doit être reliée par une clé étrangère à la table (collection) qui la précède. Concernant l'exemple 3.5, la table *Formation* est reliée à la table *Personne* à travers la clé étrangère IDFormation. L'exemple montre aussi l'utilisation du *Builder SQL JOOQ* qui joue le rôle de l'intermédiaire entre *XQuery* et *SQL*. En effet, grâce au *Builder SQL/noSQL*, la génération de code est plus facile puisqu'il suffit de faire appel à des méthodes faisant référence aux mots clés utilisés dans les langages *SQL/noSQL*.
Concernant les jointures ("," pour *XQuery*), la gestion de l'expression *On* a été reportée vers *Where* puisque *XQuery* ne gère pas cette partie.

Clause Where

La clause *Where*, quant à elle, a pour objectif de filtrer les données sélectionnées par la clause *For*. Comme les conditions utilisées dans cette clause sont ordonnées, nous nous sommes inspirés d'une implémentation d'un parser pour calculatrice et avons donc utilisé deux piles, l'une pour stocker les opérateurs et l'autre pour stocker les expressions de type «*A op B*». Ainsi, à chaque fois qu'une expression de type «*A and/or B*» est lue, les deux dernières expressions et le dernier opérateur sont dépilés, puis grâce au *Builder SQL/noSQL* que nous avons utilisé, la condition est générée puis empilée.

