Guillaume BELDILMI

Projet Systèmes de Gestion de Documents

Partie 1: Conception et mise en œuvre de la base de données MongoDB

Cahier des charges

Demandes client

On s'intéresse, ici, aux données concernent les films qui ont été diffusés dans des salles de cinéma à des fins d'analyse.

Par exemple, pour un film, on peut vouloir connaître sa description, les salles où il a été diffusé, la durée de diffusion, le nombre d'entrées, etc.

L'objectif de ces informations sera d'avoir des statistiques générales pour l'ensemble des cinémas d'une ville, mais aussi de façon plus personnalisée sur un film, ou un cinéma, etc.

En complément de ces informations, doivent être stockés et mis en relation les commentaires faits sur les réseaux sociaux sur les films qui ont été diffusés. Il peut s'agir des commentaires textuels et/ou de notes d'appréciation pour le film.

Les informations sur les films sont les informations usuelles (titre, réalisateur, durée, etc.) ainsi que sa catégorie pour permettre des recherches par catégorie. Les informations issues des réseaux sociaux pourront venir enrichir l'analyse pour un film donné, ou une catégorie de film, etc.

Premier jet de la base de données

De par ces demandes, on peut déjà identifier les nos objets métiers principaux:

- Les films et leurs caractéristiques
- Les cinémas et leurs salles
- Les avis sur les films

On peut donc déjà identifier les collections suivantes:

- Collection de films
- Collection de cinémas
- Collection d'avis

Un objet de la collection de films contiendrait les champs suivants:

- le titre du film
- sa durée
- sa date de sortie
- le synopsis du film
- la liste des réalisateurs ayant participé à la réalisation de ce film
- la liste des acteurs ayant joué dans ce film
- la liste des genres auxquels appartient ce film

Un objet de la collection de cinémas contiendrait les champs suivants:

- le nom du cinéma
- l'adresse du cinéma
- la ville où se trouve le cinéma
- la liste des salles de cinéma disponibles dans ce cinéma, avec, pour chaque salle:
    - le nom de la salle
    - le nombre de places disponibles dans la salle
    - la liste des films diffusés dans cette salle, avec, pour chaque film:
        - la référence au film concerné
        - la date de diffusion du film
        - le nombre d'entrées pour ce film

Un objet de la collection d'avis contiendrait les champs suivants:

- une référence au film concerné
- une note sur le film
- le commentaire textuel sur le film
- la source de l'avis
- l'auteur de l'avis

Conception de la base de données

Données primaires

On a donc les collections suivantes:

- Collections de `films`: cette collection répertorie les films disponibles dans la base de données. Chaque document de cette collection contient les champs suivants:
    - `titre`: titre du film
    - `duree`: durée du film
    - `date_sortie`: date de sortie du film
    - `synopsis`: synopsis du film
    - `realisateurs`: liste des réalisateurs ayant participé à la réalisation de ce film
    - `acteurs`: liste des acteurs ayant joué dans ce film
    - `genres`: liste des genres auxquels appartient ce film
- Collections de `cinemas`: cette collection répertorie les cinémas disponibles dans la base de données. Chaque document de cette collection contient les champs suivants:
    - `nom`: nom du cinéma
    - `adresse`: adresse du cinéma
    - `ville`: ville où se trouve le cinéma
    - `salles`: liste des salles de cinéma disponibles dans ce cinéma, pour chaque salle, on a les champs suivants:
        - `nom_salle`: nom de la salle
        - `nombre_places`: nombre de places disponibles dans la salle
        - `films_diffuses`: liste des films diffusés dans cette salle, pour chaque film, on a les champs suivants:
            - `film`: référence au film concerné
            - `date_diffusion`: date de diffusion du film
            - `nombre_entrees`: nombre d'entrées pour ce film
- Collections d'`avis`: cette collection répertorie les avis sur les films disponibles dans la base de données. Chaque document de cette collection contient les champs suivants:
    - `film`: référence au film concerné
    - `note`: note sur le film
    - `commentaire`: commentaire textuel sur le film
    - `source`: source de l'avis
    - `auteur`: auteur de l'avis

Mises en liens

Pour des raisons de simplicité, les références aux objets de la collection de `films` se feront sous la forme de l'identifiant de l'objet concerné (`ObjectId` généré automatiquement par MongoDB dans le champs `_id` de chaque objet), et non sous la forme d'un objet complet afin de limiter la redondance des données, faciliter la mise à jour des données et éviter les problèmes de cohérence des données.

