

UNIVERSITÉ DE MONTPELLIER
M1 ARCHITECTURE DU GÉNIES LOGICIEL ET DU WEB

Data'Clear

RAPPORT DE PROJET TER
PROJET INFORMATIQUE — HMIN601

Étudiants :

Denis BEAUGET Hayaat HEBIRET

Année : 2020 – 2021

Encadrant :

Federico ULLIANA, Florent TORNIL



UNIVERSITÉ
DE MONTPELLIER



Table des matières

| | |
|--|-----------|
| Remerciements | 3 |
| 1 Introduction | 4 |
| 1.1 Résumé | 4 |
| 1.2 L'organisation du travail | 5 |
| 1.3 Outils de travail collaboratif | 5 |
| 2 Analyse et mise en place | 6 |
| 2.1 Problématique | 6 |
| 2.2 Article et méthode | 6 |
| 2.3 Solution proposée | 8 |
| 2.4 Cas d'étude : prédiction de survie dans le Titanic | 10 |
| 2.5 Technologies | 12 |
| 2.6 Base de données | 12 |
| 2.6.1 SQLite | 12 |
| 2.6.2 MongoDB | 13 |
| 3 Conception et mapping | 14 |
| 3.1 Conception | 14 |
| 3.1.1 Conception : UML | 14 |
| 3.1.2 Introduction au mapping via SQLite | 15 |
| 3.1.3 Introduction au mapping via Graal | 15 |
| 3.2 Mapping complet via Graal | 16 |
| 3.2.1 Fonction de mapping générique SQLite | 16 |
| 3.2.2 Fonction de mapping générique MongoDB | 18 |

| | | |
|----------|------------------------------------|-----------|
| 4 | Résultat et extraction | 20 |
| 4.1 | Pourquoi | 20 |
| 4.2 | Méthode 1 : Py4J | 21 |
| 4.3 | Méthode 2 : Les fichiers | 22 |
| 4.4 | 2 méthodes ? | 23 |
| 5 | Analyse avec Python | 24 |
| 5.1 | Analyse | 24 |
| 5.2 | Résultat | 25 |
| 6 | Bilan | 27 |
| 6.1 | Et après ? | 27 |
| 6.2 | Conclusion | 28 |
| | Annexes | 30 |

Remerciements

Nous aimerions remercier toutes les personnes ayant pu rendre ce projet possible.

- Merci à Federico Ulliana pour avoir accepté le projet et avoir su comprendre et mettre en place un véritable cadre à notre idée.
- Merci à Florent Tornil pour sa patience et le temps qu'il a accordé à toutes nos questions et, sans qui, Graal serait encore pour nous un obscure objet mythique de la légende arthurienne.

*"La science est un projet coopératif qui se transmet entre les générations. C'est le relais d'une torche du professeur à l'étudiant au professeur. Une communauté d'esprits prenant racine dans l'Antiquité et se dirigeant vers les étoiles.
- Neil deGrasse Tyson"*

Chapitre 1

Introduction

1.1 Résumé

Dans le cadre du projet Travaux d'Étude et de Recherche de la première année de master informatique, nous avons décidé de proposer notre propre idée de projet sur le thème du "Parcours de la donnée", thème que nous avons abordé dans certaines UE comme "HMIN122M - Entrepôts de données et Big-Data".

Data'Clear a pour objectif de répondre à des problématiques actuelles dans le domaine de la science des données. Plusieurs thèmes sont abordés, comment obtenir des données pour supporter un processus d'analyse ? Comment réaliser des déductions et prendre des décisions en fonction des résultats ?

D'un point de vue plus général, l'utilisateur doit pouvoir comprendre les outils et les différentes manipulations que nous introduisons dans notre projet pour pouvoir se concentrer sur les résultats obtenus et en comprendre les déductions.

Dans ce TER, nous avons conçu et mis en oeuvre un cadre d'*intégration de données hétérogènes* (plus précisément, au format relationnel SQL et au format NoSQL JSON) supportant un *processus décisionnel*. Nous avons étudié et étendu la bibliothèque logicielle Java Graal¹ avec tous les objets et algorithmes nécessaires à l'intégration de données hétérogènes. La famille des processus décisionnels que nous avons visé nécessite l'utilisation de bibliothèques d'apprentissage automatique. Ainsi, nous avons établi un lien avec la bibliothèque Python Scikit-learn² et notre framework Java. La solution d'intégration et d'analyse de données mise en place a été validée sur une tâche d'apprentissage classique : la prédiction de survie des passagers du Titanic [htt16]. Avec finalement un démonstrateur réalisé avec Jupyter³.

-
1. Graal : <https://graphik-team.github.io/graal/>
 2. Scikit-Learn : <https://scikit-learn.org/stable/>
 3. Jupyter : <https://jupyter.org/index.html>

1.2 L'organisation du travail

Depuis le début de la pandémie nous sommes en binôme dans l'intégralité des UE le proposant. Nous avons une dynamique de travail similaire et nous avons su développer nos méthodes à distance pour avancer ensemble.

DataClear était pour nous une forme de confirmation, nous devions être capables de faire un projet de A à Z, plus conséquent que les précédents à travers les outils mis à notre disposition en respectant un maximum nos échéances de travail et fournir le travail nécessaire au bon déroulement du projet.

Notre première initiative a été de décider qu'un seul de nous deux poserait des questions ou communiquerait par mail avec les encadrants du projet. De ce fait, si l'un de nous avait une question il en référait d'abord à l'autre pour se mettre d'accord d'un mail commun à envoyer.

DataClear est un projet qui introduit des principes et des outils nouveaux pour nous, de ce fait, il était nécessaire pour nous de bien assimiler les objectifs et les exigences techniques liés au sujet.

Chaque étape du projet étant encadrée par des réunions régulières.

Ces réunions étaient prévues toutes les 2 semaines en moyenne avec les encadrants M.Ulliana et M.Tornil. A ceci, nous avons ajouté des réunions périodiques à deux. Celles-ci se déroulaient tous les 4 jours environ et permettaient d'avoir un suivi du travail effectué par chacun, de donner son avis et de pouvoir modifier le travail de l'autre en cas de besoin pendant le projet.

1.3 Outils de travail collaboratif

La situation nous a obligées à revoir nos méthodes de travail pour s'adapter à la distance. Notamment sur la façon de se répartir le travail et sur les outils à utiliser pour arriver au mieux à collaborer.

- Github,⁴ incontournable outil de versioning. Nous avons décidé d'y mettre l'intégralité du projet, notamment les fichiers PDF, les rapports, les plannings (Gantt..) ainsi que les fichiers contenant nos codes sans y intégrer tous les projets. L'objectif était un git épuré compréhensible pour le plus grand nombre avec le code associé pour les plus intéressés.
- Discord, outil de communication écrit et vocal permettant le partage d'écran et la création de "salon" personnalisés. Il nous a permis de créer notre propre espace de travail dédié au projet et à partager rapidement nos ressources ou nos questions entre nous et ainsi rester en contact constant.
- Pour la modélisation de nos bases de données nous avons utilisé un petit outil en ligne que nous voulions partager ici car il est open source, il s'agit de Mocodo⁵
- Pour la communication totale du groupe nous avons utilisé conjointement : Un Pad textuel proposé par Mr.Ulliana dans lequel nous écrivions nos comptes-rendus de réunion et la date des prochaines ainsi que tout élément intéressant (lien externe, ressources pédagogique) pour le projet. Et pour nos visioconférences, nous avons utilisé BBB.⁶

4. URL du Github <https://github.com/Beauget/Data-Clear>

5. URL vers Mocodo : <http://mocodo.wingi.net/>

6. Lien vers BBB : <https://bigbluebutton.org/>

Chapitre 2

Analyse et mise en place

2.1 Problématique

La problématique de Data'Clear est double. D'une part nous devons être capables de récupérer et de traiter plusieurs types de données sous des formats différents (en effet l'hétérogénéité des données est introduite par différents types de fichier : .csv, .json...) mais aussi de créer un modèle ontologique permettant de donner un cadre à nos données. D'autre part, l'extraction de nos résultats pour pouvoir faire des analyses précises via différents outils de la science des données. Ces 2 axes principaux de travail avaient leur propre problématique associé. Quels types de SGBD utilisés pour les données ? Comment prendre en compte les différences au sein des données ? Mais aussi, comment extraire nos données pour les utiliser dans un autre environnement et que le tout soit compréhensible pour un utilisateur au bout de la chaîne de travail ? C'est à toutes ces questions que DataClear répond à travers ce rapport.