Diagramme de classe

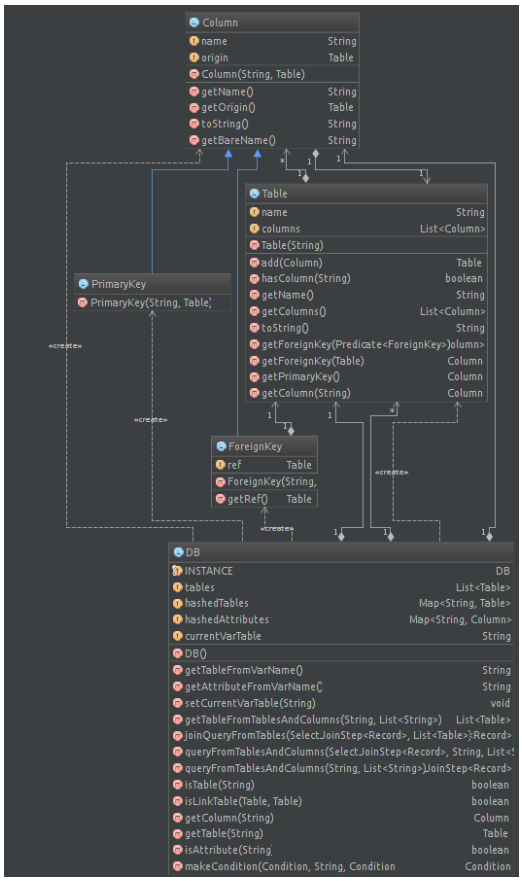

FIGURE 3.3 –

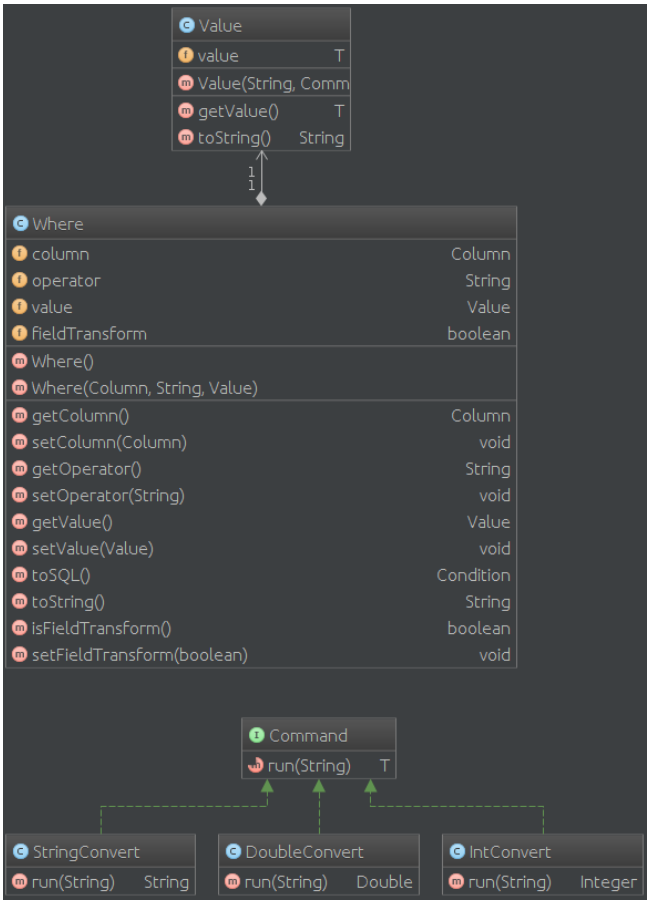

FIGURE 3.4 –

Les diagrammes de classes 3.3, 3.4 décrivent les classes utilisées pour assurer la traduction du code XQuery en SQL. En effet, la classe *DB* est un singleton qui contient les tables qui sont dans la base de données SQL (*tables*), l'association d'une variable XQuery avec une table SQL (*hashedTables*) et l'association d'une variable XQuery avec une colonne SQL (*hashedAttributes*). L'attribut *currentVarTable* représente la dernière variable XQuery lue.

La classe *Table* représente une table *SQL*, elle est ainsi composée d'un nom et d'une liste de colonnes. La classe *Column* représente une colonne SQL, elle peut être soit une clé primaire (*PrimaryKey*), une clé étrangère *ForeignKey* ou une colonne normale. La gestion de la clé primaire et la clé étrangère est crucial lors de l'évaluation des jointures (cf. sous-section 3.1.1).

La classe *Where* assure la traduction d'une condition Xquery en SQL. Cette dernière est composée d'une colonne, d'un opérateur et d'une valeur. La valeur est attribuée à une classe *Value* puisqu'il existe plusieurs types de valeurs (ici, *String*, *Double*, *Integer*, *Field*). Afin de convertir la valeur au format adéquat, nous utilisons des commandes qui sont directement injectées lors de la traduction. Afin de joindre deux conditions par un opérateur «*and*» ou «*or*», il suffit de faire appel à la méthode *makeCondition* de la classe *DB*.

Quelques exemples de l'utilisation du Wrapper

— **Traduction *XQuery-SQL***

```
for $a in document('personnes')//formations where $a/nom='INFO' return $a
select(field("*")).from(table("Personnes"))
                  .join(table("Formations")).on(field("Personnes.IDFormation"), field("Formations.IDFormation"))
                  .where(field("Formations.nom").eq("INFO"))
                  .getSQL()
Select * from Personnes join Formation on Personnes.IDFormation=Formation.IDFormation where Formations.nom='INFO'
```

FIGURE 3.5 – Exemple de traduction *XQuery-SQL*

— **Traduction *XQuery-noSQL***

```
for $a in document('formations')//formations where $a/nom='INFO' return $a
"db.Formations.find(" + queryBuilder.put("nom").is("INFO").get() + ")"
db.Formations.find({ nom: 'INFO' })
```

FIGURE 3.6 – Exemple de traduction *XQuery-noSQL*

3.2 Interrogation des bases de données

À la suite de la conversion de la requête initiale par le Wrapper, la requête est envoyée au SGBD concerné. Cette section décrit l'envoi et la réception de la requête vers les SGBD.

3.2.1 Outils utilisés

Pour choisir les SGBD [1] de l'application, des SGBD gratuits, et très utilisés aujourd'hui ont été choisis. Conformément à cette idée, le SGBD *MySQL* pour le langage *SQL*, et *MongoDB* pour le langage *noSQL*, se sont distingués.