En effet, si on stockait l'objet complet de la collection de `films` dans chaque document de la collection d'`avis`, cela nous obligerait à mettre à jour tous les documents de la collection d'`avis` à chaque modification d'un film, ce qui serait coûteux en termes de performances et de ressources et pourrait entraîner des problèmes de cohérence des données dans le cas où la valeur d'un champ d'un film serait erronée.

Cas des realisateurs et des acteurs de films et mises en perspectives

Pour les champs `realisateurs` et `acteurs` de la collection de `films`, on pourrait stocker les noms des réalisateurs et des acteurs sous forme de chaînes de caractères. Cependant, cela pourrait entraîner des problèmes de cohérence des données dans le cas où un nom de réalisateur ou d'acteur serait mal orthographié, sans oublier que deux réalisateurs ou acteurs différents pourraient avoir le même nom.

Pour éviter ces problèmes, on pourrait stocker les réalisateurs et les acteurs sous forme de références à des documents de collections de `personnes`. Cette solution permettrait de stocker des informations supplémentaires sur les réalisateurs et les acteurs (par exemple, leur date de naissance, leur nationalité, etc.) et de faire des recherches sur ces derniers. De plus, cela permettrait aussi de gérer les cas où un réalisateur ait joué dans un film ou un acteur ait réalisé un film.

Ainsi, on pourrait avoir la collection de `personnes` suivante:

- `nom`: nom de la personne
- `prenom`: prénom de la personne

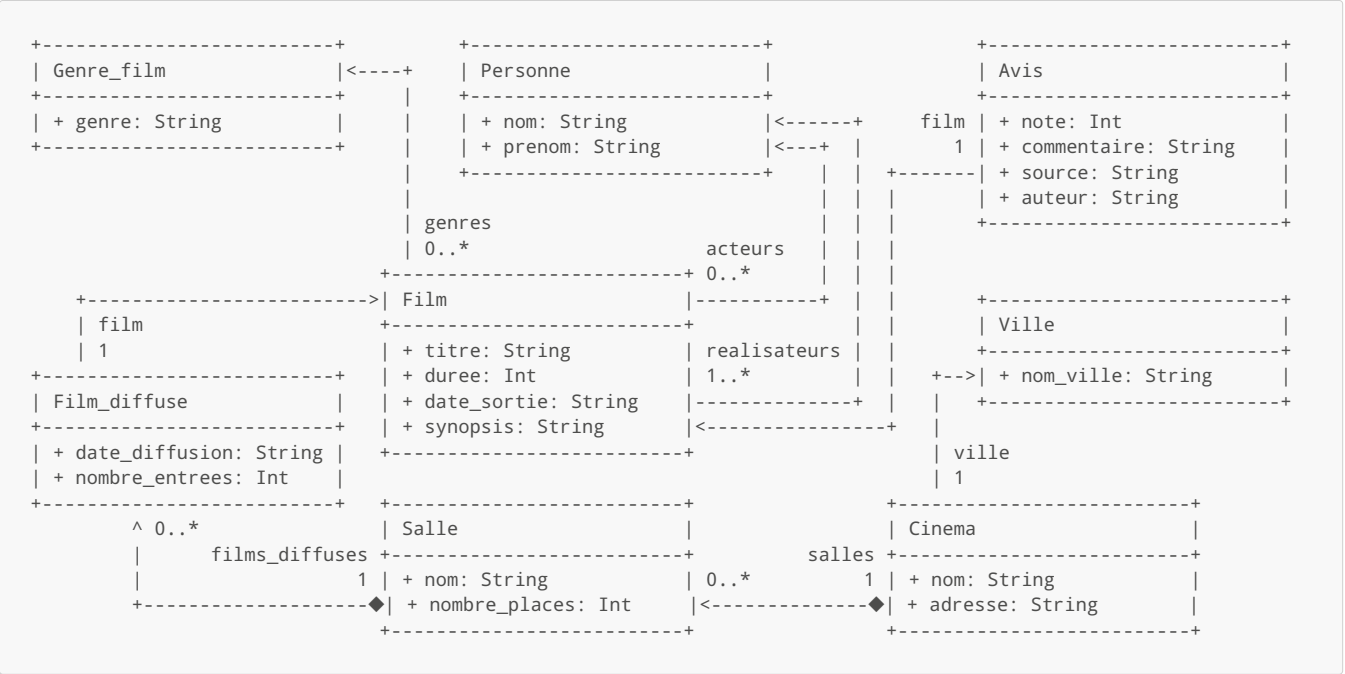
Chaque référence à un réalisateur ou à un acteur dans un document de la collection de `films` serait donc sous la forme de l'identifiant de l'objet concerné dans la collection de `personnes`.