2.2 Article et méthode

Dans la première partie de notre projet, il était important pour notre encadrant que nous comprenions ce que nous allions faire, notamment la notion de “mapping” (Le mappage des données consiste à mettre en correspondance les champs de plusieurs bases de données. C'est la première étape pour faciliter la migration des données, l'intégration des données et d'autres tâches de gestion des données comme les futures analyses) apparaissant dans le schéma du projet qui était pour nous le point nouveau. Nous devons également nous familiariser avec les données choisies et surtout avec le type de schéma que nous devons mettre en place pour le mapping des données.

Il nous a donc proposé 2 articles à lire pour comprendre ces notions et les techniques que nous allions utiliser [htt21],

Le premier article nous a permis de découvrir toutes les problématiques induites par des sources de données hétérogènes, plus précisément, comment l'extension d'un fichier et les différences syntaxiques pouvaient impacter l'intégralité du schéma à mettre en place, mais aussi d'appréhender un point important pour la suite du projet, la technique de *mapping* introduite dans cet article que nous allions utiliser dans notre projet, GAV pour “Global as View”.

GAV est une méthode qui fait partie d'une technique d'intégration plus large des données appelé “Médiateur” qui consiste à une intégration virtuelle des données qui s'oppose à l'intégration matérialisée des données (plus connu puisque c'est le principe des entrepôts de données).

Le principe du *Médiateur* se regroupe en 3 points :

- Les données restent dans les sources.
- Les requêtes sont exprimées sur le schéma global, puis décomposées en sous-requêtes sur les sources.
- Les résultats des sources sont combinés pour former le résultat final.

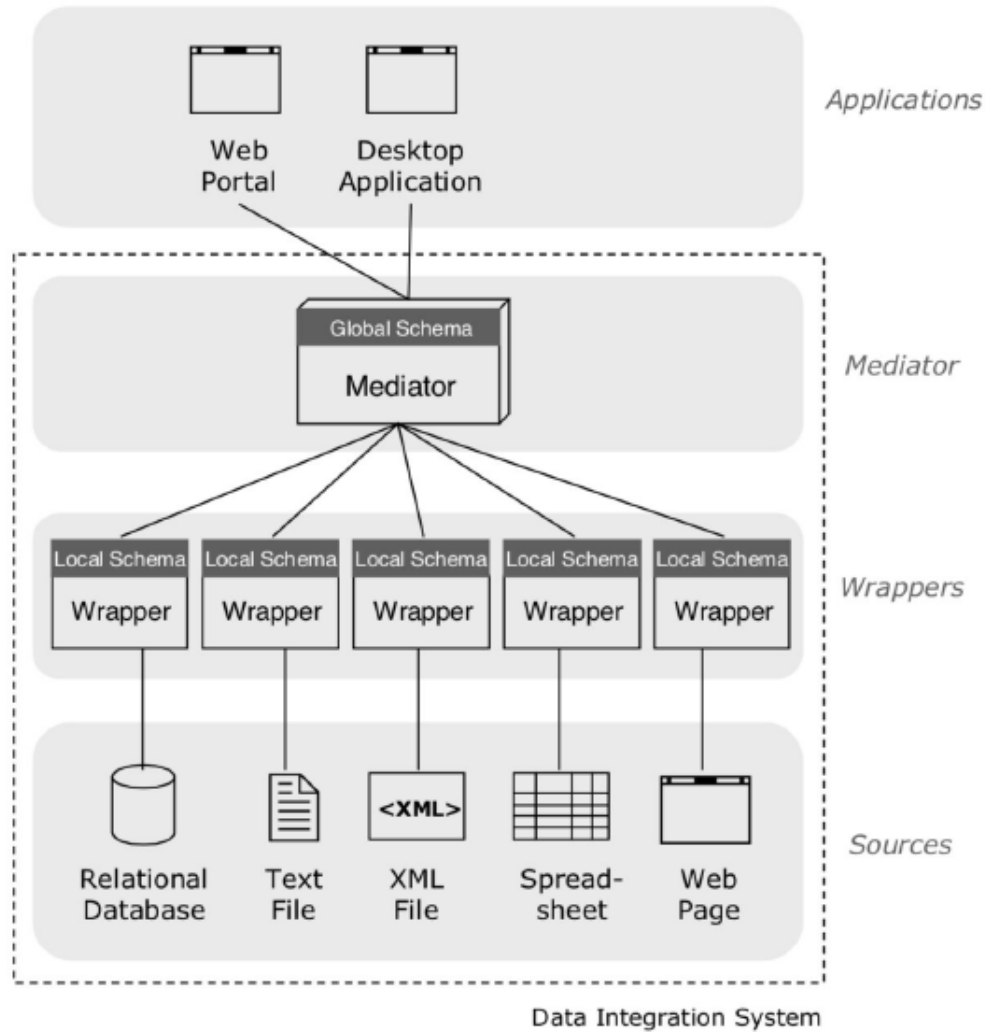


FIGURE 2.1 – Local as View / Global as view schema

Le schéma *Global-as-view* est donc un schéma “ascendant” où on part des sources pour produire le schéma global. Cela permet une traduction des requêtes plus simple, mais qui pousse à revoir tout le schéma si une nouvelle source de donnée fait son apparition.

Ce principe était parfaitement adapté à notre projet, car nous partions d’un cas d’étude, et donc, les sources étaient connues, nous étions donc moins exposés à un changement du schéma dans le futur.

Nous avons également travaillé sur certains chapitres de *Web Data Management* [SAb11]. Ce livre développe un peu plus les solutions actuelles pour le stockage des données et les utilisations des schémas GAV et LAV (Local-as-view).

On y observe des exemples concrets d'utilisation de ces 2 méthodes, mais aussi des exemples d'algorithme s'appliquant sur ces schémas. Ce qu'on peut en conclure, notamment dans le second article [SAb11] c'est que l'une des deux méthodes n'est pas foncièrement meilleure que l'autre, mais plutôt qu'elles sont plus adaptées à certaines situations. Par exemple, le désavantage principal de la méthode GAV c'est que si l'une des sources de données est amenée à changer ou qu'une nouvelle fait son apparition sous un format et/ou une syntaxe différente, l'intégralité du schéma peut-être amenée à changer. Ce désavantage, par exemple, ne s'applique que peu à notre situation puisqu'à part des corrections mineurs les données et les sources n'ont pas lieu d'être modifiées.

2.3 Solution proposée

A l'image des contraintes introduites par la problématique et les solutions imaginées à travers les articles que nous avons étudiés, nous avons agi sur plusieurs niveaux. D'abord, l'intégration des données et la gestion de leur hétérogénéité à travers 2 types de stockages. Le mapping des données suivant un schéma type "Global as view" définit avec un modèle conceptuel et la définition d'un schéma ontologique. L'extraction de nos données suivant 2 méthodes, une en réseau inter-langage et une extraction plus classique via des fichiers de sorties. Finalement, l'analyse des résultats via des techniques de la science des données. Nous en avons déduit un schéma de l'architecture du projet permettant de différencier les différents axes de travail et un ordre visuel.

