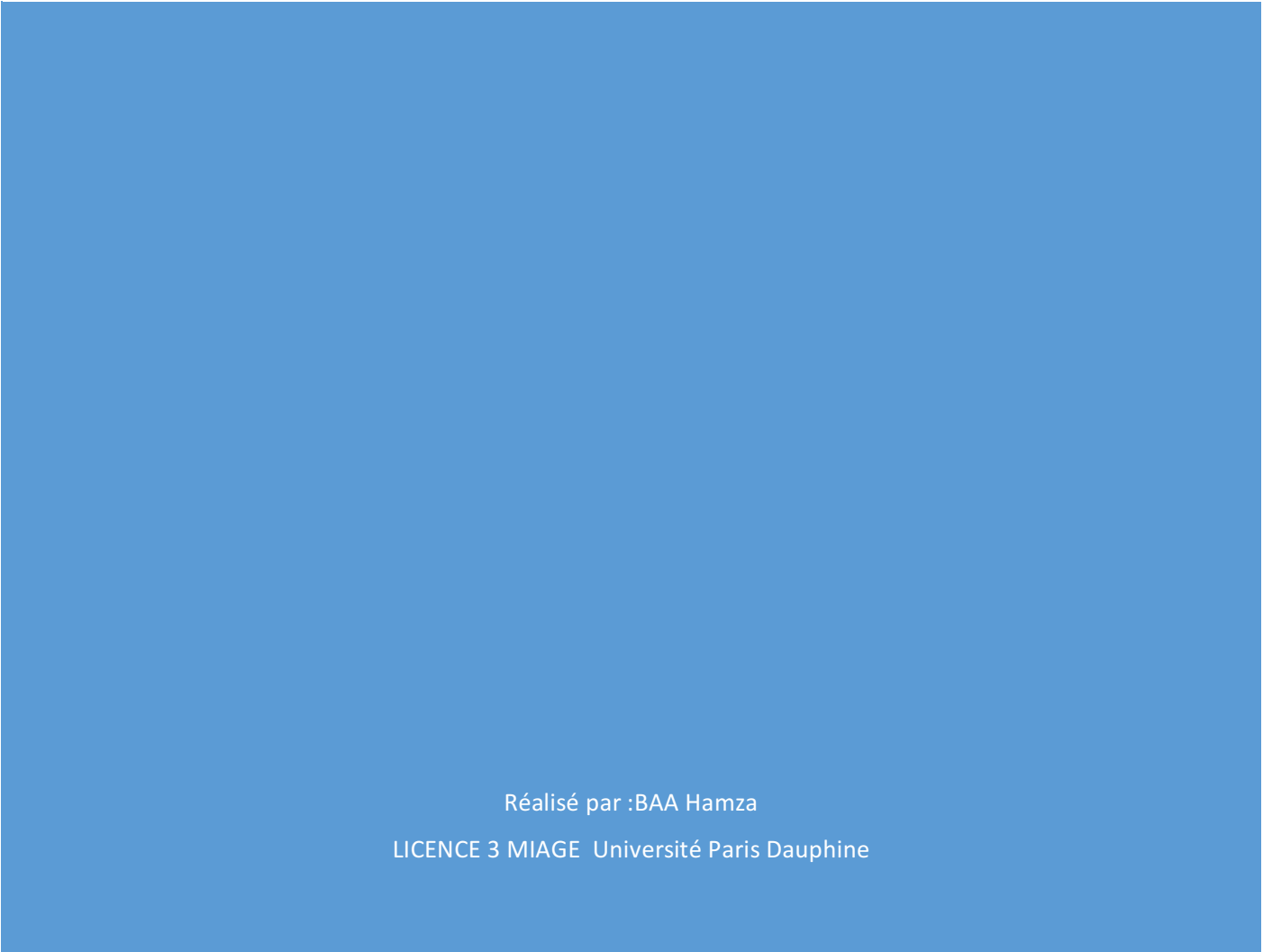




# COMPTE RENDU PROJET JAVA



Réalisé par :BAA Hamza  
LICENCE 3 MIAGE Université Paris Dauphine

🚦 **Intitulé :** correction des conflits apparus sur une base de données.