Plusieurs systèmes permettent de gérer des bases de données à partir de Java. Le framework *Hibernate* et l'interface de programmation *JDBC* [2] recouvrent ce besoin.
Sachant que l'utilisation des bases de données se limite seulement à du requêtage, sans modification de la base pendant l'utilisation, notre propre interface a été implémentée sans utiliser de grosses briques de technologies existantes. Cette implémentation génère des fichiers de compilation «*.m*» qui contiennent les requêtes. Ces fichiers sont envoyés au SGBD via l'invite de commande Windows, puis éliminés ensuite par défaut.

Ainsi, le contrôle du processus d'accès aux bases de données est total. Par exemple, il est possible d'utiliser une base de données initialisée en dehors de l'application. Dans certains cas, l'utilisateur a déjà initialisé ses bases de données en utilisant le SGBD en dehors de l'application. Il n'aura pas besoin de refaire les opérations nécessaires à son chargement car notre application chargera les bases de données déjà présentes.

De même, l'utilisateur peut choisir s'il souhaite supprimer ou réutiliser les bases de données utilisées par l'application en quittant l'application. L'utilisateur a aussi la possibilité de choisir le nom de la base de données où il travaille, et de garder les fichiers de compilation, pour lui permettre de retrouver ses données facilement s'il le souhaite à la fin du traitement. C'est pour favoriser l'utilisateur qu'il a été décidé de développer entièrement cette interface.

L'application a été testée avec Windows, mais développée en permettant une utilisation sur Linux. Les tests n'ont malheureusement pas été effectués, connaissant la longueur du projet mais des lignes de log sont imprimées dans la sortie courante pour permettre un débogage plus facile.

3.2.2 Architecture

L'application doit aussi permettre la distribution des requêtes aux SGBD. Le Wrapper envoie les requêtes à l'interrogateur. Ces derniers transfèrent les requêtes et reçoivent les résultats de celles-ci. La classe *DatabaseManager.java* permet de gérer l'initialisation des bases de données via les options du fichier de configuration [3]

1. Système de gestion de base de données
2. Java Database Connectivity
3. Voir sous section Le fichier de configuration

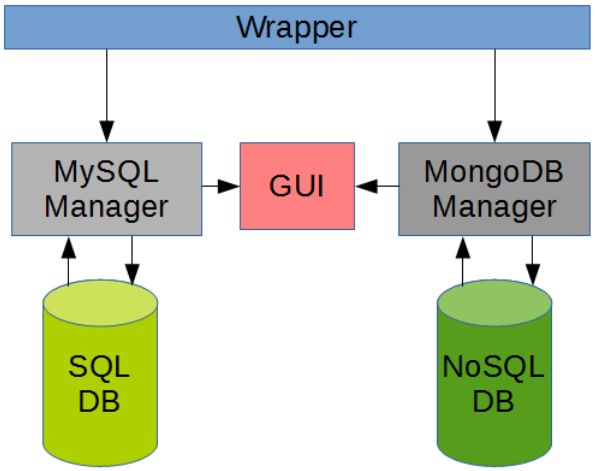

FIGURE 3.7 – Architecture de l'interrogateur

En envoyant une requête *SQL*, le Wrapper utilise la classe *MySQLManager.java*. De même, pour envoyer une requête *noSQL*, le Wrapper utilise la classe *MongoManager.java*. Ces classes gèrent les initialisations de tables si elles sont requises, ainsi que l'envoi et la réception de requêtes. La classe *PropertiesHandler.java* permet de gérer un fichier de configuration *databaseFiles.properties*, qui propose un éventail d'options à l'utilisateur.

3.2.3 Le fichier de configuration

Pour rendre le système facilement modifiable par l'utilisateur, un fichier de configuration *databaseFiles.properties* a été ajouté. Ce fichier contient une liste de paramètres modifiables, tel que le chemin d'accès au répertoire des bases de données utilisées par l'application. Deux chemins sont fournis, permettant l'usage de l'application sur une machine Unix, l'application choisissant uniquement le premier chemin existant sur la machine .

