

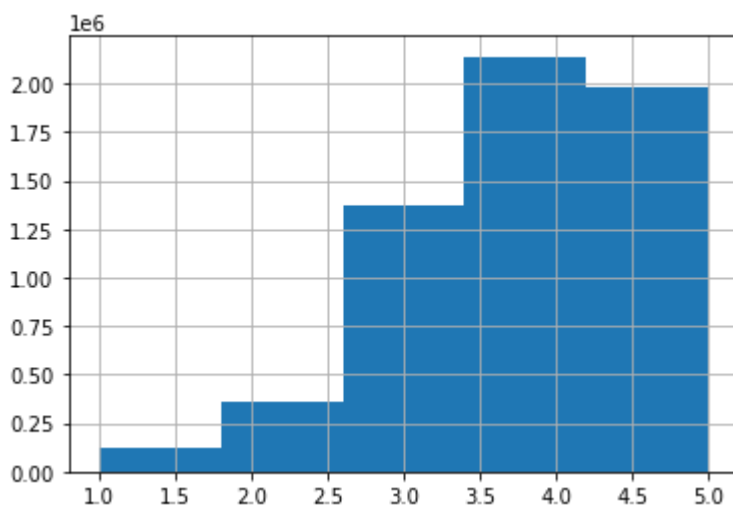
# Book-Adviser

## Exploration des données:

Premièrement nous avons décidé de faire l'exploration des données envoyées par le client.

Exploration ratings.csv :

- Ce sont les différentes notes données par les utilisateurs aux différents livres



- On remarque qu'ils y à très peu de notes basses (1-2) et beaucoup de notes hautes (4-5)

```
Entrée [30]: 1 len(ratings["user_id"].unique())
```

```
Out[30]: 53424
```

```
Entrée [31]: 1 ratings["user_id"].max()
```

```
Out[31]: 53424
```

- Les ids des utilisateurs sont uniques et continus

```
Entrée [12]: 1 len(ratings["book_id"].unique())
Out[12]: 10000

Entrée [13]: 1 ratings["book_id"].max()
Out[13]: 10000
```

- Les ids des livres sont aussi uniques et continus.

```
dtype='object')

Entrée [21]: 1 reviews_per_book = ratings.groupby("book_id")["book_id"].count()

Entrée [24]: 1 reviews_per_book.describe()

Out[24]: count      10000.000000
         mean         597.647900
         std        1267.289788
         min           8.000000
         25%        155.000000
         50%        248.000000
         75%        503.000000
         max       22806.000000
         Name: book_id, dtype: float64
```

- Tous les livres ont au minimum 8 notes, ce qu'on peut considérer comme suffisant pour la création d'un système de recommandation collaboratif.

```
Entrée [25]: 1 reviews_per_user = ratings.groupby("user_id")["book_id"].count()

Entrée [26]: 1 reviews_per_user.describe()

Out[26]: count      53424.000000
         mean       111.868804
         std        26.071224
         min        19.000000
         25%        96.000000
         50%       111.000000
         75%       128.000000
         max       200.000000
         Name: book_id, dtype: float64
```

```
Entrée [ ]: 1
```


- Chaque utilisateur a donné un minimum de 19 reviews, ce qui est également suffisant pour la création d'un système de recommandation collaboratif.

- On ne se sépare donc d'aucun utilisateur et d'aucun livre.

## Gestion des catégories :

En premier lieu, nous avons remarqué que la section de tag était très dure à utiliser telle quelle, car le champ avait été laissé libre. Certains tags ne faisaient pas sens et certains avaient plusieurs orthographes différentes. Nous avons alors eu l'idée de chercher des tags en ligne, et nous avons trouvé l'API du site openlibrary qui permettait de faire des correspondances par ISBN et qui avaient des catégories bien définies et pertinentes.