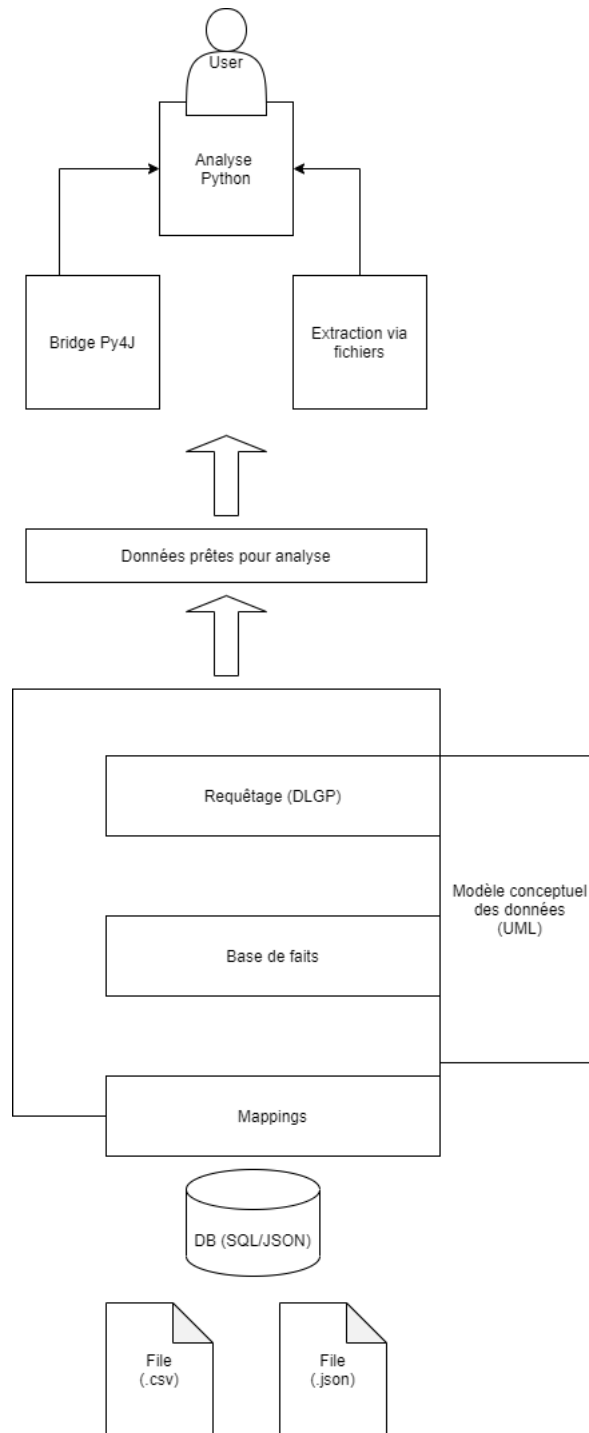


FIGURE 2.2 – Schéma de l'architecture du projet

2.4 Cas d'étude : prédiction de survie dans le Titanic



FIGURE 2.3 – Titanic

Pour ce projet, nous avons travaillé sur les données du Titanic, plus particulièrement sur les passagers et les informations associés.

Il s'agit d'un exemple classique de la littérature, qui a été choisis, car il permettait de montrer l'intérêt de l'apprentissage, mais aussi de l'intégration des données, point qui a été illustrée à travers une répartition des données sur des sources hétérogènes.

En résumé, il s'agit d'un exemple maîtrisé qui nous a permis de mettre en avant une vaste famille de cas d'utilisations dans lesquels on a des données hétérogènes JSON et SQL qu'on veut intégrer, et ensuite analyser nos résultats avec des méthodes d'apprentissage.

Nous avons donc travaillé sur 3 jeux de données correspondants aux 3 classes des passagers qui ont embarqué à bord du Titanic.

Les 2 premières classes étaient stockées dans des fichiers .csv et la troisième dans un fichier JSON, ainsi nous avons bien une hétérogénéité dans les sources de données pour créer un schéma global et travailler sur des mappings différents.

On pourrait se demander pourquoi partir de ce point de départ, plus précisément, pourquoi avoir mis ces données dans des sources diverses? C'est un exemple parlant, si chaque port d'embarquement avait sa

propre liste de passager on peut facilement imaginer que chacun aurait sa méthode d'écriture, les regrouper et les traiter est donc une mission importante et encore actuelle sur de nombreux cas (liste de passagers de l'aviation, liste de clients de restaurants différents...)

Ces données, bien que relatives au même événement, ont quand même quelques différences à prendre en compte pour la suite au delà du type de fichier. On observe, par exemple, que dans les 2 premières classes, le nom et le prénom sont un seul et même attribut alors que dans la 3ème classe le nom et le prénom sont deux éléments distincts. Une autre remarque importante c'est que dans les fichiers .csv lorsque un élément n'est pas renseigné il est remplacé par un 0 ou un "null" alors que dans le fichier JSON l'attribut n'apparaît tout simplement pas (par exemple, si la personne n'a pas survécu, la ligne sur "boat" correspondant au canot de sauvetage n'apparaît pas) Ce sont ces détails qui avaient leur importance dans notre démarche de global.

La définition des termes est la suivante :

- *pclass* : la classe dans laquelle la personne a embarqué.
- *survived* : un booléen répondant à la question : La personne a-t'elle survécu à l'accident ?
- *textitname* : Le nom et le prénom de la personne (parfois divisé en *lastname* et *firstname*).
- *sex* : le genre du passager.
- *age* : l'âge du passager.
- *sibsp* : l'addition du nombre de frères et soeurs de l'époux du passager qui sont également dans le navire
- *parch* : l'addition du nombre de parents et d'enfants du passager qui sont également dans le navire
- *ticket* : le numéro du ticket du passager (qui est parfois partagé entre plusieurs personnes)
- *fare* : le prix du voyage.
- *cabine* : les cabines assignées par le ticket. (possiblement plusieurs)
- *embarked* : la zone d'embarcation du passager.
- *boat* : le bateau d'évacuation utilisé après l'accident.
- *body* : Si le corps du passager a été retrouvé, un nombre lui a été assigné.
- *home.dest* : La provenance et/ou la destination du passager

```
1 pclass,survived,name,sex,age,sibsp,parch,ticket,fare,  
2 cabin,embarked,boat,body,home.dest  
3 1,1,"Allen, Miss. Elisabeth Walton",female,29,0,0,24160,211.3375,B5,S,2,?,  
4 "St Louis, MO"
```

Extrait des fichiers .csv

```
1 {  
2   "pclass": 3,  
3   "survived": 0,  
4   "lastname": "Abbing",  
5   "firstname": "Mr. Anthony",  
6   "sex": "male",  
7   "age": 42,  
8   "sibsp": 0,  
9   "parch": 0,  
10  "ticket": "C.A. 5547",  
11  "fare": 7.55,  
12  "embarked": "S"  
13 }
```

Extrait du fichiers .json

2.5 Technologies

Technologies utilisés

- **MongoDB** : C'est un SGBD orienté documents NoSQL ne nécessitant pas de schéma prédéfini des données. MongoDB permet de manipuler des objets structurés au format BSON (JSON binaire). En d'autres termes, des clés peuvent être ajoutées à tout moment « à la volée », sans reconfiguration de la base. ^a
- **SQLite** : C'est un SGBD SQL qui a comme particularité de ne pas reproduire le schéma habituel client-serveur mais d'être directement intégrée aux programmes. L'intégralité de la base de données (déclarations, tables, index et données) est stockée dans un fichier indépendant de la plateforme. ^b
- **Graal** : Est un logiciel développé par l'équipe Graphik de l'Université des Sciences de Montpellier. C'est une bibliothèque logiciel dédiés à la réponse de requête par ontologie et permettant la création de base de faits et de connaissance ^c

a. MongoDB <https://www.mongodb.com/fr>

b. SQLite <https://www.sqlite.org/index.html>

c. Graal : <https://graphik-team.github.io/graal/>

2.6 Base de données

2.6.1 SQLite

Pour la première et la deuxième classe de passagers (correspondant au fichier .csv) nous avons donc décidé d'utiliser le SGBD SQLite. D'une part car SQLite est compatible avec Graal et d'autre part parce que c'est un SGBD léger et facile d'utilisation ce qui correspondait parfaitement à notre objectif d'obtenir une base de données relationnel stable et utilisable rapidement en intégrant les données brutes.