Ce fichier contient aussi les fichiers utilisés pour l'initialisation rapide des bases de données, et leur type, *SQL* ou *noSQL*. Dans cette version de l'application ,trois fichiers sont seulement accessibles mais il est assez facile d'ajouter d'autres fichiers à initialiser dans le fichier de configuration et la classe qui le gère, *PropertiesHandler.java*. D'autres paramètres cités auparavant sont accessibles, tel que le nom des bases de données de chaque SGBD, l'option de suppression des fichiers de compilation, et l'option de nettoyage de la base de données à l'arrêt de l'application.

3.2.4 Interface avec les SGBD

Les classes*MySQLManager.java* et *MongoManager.java* sont l'interface avec les SGBD, elles gèrent l'envoi et la réception de données.

La réception et l'émission des requêtes se faisant par la réception de données via le terminal et non par la réception à l'intérieur de celui-ci, les réponses des SGBD ne sont pas formatées correctement. En particulier, pour le SGBD *MongoDB*, la fonction *find()* ne rend qu'un itérateur. Il est donc indispensable d'itérer à travers la réponse, mais cette itération ne permet pas de disposer d'un formatage correct.

Ces classes parsent donc le résultat pour le rendre disponible sous forme de *String[][]*. La première ligne du tableau stocke les attributs de la solution alors que les lignes suivantes stockent

les données. Si le SGBD ne renvoie pas de réponse, ces classent renverront un tableau nul. Pour terminer, l'interface graphique affiche les résultats pour l'utilisateur. Cette dernière est décrite dans la prochaine partie.

3.3 Interface graphique

Pour l'interface, nous avons choisit une solution Java avec le framework *JavaFX* pour réaliser une interface souple et modulable. L'interface a été réalisée à l'aide du logiciel *JavaFX Scene Builder* et se compose de 2 parties principales séparées chacune sur un onglet différent : l'Input et l'Output.

3.3.1 Input

La partie Input [4] est découpée en différentes sections, la première est en fait le champ texte où l'utilisateur entre sa requête *XQuery* et la soumet au système.

La deuxième section correspond à la représentation du médiateur qui va séparer la requête *XQuery* en différentes requêtes *XQuery* pour aller interroger les différentes bases de données concernées par la requête originale. Dans cette deuxième section, nous avons aussi la possibilité de préciser manuellement une requête à envoyer directement sur un Wrapper en particulier.

La troisième section correspond à la représentation du Wrapper de chaque base de données, qui parse la requête *XQuery* en entrée pour la transformer en une requête directement compréhensible par la base de données elle-même.