On pouvait retrouver les infos sous format JSON :

JSON	Données brutes	En-têtes
Enregistrer	Copier	Tout réduire   Tout développer    Filtrer le JSON
▼ ISBN:9781407109084:		
url:	"https://openLibrary.org/books/OL27312857M/The_Hunger_Games"	
key:	"/books/OL27312857M"	
title:	"The Hunger Games"	
▼ authors:		
▼ 0:		
url:	"https://openLibrary.org/authors/OL1394359A/Suzanne_Collins"	
name:	"Suzanne Collins"	
▼ 1:		
url:	"https://openLibrary.org/authors/OL9119339A/Tatiana_Maslany"	
name:	"Tatiana Maslany"	
number_of_pages:	458	
▼ identifiers:		
▼ amazon:		
0:	"1407109081"	
▼ isbn_10:		
0:	"1407109081"	
▼ isbn_13:		
0:	"9781407109084"	
▼ openlibrary:		
0:	"OL27312857M"	
▼ classifications:		
▼ lc_classifications:		
0:	""	
▼ publishers:		
▼ 0:		
name:	"Scholastic Press"	
publish_date:	"Sep 17, 2009"	

Et nous avons créé le script permettant de récupérer les catégories :

```
def cat_retrieve(isbn): ...  
... ['severe poverty', 'starvation', 'oppression', 'effects of war', 'self-sacrifice', 'Science  
fiction', 'Apocalyptic fiction', 'Dystopian fiction', 'Fiction', 'Juvenile works', 'Novels', 'Young  
adult works', 'Juvenile fiction', 'contensts', 'Young adult fiction', 'Game shows', 'Television  
programs', 'New York Times bestseller', 'Contests', 'nyt:series_books=2010-08-21', 'Long Now Manual  
for Civilization', 'Reality television programs', 'Television game shows', 'Survival',  
'Interpersonal relations', 'Roman', 'Amerikanisches Englisch', 'Sisters', 'Young women',  
'Dystopias', 'Survival skills', 'Blind', 'Books and reading', 'Reading Level-Grade 9', 'Reading  
Level-Grade 8', 'Reading Level-Grade 11', 'Reading Level-Grade 10', 'Reading Level-Grade 12',  
'Survival Stories', 'Action & Adventure', "Children's fiction", 'Survival, fiction', 'Interpersonal  
relations, fiction', 'Contests, fiction', 'Television, fiction', 'Large type books', 'Future',  
'violent', 'life risking', 'bravery.', 'Roman pour jeunes adultes', 'Habilités de survie', "Roman  
d'aventures", 'Concours et compétitions', 'Relations humaines', 'Romans, nouvelles', 'Émissions  
télévisées', 'Spanish language materials', 'Supervivencia', 'Novela juvenil', 'Relaciones humanas',  
'Programas de televisión', 'Concursos', 'YOUNG ADULT FICTION']
```

Cependant nous nous sommes alors rendu compte que les ISBN dans nos jeux de données étaient erronés et qu'il n'y avait pas d'autres moyens de faire le lien entre les données locales et celles du site. Nous avons dans le dossier "Data" un fichier "book\_categories", qui présente les démarches que l'on a pu avoir lors de cette recherche.

Nous avons donc travaillé le jeu de données des tags de base.

```
["to-read", "currently-reading", "books-i-own", "owned", "favourites", "favorites", "owned-books"]
```

Nous avons décidé de faire une liste des tags populaires qui n'apportent que peu de pouvoir explicatif pour pouvoir les écarter.

Puis nous avons pris les 5 tags les plus attribués par les utilisateurs pour chaque livre pour pouvoir faire un algorithme content-based.

28241	32075671	The Hate U Give	9569	young-adult contemporary fiction
28242	33288638	Wait for It	8892	romance contemporary contemporary-romance
28243	33288638	Wait for It	8892	romance contemporary contemporary-romance
28244	33288638	Wait for It	8892	romance contemporary contemporary-romance

Nous avons au départ pris seulement trois tags, mais on peut voir beaucoup de redondance dans les tags sur certains livres, et nous avons donc augmenté ce chiffre à 5.