Avec cette même logique, on pourrait aussi avoir des collections `genres_films` et `villes` pour stocker respectivement les genres de films et les villes où un cinéma est implanté. De cette manière, la collection de `genres_films` contiendrait des documents avec comme seul champ `genre`, et la collection de `villes` contiendrait uniquement le champ `nom_ville` (ainsi que le champ `_id` généré automatiquement par MongoDB).

De plus, les mises à jour des documents des collections de `personnes`, `genres_films` et `villes` seraient moins fréquentes que celles des documents de la collection de `films` et `cinemas`, donc leurs mises en cache serait pertinente.

Modélisation UML et schéma de la base de données

La modélisation UML de la base de données sera donc la suivante:



Et le schéma de la base de données sera le suivant:

- Collection `films`:
  - `_id`: *ObjectId*
  - `titre`: *String*
  - `duree`: *Int* (en minutes)
  - `date_sortie`: *String* (format: "YYYY-MM-DD HH:MM")
  - `synopsis`: *String*
  - `realisateurs`: *ObjectId[]*
  - `acteurs`: *ObjectId[]*
  - `genres`: *ObjectId[]*
- Collection `genres_films`:
  - `_id`: *ObjectId*
  - `genre`: *String*
- Collection `personnes`:
  - `_id`: *ObjectId*
  - `nom`: *String*
  - `prenom`: *String*
- Collection `cinemas`:
  - `nom`: *String*
  - `adresse`: *String*
  - `ville`: *ObjectId*
  - `salles`:
    - `nom`: *String*
    - `nombre_places`: *Int*
    - `films_diffuses`:
      - `film`: *ObjectId*
      - `date_diffusion`: *String* (format: "YYYY-MM-DD HH:MM")
      - `nombre_entrees`: *Int*
- Collection `villes`:
  - `_id`: *ObjectId*
  - `nom_ville`: *String*
- Collection `avis`:
  - `film`: *ObjectId*
  - `note`: *Int*
  - `commentaire`: *String*
  - `source`: *String*
  - `auteur`: *String*

N.B.: Les champs référents une date sont stockés sous forme de chaînes de caractères au format "YYYY-MM-DD HH:MM" afin de simplifier la manipulation des dates et heures dans les requêtes.

Extensions possibles

Pour enrichir la base de données, on pourrait ajouter différentes informations supplémentaires, telles que:

- des compléments d'informations sur les personnes (réalisateurs, acteurs, etc.) comme leur date de naissance, leur nationalité, etc.
- des informations sur les genres de films (description, etc.)

Partie 2: Requêtage de la base de données MongoDB

Requêtes simples

Les requêtes suivantes sont des exemples de requêtes que nous pourrions effectuer sur la base de données MongoDB.

Compter le nombre d'éléments dans chaque collection

Après avoir importé les données dans la base de données, nous pourrons effectuer les requêtes suivantes pour compter le nombre d'éléments dans chaque collection:

```
db.genres_films.count();
db.personnes.count();
db.films.count();
db.avis.count();
db.villes.count();
db.cinemas.count();
```

Tous les avis d'un film

Pour obtenir tous les avis d'un film (dont une partie serait vue depuis la page de présentation du film en question), nous pourrions effectuer la requête suivante:

```
f = db.films.findOne({titre: "Interstellar"});
liste_avis = db.avis.find({film: f["_id"]});
```

Cette requête fait partie des requêtes les plus simples que nous pourrions effectuer sur la base de données. Elle consiste à rechercher l'identifiant du film dans la collection `films` puis à rechercher tous les avis qui font référence à cet identifiant dans la collection `avis`. Le résultat de cette requête serait une liste d'avis pour le film "Interstellar".

Tous les films d'un genre

Pour obtenir tous les films d'un genre (dont une partie serait vue depuis la page de présentation du genre en question), nous pourrions effectuer la requête suivante:

```
g_sf = db.genres_films.findOne({genre: "Sci-Fi"});
liste_films_sf = db.films.find({genres: g_sf["_id"]});
```

Cette requête consiste à rechercher l'identifiant du genre dans la collection `genres_films` puis à rechercher tous les films qui font référence à cet identifiant dans la collection `films`. Le résultat de cette requête serait une liste de films de genre "Sci-Fi". Par la suite, nous pourrions affiner notre recherche avec d'autres critères (ex: les films les mieux notés, les films les plus diffusés, etc.) que nous détaillerons dans les sections suivantes.