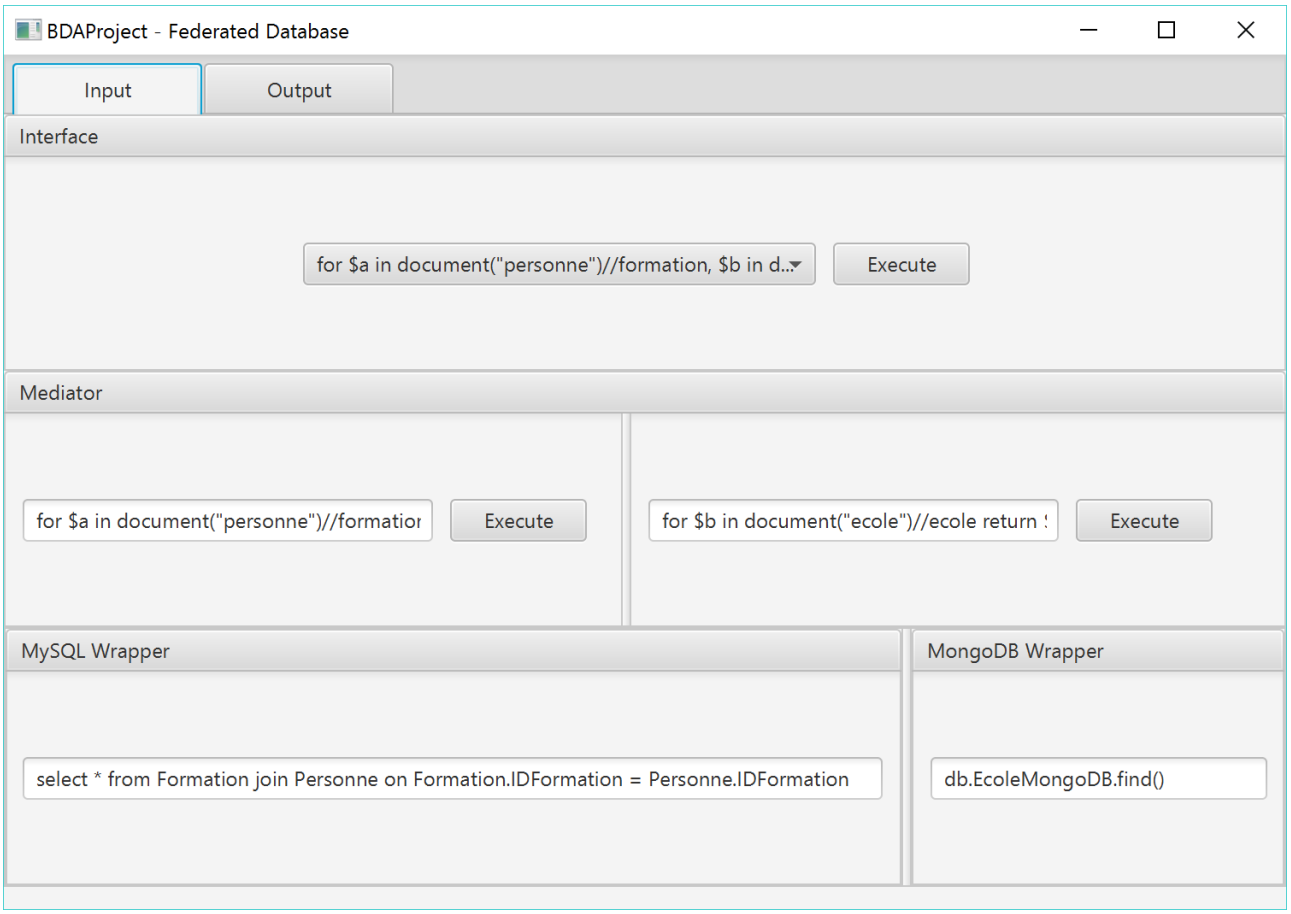

FIGURE 3.8 – Copie d'écran de l'Input

4. La saisie en entrée

3.3.2 Output

Une fois les différentes requêtes générées, la requête de chaque Wrapper est exécutée sur sa base de données et le résultat est rendu. La première partie de l'Output [5] correspond donc aux données fournies séparément par chaque base de données (ici *MySQL* et *MongoDB*).

La deuxième partie, quant à elle, correspond aux résultats de chaque base de données interrogées, recomposés par le médiateur.