Le passage des données .csv initiale dans un fichier SQLITE à été fait sous java en utilisant *SQLite JDBC Driver*, une librairie qui permet la création et la manipulation d'un fichier SQLITE et *Opencsv*, une librairie qui permet l'ouverture et la manipulation d'un fichier .csv.

```
1 String class1 = "CREATE TABLE TITANICFIRSTCLASS " +
2     "( " +
3     " PCLASS      INT, " +
4     " SURVIVED    BOOLEAN," +
5     " NAME        CHAR(100), " +
6     " SEX         CHAR(1),"+
7     " AGE         REAL,"+ //or Unknown
8     " SIBSP      INT,"+
9     " PARCH      INT,"+
10    " TICKET      VARCHAR(20),"+
11    " TFARE       FLOAT,"+
12    " CABIN       VARCHAR(20),"+//or Unknown
13    " EMBARKED    CHAR(1),"+
14    " BOAT        VARCHAR(20),"+//or Unknown
15    " BODY        VARCHAR(20),"+//or Unknown
16    " HOME_DEST   VARCHAR(40) );";
17 stmt.executeUpdate(class1);
```

Exemple de création d'une classe sous SQLite

```
1 new SQLQuery("SELECT substr(NAME,1, instr(NAME, ',') - 1) AS NOM,  
2 substr(NAME, instr(NAME, ',') + 2) AS PRENOM,  
3 AGE,  
4 SEX  
5 FROM TITANICFIRSTCLASS")
```

Exemple de requête SQL issu de nos mappings

2.6.2 MongoDB

Pour la troisième classe (correspondant au fichier JSON) nous avons utilisé le SGBD NoSQL MongoDB, le format JSON s'adaptant parfaitement à Mongo, c'était assez simple pour nous d'y intégrer nos données et ainsi répondre au problème d'hétérogénéité posé pour préparer les bases du projet. MongoDB est également compatible avec Java et permet de créer des requêtes sur les documents pour récupérer plus facilement les données.

```
1  "_id": {  
2    "$oid": "60213630500f164df8ed4055"  
3  },  
4  "pclass": 3,  
5  "survived": 0,  
6  "lastname": "Asplund",  
7  "firstname": "Master. Filip Oscar",  
8  "sex": "male",  
9  "age": 13,  
10 "sibsp": 4,  
11 "parch": 2,  
12 "ticket": 347077,  
13 "fare": 31.3875,  
14 "embarked": "S",  
15 "home": {  
16   "dest": "Sweden Worcester, MA"  
17 }
```

Exemple de document issu de notre base MongoDB

```
1 coll.find()  
2 .projection(new Document("lastname",1)  
3 .append("_id",0)  
4 .append("firstname", 1)  
5 .append("sex", 1));
```

Exemple de requête MongoDB issu de nos mappings

Chapitre 3

Conception et mapping

3.1 Conception

3.1.1 Conception : UML

Après avoir bien compris le mapping GAV et avoir préparé nos bases de données, nous devons donner un cadre à nos données dans le but de préparer le mapping.

Contrairement à nos "habitudes" (acquise à travers nos expériences de projet), nous devons créer un UML après avoir intégré les données aux bases de données, c'est un exemple supplémentaire des problématiques liées à l'hétérogénéité des données.

Cela était un petit peu déstabilisant pour nous, car nous étions habitués à commencer par faire un UML dans l'optique de créer une base de données correspondant à celui ci, ici, nous devons créer un UML dont l'objectif était de préparer des "boîtes" les plus simples possibles pour nos futurs mapping pour que les sources différentes ne soit plus un problème.

Malgré la difficulté, nous avons réussi à mettre en place un UML rassemblant les différentes informations présentes dans les classes en y ajoutant des relations et associations les plus claires et simples d'utilisation possible.

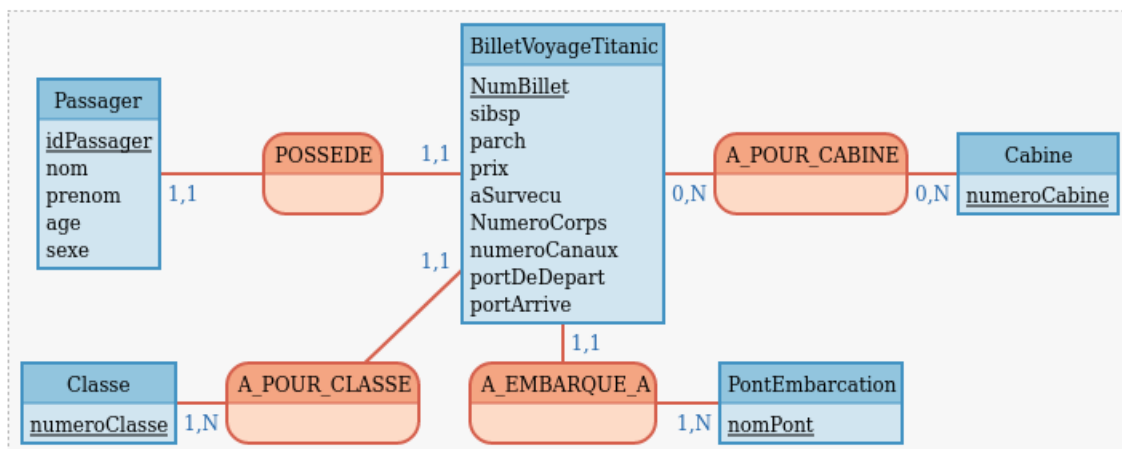


FIGURE 3.1 – Modèle conceptuel des données

3.1.2 Introduction au mapping via SQLite

Après avoir créé notre UML et avant de directement plonger dans Graal et de devoir mélanger l'apprentissage de Graal et compréhension de la notion de mapping avec Graal nous sommes passé par une étape de transition. En créant des mappings via SQLite, vue que nous étions déjà familié avec SQLite, nous pouvions nous concentrer sur cette notion.

Exemple de mapping :

```
1 String billet = "INSERT INTO TitanicMapped.BILLETVOYAGETITANIC"
2           + " (TICKET, SIBSP , PARCH , TFARE , SURVIVED , BODY ,
3           BOAT, HOME, DEST) "
4           + " SELECT TICKET, SIBSP , PARCH , TFARE , SURVIVED , "
5           + " BODY , BOAT , "
6           + " (SELECT substr(HOME_DEST,1,
7           instr(HOME_DEST, '/') - 1)) , "
8           + " (SELECT substr(HOME_DEST,
9           instr(HOME_DEST, '/' ) + 1)) "
10          + " FROM TITANICFIRSTCLASS ;";
```

Cette requête permet de récupérer les données brutes de la base de données SQLite créée précédemment et d'y appliquer les traitements nécessaire pour s'adapter à notre modèle UML à travers des fonctions SQLite comme instr() (qui permet de séparer le nom et le prénom par exemple, car comme dis précédemment le nom et le prénom n'était qu'un seul et même attribut au sein des 2 premières classes)

3.1.3 Introduction au mapping via Graal

Désormais, nous avons compris le principe de mapping pour nos données via notre UML et nos requêtes de mapping SQLite.

Pour prendre en main et découvrir les différentes possibilités nous avons commencé par sélectionner 3 passagers de la première classe en essayant de comprendre les différentes structures de données à disposition dans Graal. L'idée était que si nous pouvions faire rentrer dans notre boite certains passagers alors nous pouvions tous les faire rentrer, et cela, permettait de confirmer que notre modèle était adaptés aux données.

```
1 //Création du prédicat
2 Predicate passager = new Predicate("Passager", 4);
3
4 //Liste de Term
5 ArrayList<Term> p1Passager = new ArrayList<Term>();
6     p1Passager.add(DefaultTermFactory.instance()
7     .createLiteral("Allison"));
8     p1Passager.add(DefaultTermFactory.instance()
9     .createLiteral("Master. Hudson Trevor"));
10    p1Passager.add(DefaultTermFactory.instance()
11    .createVariable(0.9167));
```



```

12         p1Passager.add(DefaultTermFactory.instance()
13             .createLiteral("male"));
14
15         //Liste de liste de Term contenant toute les relations
16         ArrayList<ArrayList<Term>> dbPassager = new ArrayList<ArrayList<Term>>();
17         dbPassager.add(p1Passager);
18
19         // liste d'atome contenant tout les passager
20         ArrayList<Atom> passagerList = new ArrayList<Atom>();
21
22         // insertion des données
23         for (int i = 0; i < dbPassager.size(); i++) {
24
25             passagerList.add(DefaultAtomFactory.instance()
26                 .create(passager, dbPassager.get(i)));
27
28         }

```

3.2 Mapping complet via Graal

Maintenant, une relation devenait une liste d'atomes, un atome étant composé d'un prédicat (le nom de la relation) et des éléments de la relation appelé des termes (vocabulaire de la logique du premier ordre, utilisé par Graal).