Ressources : deux bases de données sous forme de fichier CSV contenant les n-uplets représentant mon ensemble de chirurgies.

La base de données est constituée d'une seule relation.

Schéma de la base de données

```
R=(  
  CHIRURGIES (  
    Id_chirurgie,  
    Date_chirurgie,  
    Heure_debut_chirurgie,  
    Heure_fin_chirurgie,  
    Chirurgien,  
    Salle  
  )  
)
```

### **Conception et modélisation du problème :**

La conception de ce problème est divisée principalement sur 3 axes qui sont :

1-phase de récupération des données de la source (fichier csv).

2-modélisation et structuration des données.

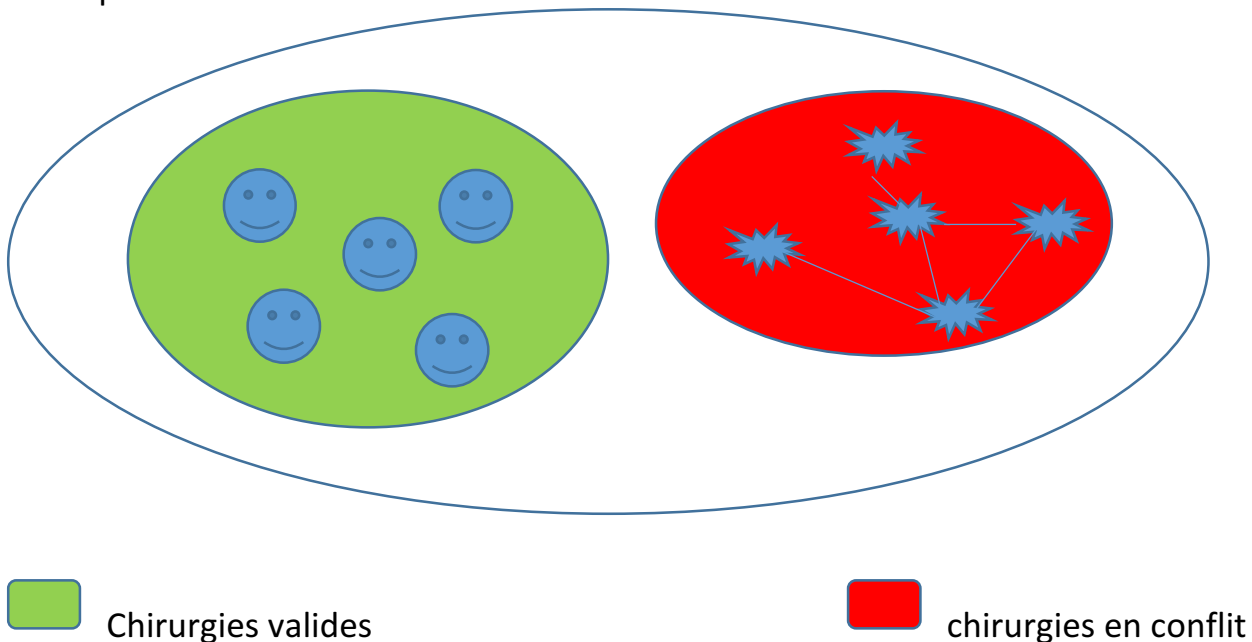
3-résolution du problème.

1-La phase de récupération des données de la source : pour cela j'ai essayé de créer un modèle standard qui permet la récupération des données d'un fichier CSV et pour cela j'ai fait appel au pattern DAO implémenté dans la classe du package com.model.utilitaire, donc l'idée était de passer du contenu de la base de données à une structure objet en mémoire, c.à.d. associer à chaque n-uplet de la base à une instance d'objet Chirurgie ceci grâce à la méthode mapToChirurgie(Map m) de la classe CsvChirurgieDAO (à savoir que les données mappées sont accessibles via la classe CsvFile, puis récupérer l'ensemble dans une liste via la méthode findAllChirurgies() de l'interface CsvDAO).