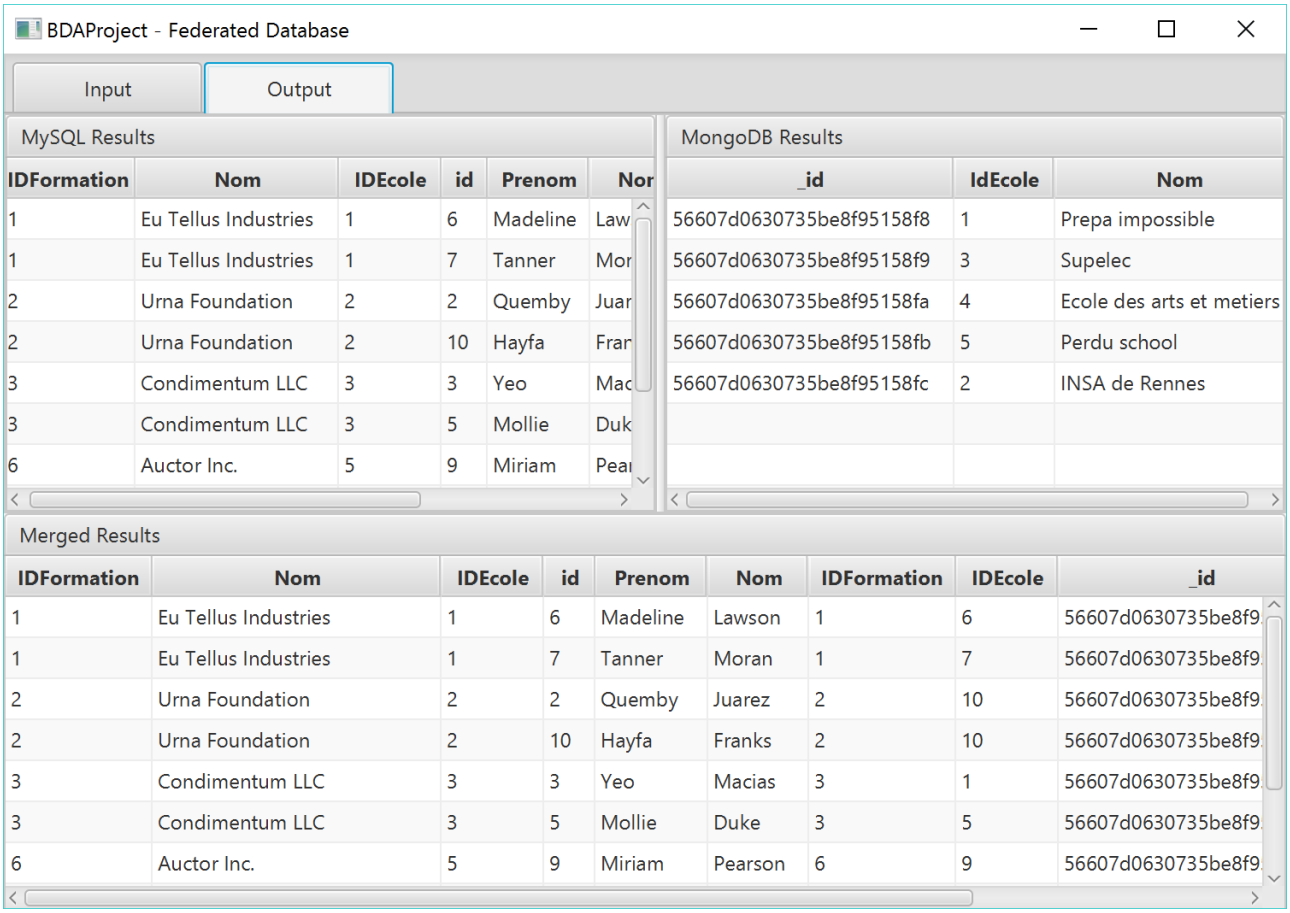

FIGURE 3.9 – Copie d'écran de l'Output

3.3.3 Fonctionnement

L'interface est simple d'utilisation, tout d'abord l'utilisateur configure sa requête sur l'onglet Input et ensuite il visualise les résultats sur l'onglet Output.

Pour configurer sa requête l'utilisateur peut utiliser l'Input de deux façons, la première en choisissant une des requêtes *XQuery* prédéfinies dans la liste déroulante et en l'exécutant pour voir les différentes étapes de génération de requêtes ; ou bien en écrivant à la main une requête *XQuery* directement dans le champ qui concerne une base de données précise.