Au delà de 5, les catégories devenaient parfois un peu trop spécifiques et n'apportaient donc que du “bruit” supplémentaire.

### **Choix et pertinence des modèles :**

Nous avons décidé de faire travailler de concert un modèle collaboratif et un modèle content-based.

Pour le modèle content-based, nous nous sommes servis des tags que nous avons filtré comme expliqué au dessus. Nous avons simplement opéré une `cosine_similarity`, pour proposer les livres avec les tags les plus proches du livre que nous avons sélectionné comme base.

Pour le modèle collaboratif, nous avons passé une grande partie de notre temps à chercher des modèles différents et des moyens de calculer leur pertinence.

Une des premières solutions que nous avons testé était d'utiliser la librairie “Surprise”, qui se spécialise dans les algorithmes de recommandation. Il y a dans le dossier “Data”, un fichier “Modele\_surprise” qui présente quelque peu la démarche que nous avons eu avec cette librairie. Nous avons implémenté un algorithme SVD. Nous avons cherché les meilleurs hyperparamètres avec un GridSearch, et nous avons atteint les métriques

suivantes (sur 5 splits de cross-validation) :

Evaluating RMSE, MAE of algorithm SVD on 5 split(s).

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std
RMSE (testset)	0.8303	0.8293	0.8300	0.8309	0.8303	0.8302	0.0005
MAE (testset)	0.6405	0.6402	0.6405	0.6415	0.6409	0.6407	0.0005
Fit time	182.41	182.35	181.68	183.13	182.94	182.50	0.51
Test time	9.40	9.78	9.76	9.54	9.23	9.54	0.21

Cependant, je ne sais pas si c'est un problème interne à la librairie, ou tout simplement un mauvais entraînement du modèle. Mais nos différents estimateurs ne voulaient à tout prix que prédire la note 3.92, quels que soient l'utilisateur et le livre concernés. Je pense que cette note permettait d'atteindre des métriques élevées car la moyenne des notes devait en être très proche.

Nous avons donc dû chercher ailleurs pour pouvoir implémenter un modèle collaboratif.

Nous avons aussi cherché du côté des réseaux de neurones et de Tensorflow.

Nous avons écrit une grande partie de la création de l'algorithme, mais nous nous sommes trouvés bloqués par une erreur, que nous n'avons toujours pas réussi à résoudre aujourd'hui. Il y a dans le fichier "Modele\_tensorflow" les démarches que nous avons eues.

Nous nous sommes donc tournés vers les SVD d'autres librairies telles que numpy ou Sklearn. Les algorithmes fonctionnaient bien en tant que tel, mais le problème était que les fichiers devaient soit être entraînés(ou tout du moins trouver les poids pour notre jeu de données en particulier) lors du lancement du programme, ce qui pouvait mettre longtemps, soit être exportés, mais on se retrouvait alors avec des fichiers de plusieurs Giga-Octets à devoir faire télécharger à l'utilisateur. On peut voir un des

modèles que nous avons développé dans le fichier “Modele\_KNN” dans le dossier “Data” et un autre dans le fichier “Modele\_SVD” du même dossier.

### **Comportement du programme :**

N’ayant pas encore réussi à intégrer un modèle qui puisse se charger rapidement et/ou sans nécessiter d’exports de fichiers volumineux. Le programme ne fonctionne qu’à moitié pour le moment.

L’idée actuellement est simple, lors du lancement, le programme nous demande si nous sommes un ancien utilisateur ou un nouvel utilisateur et nous conduit dans des fonctions différentes selon la réponse.

Si on dit être un nouvel utilisateur, l’idée sera de demander à cet utilisateur un livre qu’il a pu lire, et lui faire des recommandations en fonction de ce livre grâce à l’algorithme content-based. Cela permet d’aider l’utilisateur à se lancer pour pouvoir dépasser l’étape du cold-start.

Dans le programme comme il est actuellement. Nous calculons la cosine similarity au lancement, car cela reste très rapide. Nous avons implémenté de manière arbitraire, le fait que l’utilisateur indique avoir lu le premier livre Harry Potter, nous lui recommandons donc 10 livres en rapport avec celui qu’il indique avoir lu.