### 2-modélisation et structuration des données :

Après avoir récupéré la totalité de mes chirurgies j'ai séparé les chirurgies qui ne sont pas en conflit avec d'autres et celles qui le sont. L'implémentation est donnée dans la classe DataModel qui permettra l'accès à la liste de toutes les chirurgies, la liste des chirurgies qui ne sont pas en conflit avec aucune autre chirurgie que j'ai nommé les *chirurgies valides*.

Ainsi les autres chirurgies qui sont en conflit avec aux moins une chirurgie, cette deuxième partie est complémentaire à la première aussi les deux forme une partition de l'ensemble de toutes les chirurgie la figure qui suit montre cet aspect



Par ailleurs pour représenter les chirurgies qui sont en conflit j'ai utilisé une `Map<Chirurgie, List<Chirurgie>>` qui associer chaque chirurgie qui est en conflit avec une autre chirurgie ou plus avec la liste des chirurgies qui le sont réciproquement. la figure qui suit illustre la structure

Une fois fini j'ai modélisé l'entité de mon problème tel que j'ai associé une classe pour chaque type de conflit sachant que nous avons distingué 3 types d'où 3 classes une associée à chaque type `ConflitUbiquité` pour les conflits de type ubiquité et ainsi de suite, et les trois partagent la même classe mère, ce qui me permet d'appliquer le design pattern Factory pour les traitements et les comportements de ces conflits selon leur type.

3-Model de résolution : qui est lui répartie en deux sous model

3.a model de décision

Pourquoi ce model ?

Prenons un conflit entre deux Chirurgies C1 et C2 de type Ubiquité (les chirurgies partagent le même bloc opératoire) à priori pour ce type on essaie de chercher une solution permettant la modification de la salle de l'une des chirurgies et on part sur l'hypothèse que les temps début et fin sont corrects : donc pour la résolution de ce problème on distingue 3 cas

**1<sup>er</sup> cas** : il y'a des salles disponibles pour l'intervalle  $[d_i, f_i]$  de la chirurgie  $i$  mais pas sur la période  $[d_j, f_j]$  de l'autre.

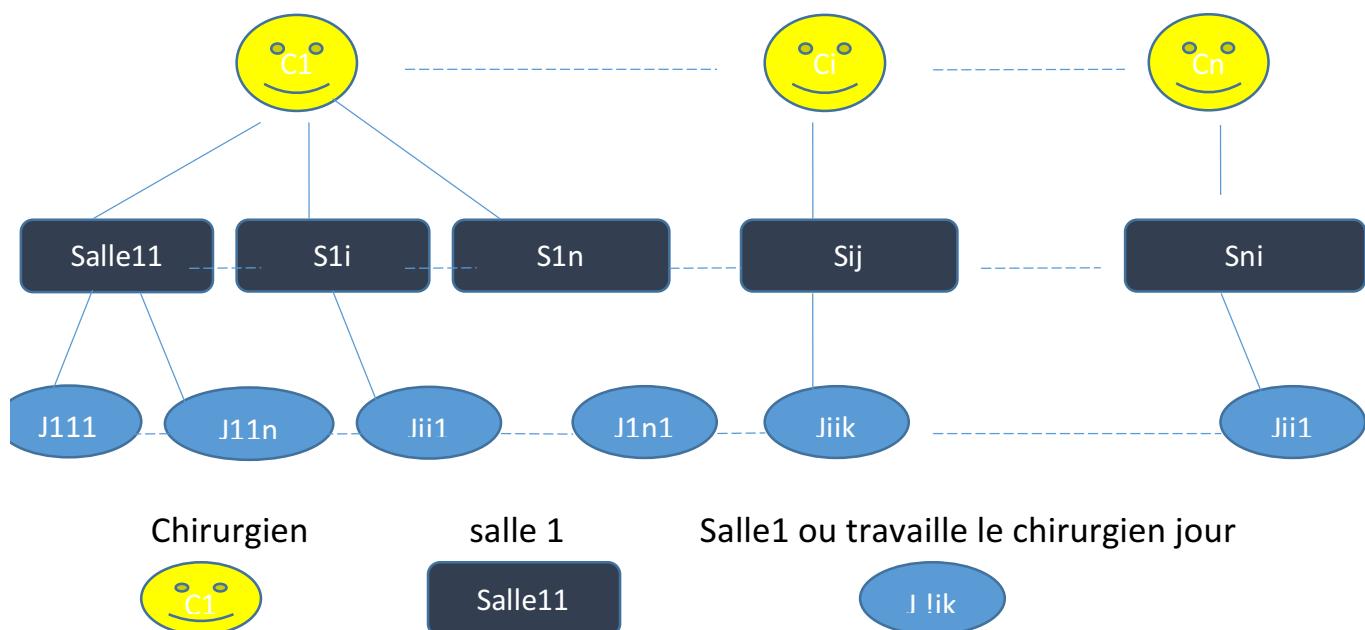
**2<sup>ème</sup> cas** : il y a des salles qui sont disponible pour les deux périodes. Dans ce cas la question qui se pose est : *laquelle des chirurgies je modifie et laquelle je garde ?*