Désormais plus familier avec Graal et les différents concepts introduit précédemment, nous pouvions mettre en place un mapping complet. Notamment une classe de mapping avec une fonction permettant, à partir d'une requête SQL et d'un prédicat d'obtenir en sortit une liste d'Atom "prête à l'emploi" avec l'arité donné par le prédicat.

3.2.1 Fonction de mapping générique SQLite

```

1     public static ArrayList<Atom> evaluate(SqliteDriver database,
2     SQLQuery query, Predicate p) throws SQLException {
3         ResultSet res = database.createStatement()
4             .executeQuery(query.toString());
5         ArrayList<Atom> tempAtomList = new ArrayList<Atom>();
6
7         try {
8
9
10            //Vérification de la cohérence de la requête et du prédicat
11
12            if(res.getMetaData().getColumnCount() != p.getArity()) {
13                System.out.println("Le nombre d'arguments
14                dans la requête est différent de la taille du prédicat");

```

```

15         return null;
16     }
17
18
19     //Création des Term pour chaque relation
20     while(res.next()) {
21         ArrayList<Term> temp = mainExemple.createTermList();
22
23         //Tant qu'il y a des données à mettre dans le prédicat
24         for(int i = 1; i <= p.getArity(); i++) {
25             String tempString = res.getString(i);
26             if(tempString != "null") {
27                 temp.add(DefaultTermFactory.instance()
28                     .createLiteral(tempString));
29             }
30             else {
31                 temp.add(DefaultTermFactory.instance()
32                     .createVariable(tempString));
33             }
34         }
35         tempAtomList.add(DefaultAtomFactory.instance().create(p, temp));
36     }
37
38
39
40     } catch(Exception e) {
41         e.printStackTrace();
42     }
43
44
45
46
47     return tempAtomList;
48
49 }

```

Cette fonction est assez simple dans son fonctionnement, elle s'occupe de vérifier la cohérence entre le prédicat et la requête (notamment le nombre d'arguments) et d'exécuter la requête pour parcourir les résultats de celle-ci en créant des termes correspondant au couple (Liste de termes, prédicat) de Graal . Elle permet également de récupérer et de traiter les données provenant de notre base de données rapidement et simplement en 1 ligne d'appel de ce type :

```

1  //Exemple d'appel de la fonction pour créer la classe "passager" provenant
2  //de notre UML et l'ajouter à notre liste d'Atom.
3
4
5  SQLQuery sqlQuery = new SQLQuery
6  ("SELECT substr(NAME,1, instr(NAME, ',') - 1) AS NOM
7  ,substr(NAME, instr(NAME, ',') + 2) AS PRENOM,AGE,SEX

```

```

8 FROM TITANICFIRSTCLASS");
9
10 passagerList.addAll(SQLMappingEvaluator.evaluate(testBase,sqlQuery,
11 new Predicate("passagerRelation",4)));

```

3.2.2 Fonction de mapping générique MongoDB

```

1 public static ArrayList<Atom> evaluate(MongoCollection<Document> coll,
2 FindIterable<Document> query, Predicate p) {
3
4     ArrayList<Atom> tempAtomList = new ArrayList<Atom>();
5
6     try {
7
8         for(Document x : query) {
9             ArrayList<Term> temp = mainExemple.createTermList();
10             // "Document{{lastname=Mahon, firstname=Mr. John, sex=male}}
11             //Tant qu'il y a des données à mettre dans le prédicat
12             String tempString = "";
13             for(int i = 1; i <= p.getArity(); i++) {
14                 String tempS = x.toString();
15                 String[] pls = tempS.split("[a-zA-Z_0-9]*=");
16
17                 if(i >= pls.length) {
18                     tempString = "null";
19                 }
20                 else {
21
22                     tempString = pls[i];
23                     tempString = tempString.replace("}}", "");
24                     tempString = tempString.replace(",", "");
25                     System.out.println(tempString);
26                 }
27                 if(tempString != "null") {
28                     temp.add(DefaultTermFactory.instance()
29                         .createLiteral(tempString));
30                 }
31                 else {
32                     temp.add(DefaultTermFactory.instance()
33                         .createVariable(tempString));
34                 }
35             }
36             tempAtomList.add(DefaultAtomFactory.instance()
37                 .create(p, temp));
38
39         }
40     } catch(Exception e) {
41         e.printStackTrace();

```

```

42         }
43
44         return tempAtomList;
45     }

```

Cette fonction est un peu différente de la précédente, d'une part parce que les requêtes MongoDB sont différentes des requêtes SQL et d'autre part, car le format en sortie des requêtes est un peu particulier ce qui nous pousse à effectuer quelques transformations pour correspondre à notre schéma GAV.

```

1  //Exemple d'appel de la fonction pour créer la classe "passager" provenant
2  //de notre UML et l'ajouter à notre liste d'Atom.
3  passagerList.addAll(MongoDBMappingEvaluator.evaluate(coll,coll.
4  find().projection(new Document("lastname",1)
5  .append("_id",0)
6  .append("firstname", 1)
7  .append("sex", 1)
8  .append("age", 1)),
9  new Predicate("passagerRelation",4)));

```

Cela nous permet de créer des mappings rapidement et de traiter une grande quantité de données provenant de différentes bases de données pour les faire correspondre à notre modèle UML (on pourrait facilement imaginer un système de multi-threading pour traiter plusieurs bases en simultané et de manière très rapide). Un autre élément important que nous avons obtenu à travers ces classes génériques c'est que désormais ces mappings ne sont plus dépendants des données, si des lignes sont ajoutées aux sources nos mappings fonctionnent quand même correctement.

Chapitre 4

Résultat et extraction

4.1 Pourquoi

A ce stade du projet nous avons maintenant des données dans un format **unique** que nous avons nous-même défini et avec lequel nous étions familiers, cela fait abstraction des sources et nous pouvions désormais travailler comme nous en avions l'habitude.

Après avoir utilisé Java tout au long de notre projet, nous avons décidé pour la suite et notamment pour l'analyse des données d'utiliser un langage différent, plus adapté à la science des données : **Python**.

Pour plusieurs raisons :

- Data'Clear avait pour objectif d'être compréhensible pour le plus grand nombre et une présentation via Jupyter et un langage plus spécialisé dans la science de la donnée était plus adapté.

- La modification des données directement dans Graal nous poussait à créer des règles supplémentaires ce qui complexifiait le traitement des données et ce n'était pas l'objectif.

Pour cela, maintenant, que les données respectaient le modèle que nous avons créé, il fallait pouvoir les extraire efficacement sans altérer notre modèle. Nous avons utilisé le requêtage DLGP¹ pour obtenir des lignes de résultat correspondant à chaque passager respectant le format que nous avons mis en place au sein de Graal.

```
1 ConjunctiveQuery query = DlgpParser.parseQuery("(?(A,B,C,D,M,E,F,G,H,K) :- "  
2     + " passagerRelation(A,B,C,D) ,"  
3     + " cabineRelation(A,B,M) ,"  
4     + " aPourClasse(A,B,E) ,"  
5     + " voyageTitanic(A,B,F,G,H,I,J,K,L) ." );
```

1. DLGP : <https://graphik-team.github.io/graal/doc/dlgp>

4.2 Méthode 1 : Py4J

Comme nous travaillons avec Graal en Java sous Eclipse et que nous voulions utiliser Python pour la suite il était nécessaire de créer un premier “pont” entre notre travail avec Graal et Python pour garder une linéarité dans le parcours de nos données.