Nous n’avons pas pu implémenter la partie pour un ancien utilisateur faute de modèle pour le moment. Cette fonction aurait sans doute été assez courte, car elle aurait fait appel à une fonction de génération de candidats, puis à une fonction de ranking, et aurait simplement renvoyé le résultat après ces étapes.

Pour la génération de candidats, nous voulions générer 100 candidats par collaborative filtering. L’idée était d’extrapoler les différentes notes qu’un utilisateur aurait pu mettre à différents livres, et de prendre les 100 meilleures. On peut voir cet algorithme en action dans le fichier

Modele\_SVD, et également dans la fonction “recommend\_books\_SVD” du fichier main (elle est entièrement commentée, faute de données et d’algorithme à disposition).

Ces 100 candidats sont générés avec une note pouvant dépasser les limites de 0 et 5 du jeu de base, (mais sans aller dans les extrêmes), nous avons décidé de ne pas limiter ces notes à 0 et 5 pour la simple raison que cela nous permet de mieux ordonner les choix si on a une note à 5.05 et une note à 5.1, que si nous avons deux notes à 5.

L’idée était ensuite de générer 100 candidats avec un content-based, se basant sur les livres que l’utilisateur a déjà lu. On les ordonnerait également en fonction de leur similarité et nous donnerions des notes selon le bracket : Pour les 10 premiers (les plus similaires aux goûts de l’utilisateur), nous pourrions donner 5 (ça serait la forme la plus simple que nous voudrions essayer dans un premier temps. Pour le futur, on peut également penser à calquer cette note sur la note maximale que l’utilisateur a pu donner par le passé ou autre). Pour les candidats classés de 11 à 30 on donnerait 4, pour les candidats de 30 à 60 on donnerait 3, et pour les candidats de 60 à 100, on donnerait 2.

L’idée de mapper des notes à des “buckets” permet plusieurs choses : déjà de pallier le manque de précision des valeurs de similarités : vu que l’on ne prend que 5 tags pour calculer cette similarité, on aura sans doute beaucoup de candidats qui partagent par exemple 3 tags avec notre livre cible, et qui auront donc le même score, et les noter différemment serait donc étrange. Même s’il est peu probable que les buckets tombent exactement juste sur les changements de score, on sait qu’en moyenne des livres avec une similarité commune, auront la plupart du temps, une note commune. Ensuite, le fait de mapper des notes de 2 à 5 semble logique



plutôt que d'aller jusqu'à 0, car il ne faut pas oublier que c'est une base de livre que l'utilisateur est censé pouvoir apprécier.

L'idée serait alors d'additionner ces deux notes et de proposer les N livres les plus intéressants après la mise en place des deux algorithmes.

L'avantage de cette méthode des notes additives, est qu'elle permet d'ajouter des paramètres facilement si on le souhaite.

Par exemple, imaginons que l'utilisateur indique ne pas vouloir de livres trop populaires, on pourrait par exemple ajouter une note inversement proportionnelle à la popularité, et prendre le résultat.

Pour ce qui est de la sérendipité, on peut avoir deux points de vue :