**3<sup>ème</sup> cas** : il n'y a ni des salles disponibles pour la période de la chirurgie  $C_1$  ni pour celle de  $C_2$ , dans ce cas un autre débat s'ouvre et là nous avons qu'à ignorer l'hypothèse que les temps début et fin des chirurgies sont valides et corrects. Par contre la question dont nous avons poser précédemment revient quelle chirurgie dois-je modifier ??

Pour le cas 2 et cas 3 si une seule salle est disponible, tant mieux pour nous on le substitue directement pour la chirurgie erronée, par contre si nous avons plus d'une salle à proposer, une autre question très intéressante s'impose : *laquelle des salles proposées est la plus convenable pour résoudre le conflit ??*

Donc les questions qui se sont posée précédemment nous mettent les deux devant un choix par contre nous aimerons bien que ce choix soit bien fait ! et c'est exactement pour cela que la classe ModelDecision existe.

Présentation du model décision : d'abord il est conçu à partir des chirurgies qui sont valide et que nous avons séparé grâce à la la classe DataModel. En principe il est sous forme d'un entrepôt représenté par une map à différent niveaux la figure ci-dessous montre la structure choisie pour la répartition des chirurgies :



Cette structure me permet de repartir les chirurgies sur un espace de trois dimension qui sont :(chirurgien, salle et jours de la semaine) :

Et de là je peux avoir des informations sur chaque dimension et sur combinaison de deux ou trois (Ex : l'ensemble des chirurgies qu'avait fait le chirurgien X, toutes les chirurgies qui sont faites dans la salle j, toutes les chirurgies qui sont faites dans la salle J *par le chirurgien C ...etc.*), et de là on justifier la surcharge de la méthode allof()de la classe ModelDecision qui m'offre des accès multicritères et qui me renvoie la liste des chirurgies satisfaisants les critère fournis en paramètre. De cette liste j'ai constaté que nous pouvons extraire 4 information très essentielles qui représentent mes critères de décision et qui sont définis dans l'énumération SousCritere de cette manière :

Proba : qui représente la probabilité d'apparition de mes clés de recherche par exemple :