Pour cela, nous avons utilisé Py4J². Py4J est un outil permettant d’accéder dynamiquement à des objets Java à travers une JVM, ainsi, on a pu directement effectuer nos requêtes DLGP à travers un objet Java et directement appeler les requêtes DLGP côté Python sur notre base de connaissances Graal. Néanmoins cette méthode imposait de nouvelles transformations au sein de Python, pour éviter ce problème, nous avons réfléchi à une méthode différente pour l’extraction de nos résultats.

```
1  from py4j.java_gateway import JavaGateway
2  #Début Méthode 1 : Requête directe de la KB Graal
3  gateway = JavaGateway()
4
5  graal = gateway.entry_point
6  print(graal)
7
8
9  dataQuery = graal.evaluate("? (A,B,C,D,M,E,F,G,H,K) :- " +
10 " passagerRelation(A,B,C,D)," +
11 " cabineRelation(A,B,M)," +
12 " aPourClasse(A,B,E)," +
13 " voyageTitanic(A,B,F,G,H,I,J,K,L).")
```

Extrait d’une ligne de sortie DLGP.

```
1  {M->"null",
2  H->"1.0",
3  K->"null",
4  E->"2",
5  D->"F",
6  G->"1.0",
7  F->"250651",
8  A->"Lahtinen",
9  C->"26.0",
10 B->"Mrs. William (Anna Sylfven)"} 
```

2. Py4j : <https://www.py4j.org/>

4.3 Méthode 2 : Les fichiers

Une autre méthode qui nous a semblé palier à plusieurs problèmes. Py4J imposait, par exemple, de relancer les requêtes pour créer la connexion entre Python et notre code Java, cela permettait d'avoir toujours des données à jours, mais c'était une contrainte à prendre en compte. Dans le cadre de notre cas d'étude les données avaient peu de chance d'être souvent modifié donc nous avons décider de récupérer le résultat de nos requêtes DLGP et de les écrire dans des fichiers .csv qui sont particulièrement adapté pour les analyses via Python.

```
1 //Méthode d'écriture d'une requête
2 public static void writeQuery(CloseableIterator<?> result
3     ,String txt) throws IOException {
4     try {
5
6         File file = new File("C:\\Users\\beaug\\Desktop\\M1S2\\TER\\
7         Python\\" + txt + ".csv");
8
9         if(!file.exists()) {
10             file.createNewFile();
11         }
12
13         FileWriter fw = new FileWriter(file.getAbsolutePath());
14         BufferedWriter bw = new BufferedWriter(fw);
15
16         while(result.hasNext()) {
17             bw.write(result.next().toString() + "\n");
18         }
19         bw.close();
20
21         } catch (IOException e) {
22             e.printStackTrace();
23         }
24     }
25
26 }
27
```

Nous obtenons alors 2 fichiers :

| survived | lastname | firstname |
|----------|----------|-------------------------|
| 0 | Lindblom | Miss. Augusta Charlotta |
| 0 | McCrie | Mr. James Matthew |
| 1 | Baclini | Miss. Eugenie |
| 0 | Rice | Master. Arthur |
| 1 | Davies | Master. John Morgan Jr |

FIGURE 4.1 – Extrait du premier fichier témoin contenant les passagers et leur attribut de survie

| embarked | sibsp | boat | pclass | sex | parch | ticket | lastname | age | firstname |
|----------|-------|------|--------|--------|-------|----------|----------|------|--------------------------------------|
| NaN | 1.0 | NaN | 2 | female | 1.0 | 250651 | Lahtinen | 26.0 | Mrs. William (Anna Sylfven) |
| NaN | 1.0 | NaN | 2 | male | 1.0 | 250651 | Lahtinen | 30.0 | Rev. William |
| E8 | 0 | 5 | 1 | male | 1 | 113806 | Chambers | 27.0 | Mr. Norman Campbell |
| E8 | 0 | 5 | 1 | female | 1 | 113806 | Chambers | 33.0 | Mrs. Norman Campbell (Bertha Griggs) |
| NaN | 0.0 | NaN | 2 | male | 0.0 | SC 14888 | Parker | 28.0 | Mr. Clifford Richard |

FIGURE 4.2 – Extrait du second fichier contenant toutes les données à analyser

Ici, on simule alors la possibilité que les fichiers ne sont pas issue du même jeux de données.

4.4 2 méthodes ?

On pourrait se demander pourquoi, à ce stade, avoir proposé deux méthodes différentes pour l'extraction de nos données avant leur analyse. On pourrait imaginer que l'une est meilleure que l'autre et faire abstraction de la seconde, mais encore une fois cela dépend de l'utilisation et du cas de figure.

Si les données subissent un grand nombre de mises à jour régulières mais minimise le système en réseau via Py4J est plus adapté si on considère extraire uniquement les lignes modifiés, là où la réécriture des fichiers prend du temps avec un grand nombre de données.

D'autre part, la situation inverse, très peu de modification et des données inchangés sur le long terme. On préférera alors les fichiers, là où maintenant une architecture réseau ne sera que peu pertinente en terme de ressources et d'utilité.

Chapitre 5

Analyse avec Python

5.1 Analyse

On obtient, après une fusion des deux documents (deux csv avec la méthode 2 et deux dataframes avec la méthode 1) par le nom et le prénom ainsi qu'une suppression des personnes dont on n'a pas l'information survived, le fichier suivant :

| embarked | sibsp | boat | pclass | sex | parch | ticket | lastname | age | firstname | survived |
|----------|-------|------|--------|--------|-------|----------|----------|------|--------------------------------------|----------|
| NaN | 1.0 | NaN | 2 | female | 1.0 | 250651 | Lahtinen | 26.0 | Mrs. William (Anna Sylfven) | 0 |
| NaN | 1.0 | NaN | 2 | male | 1.0 | 250651 | Lahtinen | 30.0 | Rev. William | 0 |
| E8 | 0 | 5 | 1 | male | 1 | 113806 | Chambers | 27.0 | Mr. Norman Campbell | 1 |
| E8 | 0 | 5 | 1 | female | 1 | 113806 | Chambers | 33.0 | Mrs. Norman Campbell (Bertha Griggs) | 1 |
| NaN | 0.0 | NaN | 2 | male | 0.0 | SC 14888 | Parker | 28.0 | Mr. Clifford Richard | 0 |

FIGURE 5.1 – Extrait des données initial après fusion

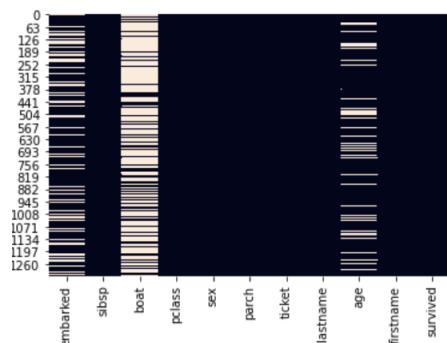


FIGURE 5.2 – Répartitions des données null (en clair) dans le fichier, les nombres représentent l'index des passagers dans le fichier

| | embarked | boat | sex | ticket | lastname | firstname |
|---------------|----------|------|------|----------|-----------|-----------|
| count | 999 | 370 | 1320 | 1320 | 1320 | 1320 |
| unique | 179 | 73 | 2 | 929 | 875 | 1141 |
| top | S | 4 | male | CA. 2343 | Andersson | Mr. James |
| freq | 498 | 32 | 849 | 11 | 11 | 15 |

FIGURE 5.3 – nombre de données dans chaque colonne, le nombre de différents, l'élément le plus présent et sa fréquence

Une analyse des données dans sa globalité nous a permis de réduire le nombre de catégories intéressantes. Par exemple, les informations telles que le port d'embarquement ou le ticket étaient trop personnelle et à priori ne semblaient pas pertinent dans l'analyse de la survie du passager. Le but étant de généraliser les données, nous avons modifié ou ajouté certaines valeurs :

- sibsp et parch on été additionné pour former fsize : *family size*, le nombre de personnes de la même famille que le passager qui sont également dans le bateau.
- Le nom de chaque personne a été réduit à son titre (title) (Mrs, Miss, Lady...). Ces titres on ensuite été modifié pour former des groupes plus large. Par exemple :
 - Lady , Countess, Don sont devenu Royalty
 - Mme, Ms, Mrs, Miss, Mlle sont devenu Miss
- Cette catégorisations à permis de donner un âge aux personnes n'en ayant pas. Celui-ci a été déterminé selon l'âge moyen des personnes ayant le même titre.

| age | survived | fsize | title | 1 | 2 | 3 | female | male |
|------|----------|-------|---------|---|---|---|--------|------|
| 26.0 | 0 | 2.0 | Mr | 0 | 1 | 0 | 1 | 0 |
| 30.0 | 0 | 2.0 | Officer | 0 | 1 | 0 | 0 | 1 |
| 27.0 | 1 | 1.0 | Mr | 1 | 0 | 0 | 0 | 1 |
| 33.0 | 1 | 1.0 | Mr | 1 | 0 | 0 | 1 | 0 |
| 28.0 | 0 | 0.0 | Mr | 0 | 1 | 0 | 0 | 1 |