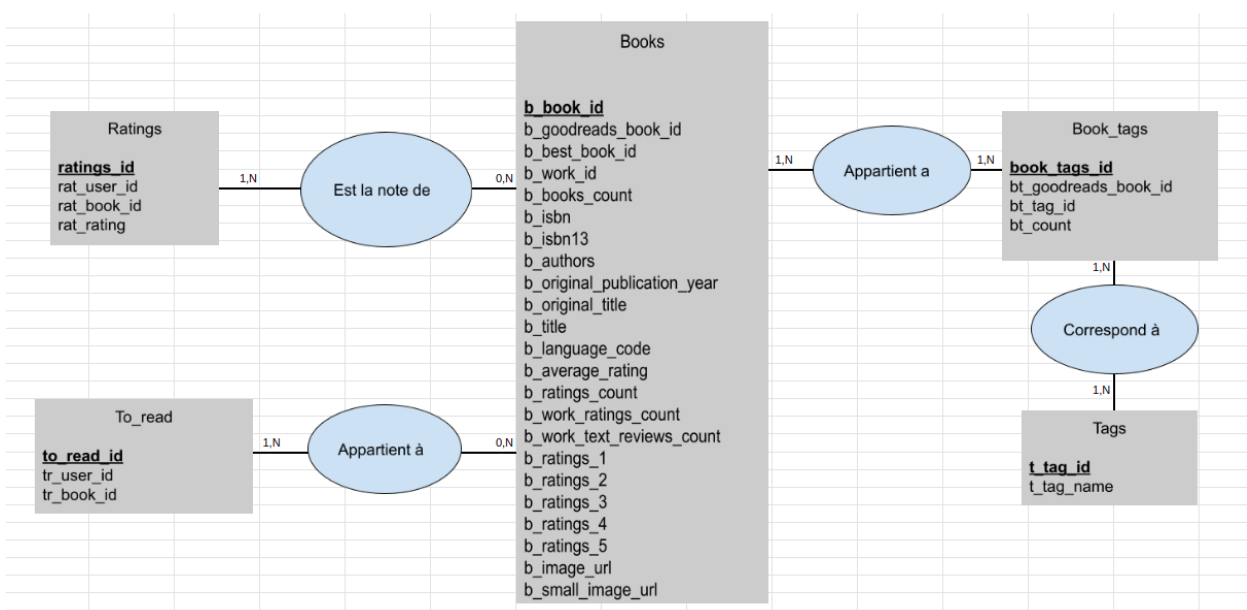
- Le premier serait de dire que l'algorithme content-based apporte déjà son lot de sérendipité quant à la popularité des livres. Les tags étant attribués (on peut le penser) plus ou moins uniformément, on a autant de chance de tomber sur un best-seller qu'un livre moins connu en fonction des tags.
- Le deuxième étant de se dire que l'on VEUT proposer à tout prix des livres avec des catégories différentes, mais qui peuvent nous intéresser quand même , et à ce moment-là il suffirait de revoir les termes de l'addition. Par exemple, on pourrait se dire qu'une fois les similarités calculées avec le content-based, on prendrait les mêmes types de buckets en attribuant toujours les mêmes notes mais en valorisant cette fois si les similarités les plus faibles. Donc on aurait à la fois des livres que des utilisateurs qui nous sont semblables ont lu, mais qui sont également très différents des livres que nous avons l'habitude de lire.