5. Le résultat en sortie

4 Analyse de notre solution

Dans ce chapitre, notre solution est analysée en insistant sur les outils utilisés. Concernant les points négatifs, des améliorations seront proposées.
Dans la prochaine section est présentée l'analyse de l'architecture de la solution.

4.1 Architecture

Le premier objectif du projet était de respecter l'architecture d'une base de données fédérée. Le temps ne nous a pas permis d'implémenter entièrement le médiateur, mais son comportement a été simulé pour permettre le bon fonctionnement de l'application. Par conséquent, la réalisation du médiateur reste un axe de travail en vue de l'amélioration de l'application.

Le *Wrapper* a été implémenté, convertissant les requêtes *XQuery* en requêtes *noSQL* et *MySQL*. Ce Wrapper fonctionne de manière correcte mais ne traite qu'un nombre limité de requêtes, dû à la complexité du langage *XQuery* et à la durée limitée du projet. Cette limitation faisait partie du cahier des charges initial du projet. Par conséquent, l'évolution du Wrapper et de ses grammaires pour prendre en charge un plus grand nombre de syntaxe possibles est un point d'amélioration crucial.

Le fichier de configuration est une des réussites du projet. En effet, ce fichier permet de contenter plusieurs types d'utilisateurs tout en évitant d'opérer à de gros changements sur l'application. Il est ainsi possible de choisir le nom de sa base de données et ainsi réutiliser une base de données existante, ou de repartir de zéro. Il est aussi possible de conserver les fichiers de compilation de l'interrogateur à des fins de débogage. L'utilisateur détient le choix de l'utilisation du programme et c'est ce qui fait la différence.

4.2 Méthode de Travail

Le deuxième objectif principal était de fournir une application fonctionnelle, répondant aux objectifs de requête sur des bases de données hétérogènes.

Cet objectif a été atteint. Le choix de 2 SGBD les plus utilisés au monde *Mysql* et *MongoDB* nous a permis de rendre notre application polyvalente par rapport au besoin. Ainsi, un utilisateur sera plus intéressé par notre application car elle reprend les technologies du marché.

Notre projet a été réalisé avec le modèle du Cycle en V, tout en ayant distribué les responsabilités à toute l'équipe de développement. Les premières réunions nous ont permis de fixer

le cahier des charges du projet, et de répartir les tâches entre les membres de l'équipe. Après développement, des tests fonctionnels ont été effectués pour vérifier l'implémentation. Ensuite, les modules ont été intégrés et testés un à un pour nous permettre de gagner du temps sur le débogage. Le rapport a été écrit en respectant les règles de typographie[8].

4.3 Outils utilisés

Le langage de développement Java a été choisi pour sa simplicité et sa polyvalence. Les IDE [1] *Eclipse* et *IntelliJ*, compatibles avec Java ont été utilisés. *Maven* a été utilisé pour importer des dépendances dans le Wrapper, ainsi que *JavaCC* pour la compilation des fichiers. La grammaire du site W3C a été réutilisée pour faire le le Wrapper, et l'outil *Jooq*, respectivement *QueryBuilder* pour le langage *noSQL* a été utilisé pour créer les requêtes de bases de données à la sortie du Wrapper.

L'utilisation du logiciel *JavaFX Scene Builder* pour l'implémentation de l'interface graphique en Java a fortement accéléré le développement de l'Interface Homme Machine. L'interface est intuitive, elle montre toutes les étapes majeures de l'architecture de l'application tout en restant simple.
L'utilisateur n'est pas perdu, la sobriété de l'interface graphique lui permettra de trouver les commandes importantes rapidement. Le fait que le fichier de configuration ne soit pas accessible via l'IHM [2] n'empêche pas l'utilisateur de le modifier soi-même en dehors de l'application, mais permet aux utilisateurs lambda de ne pas avoir à interagir avec cette partie directement. L'usage des paramètres par défaut simplifie l'expérience utilisateur.
Ce logiciel est facilement utilisable, et constitue une des technologies importantes du projet.