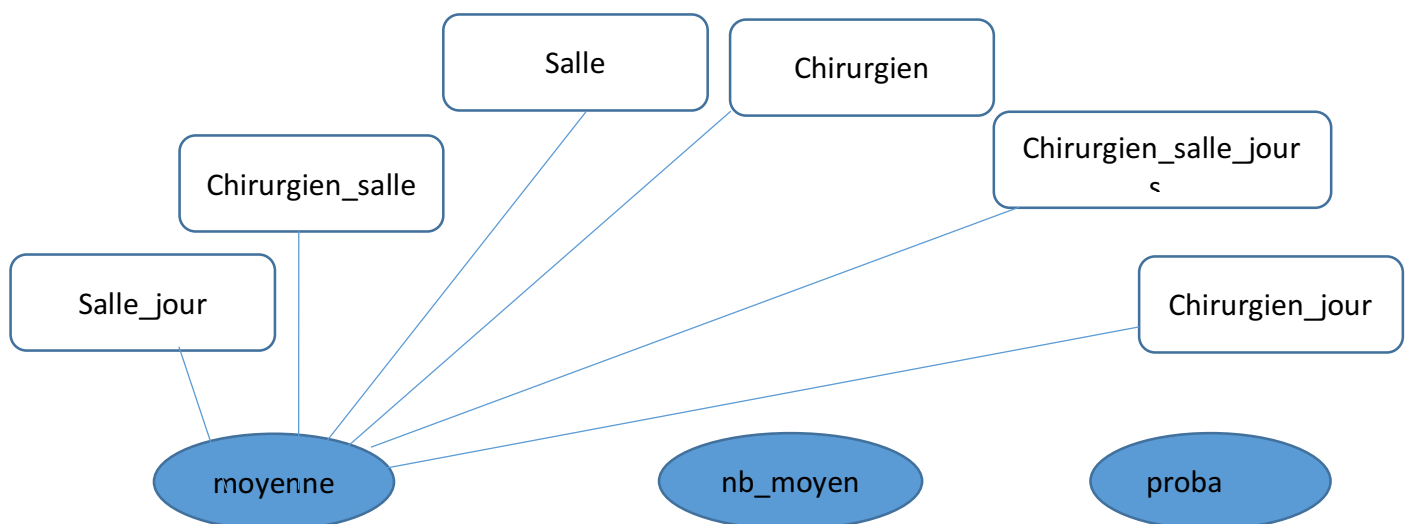
Si on fait une recherche selon le chirurgien X et qu'on applique Probe : on aura comme résultat la probabilité que ce chirurgien X travaille, et si on recherche selon un chirurgien X et une salle S dans ce cas nous aurons la probabilité que ce chirurgien X travaille dans la salle S ou en d'autres termes la probabilité que la salle S sera occupée par le chirurgien X ...etc.

Le deuxième critère c'est la moyenne des durée d'un ensemble de chirurgies que j'ai nommé avg.

Le 3eme est le nombre moyen de chirurgie effectué une même date ; par exemple si on applique ce critère sur l'ensemble des salle d'un chirurgien X cela veut dire le nombre moyen de chirurgie que ce chirurgien effectue par jour ...etc.

Le quatrième c'est de prendre en compte la chirurgie suivante et la précédente chronologiquement et de comparer l'écart d'une chirurgie susceptible de se faire modifier avec l'écart moyen

La figure qui suit montre la structure de mon model de décision à deux niveaux



Ensuite après avoir calculer les différentes valeurs associées à ces critères j'ai choisi de choisir la méthode des sommes pondérées en calculant un score(C) pour la chirurgie C.

Par contre j'ai remarqué que les échelles de mes critères sont très variables (Ex : proba [0,1], durée moyenne [0, N] et N dépend de l'unité de temps choisie ...etc.

Solution : Normalisation des données.

Pour résoudre ce problème et puisque les proba sont comprises entre 0 et 1 j'ai décidé de normaliser les autres critères en utilisant le minimax scaler qui me permet d'avoir des valeurs comprises entre 0 et 1.

Une fois normalisés, on remarque bien que les critères n'ont pas un même degré d'importances pour cela j'ai affecté des poids arbitraires pour les critères de la façon suivante :

Critère	proba	Durée moyenne	Nb moyen
Poids affecté	10	-4	-1

#### Justification :

Le critère le plus important est celui de probabilité et égale à la somme des autres.

Les critères dont les poids sont négatifs sont à minimiser.