**Base de données :**

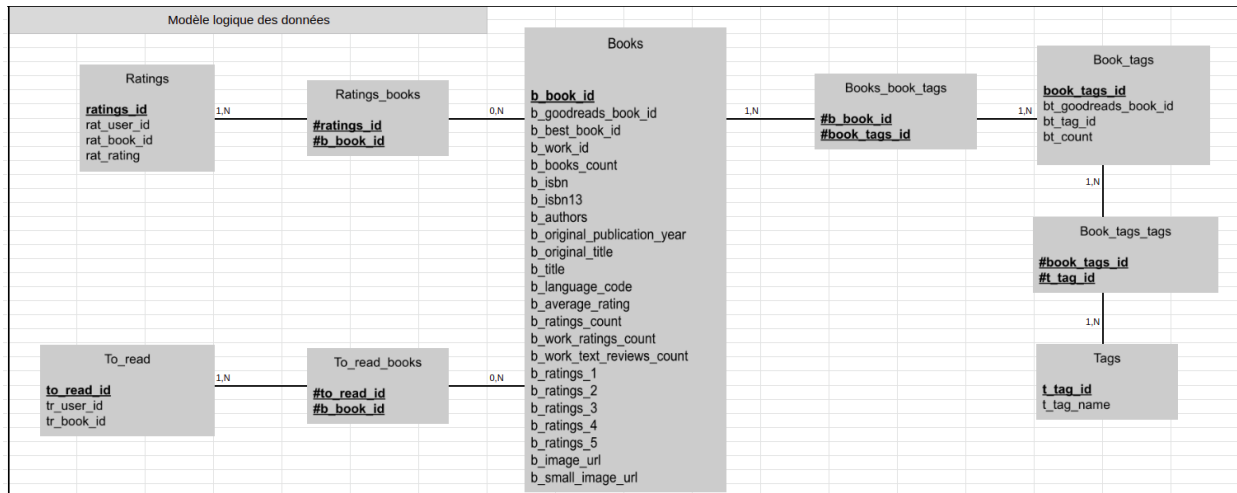
## Dictionnaire des données

Libellé	Type	Taille	Description
Rating			
<u>ratings_id</u>	INT	10	Id pour chaque user/livre/rating
rat_user_id	INT	10	Id des utilisateur ayant mis une note à un livre
rat_book_id	INT	10	Id des livres ayant une note attribué par un utilisateur
rat_rating	INT	10	Note attribué par un utilisateur à un livre
Books			
<u>b_book_id</u>	INT	5	Id des livres
b_goodreads_book_id	INT	10	Id des édition les plus populaire pour chaque livre
b_best_book_id	INT	10	Id des édition les plus populaire pour chaque livre
b_work_id	INT	10	Id du livre dans un sens
b_books_count	INT	5	Nombre d'exemplaire d'un livre
b_isbn	VARCHAR	15	Numéro international normalisé du livre (10 chiffres)
b_isbn13	VARCHAR	30	Numéro international normalisé du livre (13 chiffres)
b_authors	VARCHAR	100	Nom de l'auteur du livre
b_original_publication_year	FLOAT	10	Année de publication originale du livre
b_original_title	VARCHAR	100	Titre du livre dans sa version originale
b_title	VARCHAR	100	Titre du livre
b_language_code	VARCHAR	10	Langue du livre
b_average_rating	FLOAT	5	Note moyenne par livre
b_ratings_count	INT	10	Nombre de total de note pour chaque livre
b_work_ratings_count	INT	10	
b_work_text_reviews_count	INT	10	
b_ratings_1	INT	10	Nombre de rating 1 attribué à un livre
b_ratings_2	INT	10	Nombre de rating 2 attribué à un livre
b_ratings_3	INT	10	Nombre de rating 3 attribué à un livre
b_ratings_4	INT	10	Nombre de rating 4 attribué à un livre
b_ratings_5	INT	10	Nombre de rating 5 attribué à un livre
b_image_url	VARCHAR	150	lien de l'image de couverture du livre
b_small_image_url	VARCHAR	150	lien de l'image de couverture du livre version small
Book_tags			
<u>book_tags_id</u>	INT	10	Id des book_tags
bt_goodreads_book_id	INT	10	Id des édition les plus populaire pour chaque livre
bt_tag_id	INT	10	Id du tag
bt_count	INT	10	nombre de tag par livre marqué "à lire" ?
Tags			
<u>t_tag_id</u>	INT	5	Id du tag
t_tag_name	VARCHAR	50	nom du tag (ex : "-history")
To_read			
<u>to_read_id</u>	INT	10	ID des livre to_read
tr_user_id	INT	10	Id des utilisateurs ayant marqué un livre comme "à lire"
tr_book_id	INT	10	Id des livres marqué comme "à lire"