Plusieurs outils dans la partie Interrogateur du projet auraient pu être utilisés, tel que le framework *Hibernate*, ou la technologie *JDBC*. Le projet étant limité dans le temps et dans le contenu des requêtes *XQuery*, nous avons décidé ne pas réutiliser de technologies existantes pour éviter d'utiliser de grands moyens pour un projet modeste. Nous avons profité de cette décision en ajoutant le fichier de configuration nous permettant d'améliorer l'expérience utilisateur.

De même, nous avons fait le choix de ne pas utiliser de bases de données sur un serveur. Aujourd'hui, cette technologie est très répandue mais nous avons considéré que le terme « fédéré » implique une fédération de différents types de base de données en priorité, et non leur utilisation sur un serveur. C'est pourquoi nous avons choisi différents SGBD.

Cependant, en vue d'une probable amélioration de l'application, il est possible d'ajouter une classe gérant la connexion avec un serveur extérieur, et d'utiliser cette connexion pendant l'initialisation de l'application. Ainsi, l'utilisation de bases de données fédérées sur un serveur devient possible.

Finalement, notre projet a atteint ses objectifs. Les technologies du projet ont été bien utilisées et ont permis de mener à bien le développement de la solution.

1. Integrated Development Environment ou Environnement de développement intégré
2. Interface Homme Machine

5 Conclusion

Durant le cours de base de données avancées et durant ce projet, nous avons pu appréhender et comprendre les principes d'une base de données fédérée. L'objectif d'une base fédérée est de donner aux utilisateurs un outil permettant d'interroger plusieurs systèmes à priori hétérogènes et d'unifier le résultat.

Dans ce rapport, nous avons d'abord introduit le principe d'une base de donnée fédérée et aussi introduit l'objectif de notre projet : la réalisation d'un petit système d'information d'entreprise.

Ensuite, nous avons détaillé nos choix techniques d'implémentation. En effet, nous avons implémenté la partie source de l'architecture fédérée permettant de traduire un langage (ici, *XQuery*) en un autre langage compréhensible par le SGBD (ici, *SQL* et *noSQL*) mais également l'interrogation des bases de données.

De plus, nous avons créé une interface graphique afin d'illustrer nos propos et de faire comprendre le fonctionnement de notre solution.
Enfin, nous avons présenté des analyses sur les résultats obtenus.

Nous avons mené à bien le projet, malgré l'omission ou la réduction de quelques problèmes, tels que le médiateur à cause de la longueur du projet.

Nous pouvons donc dire, que les bases de données fédérées étant donné leur complexité, sont peu courantes sur le Web (en terme de solutions) mais beaucoup développées et utilisées en entreprise.

En ouverture, nous voulons préciser que des améliorations sont possibles, en particulier la partie serveur et médiateur qui rendrait notre application plus polyvalente dans le milieu de l'entreprise.

Bibliographie

[1] M. D. Heimbigner, D., "A federated architecture for information management. acm transactions on office information systems (tois)," vol. 3, pp. 253–278, 1985.

[2] L. J. Sheth, A.P., "Federated database systems for managing heterogeneous, distributed and autonomous databases. acm computing surveys," vol. 22, 1990.

[3] "Business integration." http://slideplayer.fr/slide/1298796/.

[4] "Gestion de données réparties." http://mon.univ-montp2.fr/claroline/backends/download.php?url=L0NvdXJzL0NPVVJTX0ludOlncmF0aW9uLnBkZg%3D%3D&cidReset=true&cidReq=GMIN332.

[5] "Grammaire xquery." http://www.w3.org/2007/01/applets/xquery-grammar.jj.

[6] L. Eder, "Java object oriented querying." http://www.jooq.org.

[7] M. Inc., "Mongodb api for java." http://api.mongodb.org/java/current/com/mongodb/QueryBuilder.html.

[8] J. André, "Petite leçon de typographie." http://jacques-andre.fr/faqtypo/lessons.pdf.