FIGURE 5.4 – Extrait des données après modifications

C'est ensuite suivi une normalisation des données avec Standard Scaler.¹

| age | survived | fsize | title | 1 | 2 | 3 | female | male |
|-----------|----------|-------|-------|---|---|---|--------|------|
| -0.302743 | 0 | 2.0 | 2 | 0 | 1 | 0 | 1 | 0 |
| 0.001782 | 0 | 2.0 | 4 | 0 | 1 | 0 | 0 | 1 |
| -0.226612 | 1 | 1.0 | 2 | 1 | 0 | 0 | 0 | 1 |
| 0.230177 | 1 | 1.0 | 2 | 1 | 0 | 0 | 1 | 0 |
| -0.150480 | 0 | 0.0 | 2 | 0 | 1 | 0 | 0 | 1 |

FIGURE 5.5 – Extrait des données après normalisation

5.2 Résultat

Les données étant prête pour la classification nous avons séparé une nouvelle fois les informations du passager et la colonne “survived” afin de pouvoir créer des jeux d'apprentissages via *RepeatedKfold* [htt20], une méthode de cross validation répétée. La cross validation permet notamment d'éviter le sur-apprentissage des données et donc de rendre le classifieur plus efficace face à de nouvelles données.

1. Standard Scaler

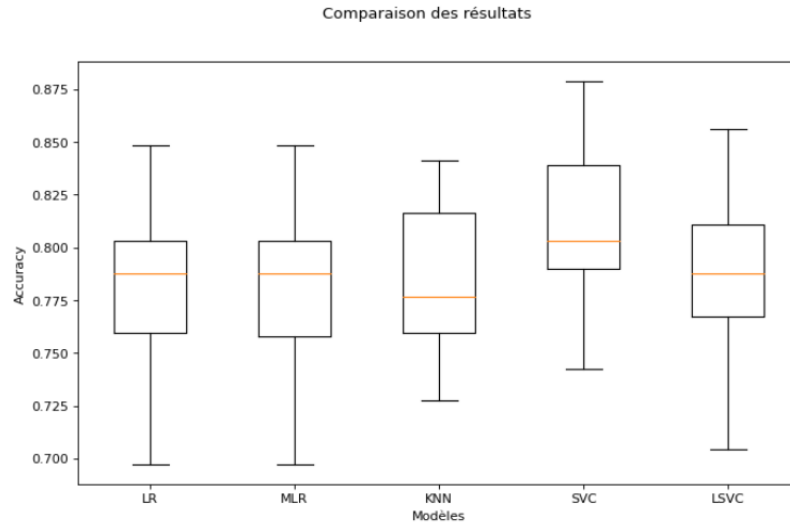


FIGURE 5.6 – Comparaison des accuracy entre différent classifieurs

Nous avons atteint 81% d'accuracy avec le classifieur linéaire *Support vector machine*(SVM)². Un résultat qui concorde avec les études existantes sur ce jeu de données. C'est une conclusion satisfaisante et rassurante puisque nos mappings et nos transformations n'ont donc pas altéré la qualité des données initial, élément très important dans le cadre de la science des données.

2. Support vector machine

Chapitre 6

Bilan

6.1 Et après ?

Après avoir travaillé plusieurs mois sur notre cas d'étude nous arrivons à un résultat satisfaisant d'un point de vue technique. Nous avons travaillé avec de nouveaux outils que nous maîtrisons aujourd'hui et nous avons pu aller au bout du projet en proposant une analyse et des résultats basé sur notre propre travail. Néanmoins à travers ces différentes étapes de travail et des résultats obtenu, on peut prendre du recul et s'éloigner du cas d'étude. Finalement, les différentes solutions proposés à travers *Data'Clear* ne sont pas spécifiques à notre cas d'étude mais peuvent être adapté à d'autres exemples ou domaines. Répondre à l'hétérogénéité des sources et proposer une "boîte" adapté à la situation est une solution global. Une analyse détaillé des données n'est que la suite logique de la bonne "préparation" de celle-ci et les résultats obtenue peuvent être présenté et compris par le plus grand nombre. Ce modèle peut-être proposé aussi bien à des particuliers qu'à une société.

Ce projet peut être donc repris, dans ces grandes lignes et dans les solutions proposées pour être adapté à d'autres situations ou à d'autres cas d'étude en utilisant les mêmes réflexions et technologies que nous avons utilisés ici.

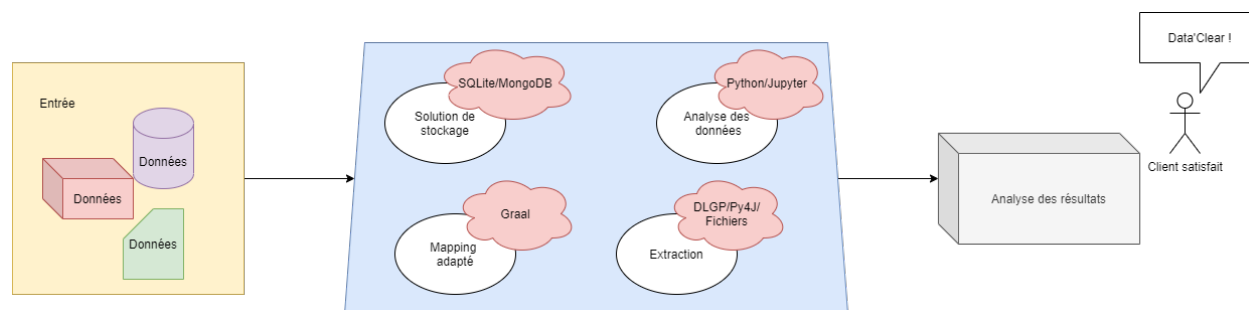


FIGURE 6.1 – Data'Clear : une solution complète

6.2 Conclusion

Data'Clear est le premier sujet de TER que nous proposons et c'est également le premier projet conséquent que nous faisons à deux, tout cela dans une période particulière. Néanmoins, l'accompagnement proposé par nos encadrants nous a permis de mieux appréhender le sujet et d'avoir du recul sur les objectifs et de pouvoir travailler efficacement en équipe.

Si nous devons en tirer une leçon, c'est que c'est important dans un projet de travaillé ensemble, de comprendre les spécificités du sujet avant de se lancer tête baissé sur nos claviers et c'est un facteur qui nous a énormément aidés tout au long du projet. Prendre du recul sur l'avancée du projet, au-delà du nombre de lignes de code, nous avons appris à réfléchir à une solution plutôt que d'appliquer directement une solution proposée. Réfléchir et prendre le temps nécessaire à chaque fonctionnalité nous a paradoxalement fait gagner de précieuses heures que nous avons pu réinvestir dans le projet et ainsi améliorer notre travail.

Nous sommes satisfaits de ce que nous avons produit tout au long du semestre et heureux d'avoir pu travailler sur un sujet qui nous intéressait avec des professionnelles du domaine dans le cadre de notre formation. Nous avons pu répondre à un grand nombre des questions que nous nous posions sur la science des données en apprenant également à mener un projet.

Tu ne connaîtras jamais la fin de l'histoire en faisant demi-tour à deux minutes de la victoire.

Bibliographie

- [htt16] Patrick Triest : [HTTPS://HUMANSOFDATA.ATLAN.COM/2016/07/MACHINE-LEARNING-PYTHON/](https://humansofdata.atlan.com/2016/07/machine-learning-python/). *Article d'analyse des données du Titanic*. 2016.
- [htt20] Jason Brownlee [HTTPS://MACHINELEARNINGMASTERY.COM/REPEATED-K-FOLD-CROSS-VALIDATION-WITH-PYTHON/](https://machinelearningmastery.com/repeated-k-fold-cross-validation-with-python/). *repeatedkfold cross-validation with python*. 2020.
- [htt21] Maurizio Lanzerini [HTTPS://CORE.AC.UK/DOWNLOAD/PDF/188826491.PDF](https://core.ac.uk/download/pdf/188826491.pdf). *Managing data through the lens of an ontology*. 2021.
- [SAb11] M-C.Rousset et P.Senellart <http://webdam.inria.fr/Jorge/files/wdm-data-integration.pdf> S.ABITEBOUL I.MANOLESCU P.RIGAUX. *Web Data Management*. 2011.