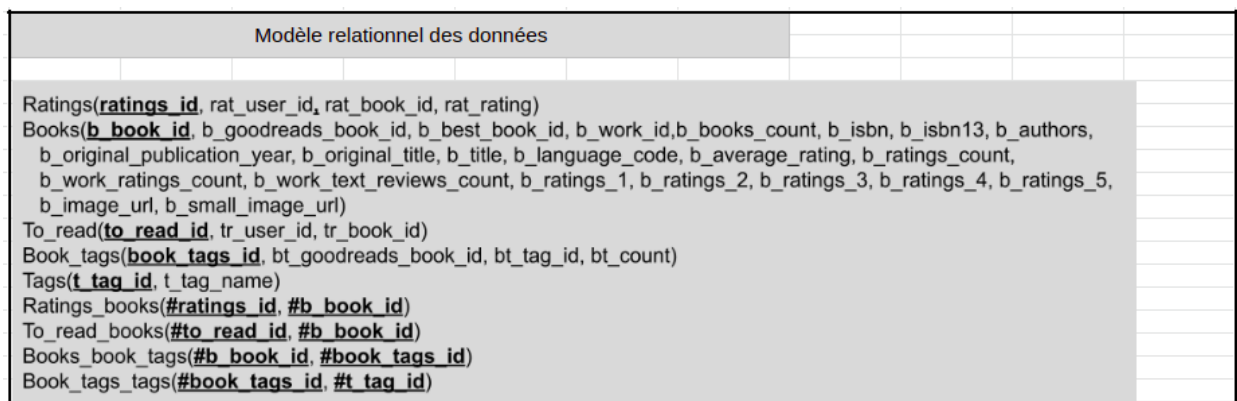
## Modèle conceptuel des données

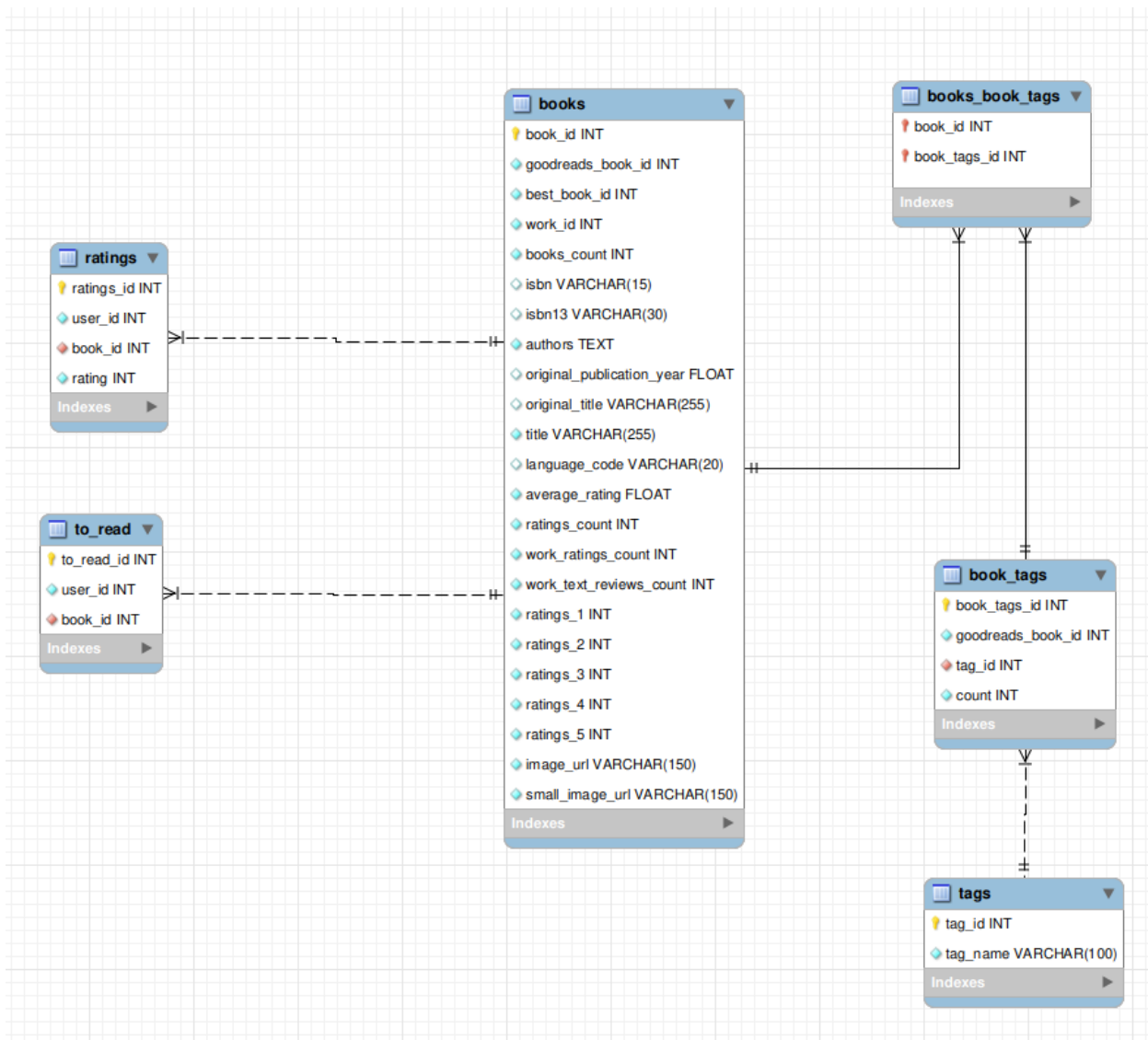


## Modèle logique des données



## Modèle relationnel des données





## UPDATE (01/10):

Nous avons compris d'où venait l'erreur rencontrée avec la librairie "Surprise".

```

uid = str(196) # raw user id (as in the ratings file). They are **strings**!
iid = str(302) # raw item id (as in the ratings file). They are **strings**!

# get a prediction for specific users and items.
pred = algo.predict(uid, iid, r_ui=4, verbose=True)
  
```

La documentation semblait spécifier vouloir des strings, cependant l'erreur venait justement du fait que l'on donnait des strings à la méthode "predict". On peut voir sur ces tests : qu'à chaque fois que l'on passe au moins un string le résultat est toujours le même. Et que l'algorithme se met à fonctionner lorsque l'on passe des integers à la place.

```
Entrée [12]: 1 prediction = algo.predict("1", "258")
              2 prediction.est
Out[12]: 3.9198655261735214

Entrée [15]: 1 prediction = algo.predict("3", "258")
              2 prediction.est
Out[15]: 3.9198655261735214

Entrée [16]: 1 prediction = algo.predict("3", 258)
              2 prediction.est
Out[16]: 3.9198655261735214

Entrée [18]: 1 prediction = algo.predict(1, 258)
              2 prediction.est
Out[18]: 4.07279152359331
```