Aussi pour les combinaisons (salle, chirurgien salle ...etc.) elles n'ont pas réellement le même poids pour cela je leur ai associé des coefficients dans le calcul de score qui est une somme de ces derniers :

Combinaison	Salle	chirurgien	Salle_jour	Ch_jour	Ch_sal_jou	Ch_salle
Poids	0,15	0,25	0,15	0,25	0,25	0,30

Pour le calcul du score il se fait de la manière suivante :

Pour chaque combinaison (salle, salle chirurgien ...etc.) on lui affecte son score qui est la somme pondérée des valeurs associées aux critères multipliée fois leur poids

Valeur (Chirurgie c, salle s) =  $10 \text{ proba}(s) - 5 \text{ durée Moyenne}(s) - \text{nombre}(\text{chirurgie})$ .

Une fois on a eu les scores pour toutes les combinaisons on fait la somme pondérée selon le tableau 2, ce qui nous permettra d'avoir le score global pour une chirurgie.

### 3-b Model de résolution :

Nous avons trois type de conflit et chaque type

1-ubiquité : les deux chirurgies partagent la salle.

2-interference : quand les chirurgies partagent le chirurgien.

3-cheveauchement : qui est une conjonction des deux types précédent (c.à.d. les chirurgies partagent la salle et le chirurgien)

### Méthodes de résolution :

#### 1- Pour les conflits de type ubiquité :

On cherche les salles qui étaient disponibles sur la période des deux chirurgies

Si on ne trouve pas pour les deux ->on s'est trompé dans les dates.

Si on trouve pour une mais pas pour l'autre ->donc on modifie la salle pour celle

Dont il existe des salle disponible au moment ou elle se déroulait.

Si on trouve des salles qui se sont disponible pour les deux chirurgies :

On choisit celle qui a un Min (score (c)) entre les deux.

Une fois on a décidé laquelle des chirurgies on lui corrige la salle

-si on trouve une seule salle disponible pour cette chirurgie je remplace celle de

Cette dernière avec cette salle.

-s'il y a plus d'une salle dans ce cas là j'ordonne ces salles selon le score de cette chirurgie après avoir substituer la salle de cette dernière avec chacune des salles proposées, puis je choisis celle qui a un score max.

#### 2- Pour les conflits de type interférence :

Le même raisonnement que le type ubiquité et valide pour celui-là sauf qu'au lieu de manipuler les salles on manipule les chirurgiens puisque c'est le chirurgien qui est en cause du conflit.

#### 3- Pour ceux de type chevauchement :

Nous devons modifier la salle et le chirurgien de la même manière que les deux types précédent : si cela et impossible dans le cas ou aucun chirurgien et aucune salle ne pourraient être proposer pour l'une des chirurgies alors dans ce cas on est devant une redondance ou une erreur sur les dates.

Sinon on sera devant plusieurs scenarios existent ; la figure suivante illustre l'ensemble de ces scenarios.

La chirurgie C1		La chirurgie C2	
Salle	Chirurgien	salle	chirurgien

Oui	Oui		
		oui	oui
oui			oui
	Oui	oui	

Donc le tableau résume les cas qui sont : modifier la chirurgie et la salle pour l'une des chirurgies ou modifier la salle pour l'une et la chirurgie pour l'autre. Pour chaque cas on peut avoir une ou plusieurs combinaison (salle-chirurgien) qui la satisfait comme on peut ne pas avoir aucune.

Pour cela on essaie d'appliquer toutes les modifications possibles, puis calculer en même temps le score des deux chirurgies, puis on choisit la combinaison qui aboutit à un meilleur score pour les deux chirurgies.

Interprétation des résultats :

Les résultats étaient moyennement bien et les statistiques concernant ces derniers étaient comme suit :

	ubiquité	interférence	chevauchement	total
totaux	248	28	99	375
résolus	142	24	70	236
Pourcentage	57,25	85,71	70,70	62,93

On en conclue que le model est moyennement bien par contre on remarque à partir de ces résultats que plus nous avons des conflits moins le model est efficace.

A savoir il existe beaucoup de méthodes permettant la résolution de ces types de problèmes

Qui sont des méthodes machine Learning mais il n'existe pas des APIs fournies dans le langage java.