Annexes

Intégrer des données provenant de différentes sources en présence d'une ontologie (lire : un modèle conceptuel des données en UML) afin de préparer les données à l'apprentissage et la prédiction.

1 Les étapes

1.1 Chargement des données

La première étape consiste à charger les données des fichiers dans des bases de données adaptées. Nous optons pour deux systèmes, si possible différents, de base de données SQL pour les fichiers CSV ainsi qu'une base de données NoSQL pour le fichier JSON.

1.2 Définir le modèle ontologique

La seconde étape consiste à se mettre à la place des experts du domaine afin de définir un modèle ontologique (modèle conceptuel UML) permettant de décrire les données de façon unique.

1.3 Intégrer les données

La troisième étape consiste à intégrer les données des différentes sources dans le modèle ontologique (modèle conceptuel UML) en mettant en place un système de mappings de la forme

Requête sur la source $-i$. Données compatibles avec le format ontologique.

1.4 Graal

La quatrième étape consiste à utiliser Graal afin d'appliquer les mappings et les règles afin d'obtenir des données uniformes et complètes.

1.5 Apprentissage Prédiction / Visualisation

La cinquième étape consiste à déterminer qui va survivre au naufrage.

FIGURE 2 – Cahier des charges initial






















































| Travail |  | Nom | Durée | Début |
|--------------|---|---|--------------|----------------|
| 395,2 heures |   | <input type="checkbox"/> Data'Clear | 34 jours? | 29/01/21 08:00 |
| 3,2 heures |   | Planning Prévisionnel | 2 jours | 17/02/21 08:00 |
| 16 heures |   | <input type="checkbox"/> Etudes bibliographique | 1 jour | 29/01/21 08:00 |
| 16 heures |   | Etudes articles (EP) | 1 jour | 29/01/21 08:00 |
| 64 heures |   | <input type="checkbox"/> Chargement des données | 4,125 jours | 03/02/21 16:00 |
| 24 heures |   | BDD NoSQL | 3 jours | 03/02/21 16:00 |
| 24 heures |   | BDD SQL | 3 jours | 05/02/21 08:00 |
| 16 heures |   | Test + Rapport MAJ | 1 jour | 17/02/21 08:00 |
| 60 heures |   | <input type="checkbox"/> Définir le modèle ontologique | 3,75 jours? | 19/02/21 08:00 |
| 16 heures |   | Etudes détaillées | 1 jour? | 19/02/21 08:00 |
| 30 heures |   | Définir le modèle | 1,875 jours | 24/02/21 08:00 |
| 14 heures |   | Ecrire les règles | 0,875 jours | 26/02/21 16:00 |
| 68 heures |   | <input type="checkbox"/> Intégrer les données | 4,906 jours? | 03/03/21 15:00 |
| 16 heures |   | Implémentation règles | 1 jour? | 03/03/21 15:00 |
| 42 heures |   | Mappings via le modèle | 3,281 jours? | 05/03/21 15:00 |
| 10 heures |   | Test + Rapport MAJ | 0,625 jours? | 19/03/21 08:15 |
| 72 heures |   | <input type="checkbox"/> Graal | 4,5 jours? | 24/03/21 14:15 |
| 16 heures |   | Etudes préalables | 2 jours? | 24/03/21 14:15 |
| 20 heures |   | Intégrer le modèle à Graal | 1,25 jours? | 26/03/21 14:15 |
| 20 heures |   | Appliquer le mappings sur | 1,25 jours? | 31/03/21 16:15 |
| 16 heures |   | Test + Rapport MAJ | 1 jour? | 07/04/21 09:15 |
| 112 heures |   | <input type="checkbox"/> Apprentissage et prédiction | 13 jours? | 14/04/21 08:00 |
| 20 heures |   | Mise en place d'apprentissage | 1,25 jours? | 14/04/21 08:00 |
| 20 heures |   | Résultat des prédictions | 1,25 jours? | 16/04/21 10:00 |
| 24 heures |   | Visualisation | 1,5 jours | 21/04/21 12:00 |
| 48 heures |   | Rapport + Bilan | 7 jours? | 05/05/21 08:00 |

FIGURE 3 – Planning des tâches final

Profil de risque

| Profil de risque | | | | | | |
|----------------------|---|---|---|---|---|---|
| Risque | 0 | 1 | 2 | 3 | 4 | 5 |
| Taille du projet | O | O | O | X | O | O |
| Difficulté technique | O | O | X | O | O | O |
| Degré d'intégration | O | O | X | O | O | O |
| Organisation | O | X | O | O | O | O |
| Changement | O | O | X | O | O | O |
| Equipe | X | O | O | O | O | O |

FIGURE 4 – Profil de risque du projet

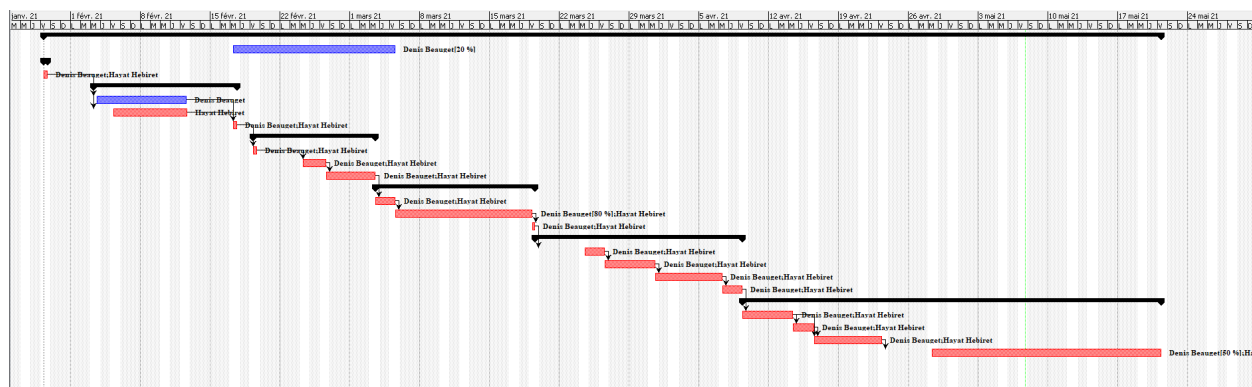


FIGURE 5 – Planning prévisionnelle

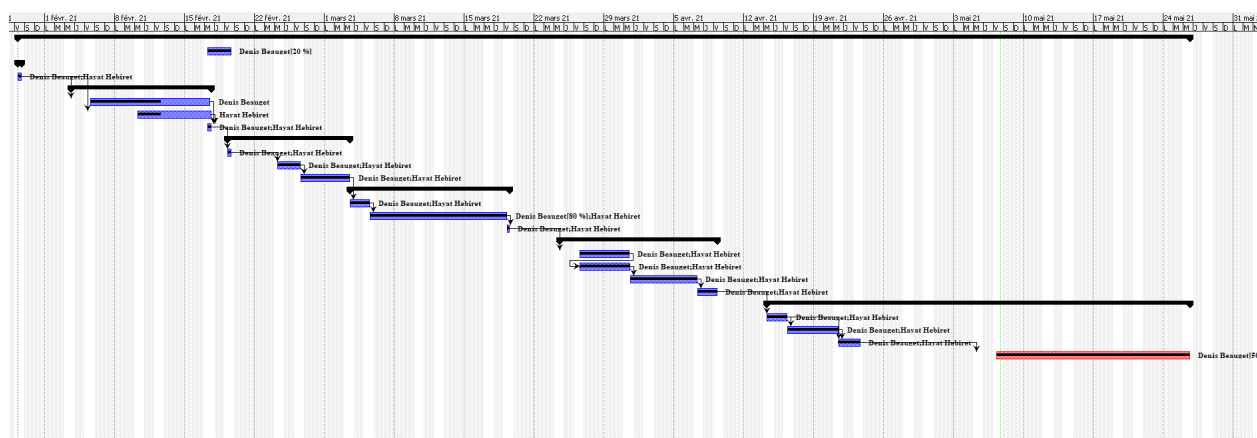


FIGURE 6 – Planning final mis à jour

<https://youtu.be/piaip86IgtQ>

FIGURE 7 – Vidéo de présentation du projet