Après plusieurs essais nous voulions donc créer un algorithme collaboratif se basant sur les utilisateurs, et les modèles KNN semblaient avoir des meilleurs résultats que les SVD. Cependant lorsque l'on fait travailler les KNN sur les utilisateurs, nous nous confrontons à un problème de mémoire :

```
MemoryError: Unable to allocate 21.3 GiB for an array with shape (53424, 53424) and data type float64
```

Nous avons donc fait un filtrage collaboratif se basant sur les items, ce qui semblait être plus indiqué par la documentation de toute façon.

Le programme est maintenant fonctionnel bien que dépourvu d'interface graphique pour le moment.

Nous avons changé la façon de calculer les différents scores.

On calcule maintenant dans la fonction “collaborative\_generation”, une prédiction pour chaque livre non lu par un utilisateur donné (en fonction des autres utilisateurs).

Puis dans la fonction “content\_based\_generation” nous calculons pour chaque livre, un score de similarité sur 1 avec un livre donné.

Puis dans la fonction “user\_content\_based\_generation” nous calculons de la même façon des scores de similarités entre les livres déjà lus par un utilisateur et ceux qu'il n'a pas encore lu.

Enfin, dans la fonction “merge\_recommendations” nous générons un certain nombre de candidats grâce à “collaborative\_generation”, puis nous ajoutons à ces candidats (et à leur note prédite), le score de similarité (multiplié par un facteur décidable).

Cela permet de se baser en premier lieu sur l'approche collaborative, et de donner le poids que l'on souhaite au contenu, nous permettant ainsi plus ou moins de sérendipité selon nos choix.