Note d'appréciation générale d'un film avec MapReduce

Pour obtenir la note d'appréciation générale d'un film, nous pourrions utiliser la méthode `mapReduce` de MongoDB pour effectuer un calcul de moyenne des notes d'appréciation d'un film. Voici un exemple de requête pour obtenir la note d'appréciation générale de tous les films:

```
db.note_g.drop();
var my_map = function() {
  emit(this.film, this.note);
}
var my_reduce = function(film_id, notes) {
  return Array.avg(notes);
}
db.avis.mapReduce(my_map, my_reduce, {out: "note_g"});
note_g_films = db.note_g.aggregate([
  { $lookup: { from: "films", localField: "_id", foreignField: "_id", as: "film" } },
  { $unwind: "$film" },
  { $project: { _id: 0, film: "$film.titre", value: "$value" } }
]);
```

La fonction `my_map` permet de créer une liste de couples (clé, valeur) où la clé est l'identifiant du film et la valeur est la note d'appréciation. La fonction `reduce` permet de calculer la moyenne des notes pour chaque film. L'appel à la méthode `mapReduce` permet d'effectuer ces opérations et de stocker le résultat dans la collection `note_g`.

Le résultat de cette requête est peu exploitable en l'état, car il ne contient que les identifiants des films et les notes d'appréciation moyennes, restant peu lisibles dans l'état. Pour obtenir les titres des films et les notes d'appréciation moyennes, nous effectuons une jointure avec la collection `films` à l'aide de la méthode `aggregate` de MongoDB et des directives `$lookup`, `$unwind` et `$project`.

Les 10 films les plus diffusés

Cette requête peut permettre d'analyser la fréquence de diffusion des films dans les cinémas.

```
db.cinemas.aggregate([
  { $unwind: "$salles" },
  { $unwind: "$salles.films_diffuses" },
  { $group: { _id: "$salles.films_diffuses.film", count: { $sum: 1 } } },
  { $sort: { count: -1 } },
  { $limit: 10 },
  { $lookup: { from: "films", localField: "_id", foreignField: "_id", as: "film" } },
  { $unwind: "$film" },
  { $project: { _id: 0, film: "$film.titre", count: "$count" } }
]);
```

Pour effectuer cette requête, nous utilisons la méthode `aggregate` de MongoDB afin d'effectuer des opérations suivantes à partir des données de la collection `cinemas`:

- Tout d'abord, nous déroulons les tableaux `salles` et `films_diffuses` pour pouvoir accéder aux films diffusés dans chaque salle avec la méthode `$unwind`.
- Ensuite, nous regroupons les films par leur identifiant avec la méthode `$group` en comptant le nombre de fois qu'ils ont été diffusés dans les salles. Puis nous trions les films par le nombre de fois qu'ils ont été diffusés avec `$sort` et nous limitons le résultat à 10 films avec `$limit`.
- Jusqu'ici, nous avons obtenu les identifiants des 10 films les plus diffusés. Pour obtenir les titres de ces films, nous faisons une jointure avec la collection `films` avec la méthode `$lookup`. La méthode `$unwind` nous permet de dérouler les tableaux résultants de la jointure, puis nous projetons les titres des films et le nombre de fois qu'ils ont été diffusés avec `$project`. Nous donnant ainsi les titres des 10 films les plus diffusés triés par ordre décroissant du nombre de fois qu'ils ont été diffusés.

Les noms des 10 cinémas diffusant le plus de films les mieux notés

Cette requête peut permettre d'analyser l'attractivité des cinémas en fonction de l'appréciation des films qu'ils diffusent.

```
db.cinemas.aggregate([
  { $unwind: "$salles" },
  { $unwind: "$salles.films_diffuses" },
  { $lookup: { from: "films", localField: "salles.films_diffuses.film", foreignField: "_id", as: "film" } },
  { $unwind: "$film" },
  { $lookup: { from: "note_g", localField: "film._id", foreignField: "_id", as: "note" } },
  { $unwind: "$note" },
  { $match: { "note.value": { $gte: 4 } } },
  { $group: { _id: "$_id", count: { $sum: 1 } } },
  { $sort: { count: -1 } },
  { $limit: 10 },
  { $lookup: { from: "cinemas", localField: "_id", foreignField: "_id", as: "cinema" } },
  { $unwind: "$cinema" },
  { $project: { _id: 0, cinema: "$cinema.nom", count: "$count" } }
]);
```

Cette requête est plus complexe que les précédentes, car elle nécessite de faire des jointures entre plusieurs collections et de faire des calculs sur les données. Pour cela nous utilisons la méthode `aggregate` de MongoDB afin d'effectuer des opérations suivantes à partir des données de la collection `cinemas`:

- Tout d'abord, nous déroulons les tableaux `salles` et `films_diffuses` pour pouvoir accéder aux films diffusés dans chaque salle avec la méthode `$unwind`.
- Ensuite, nous faisons une jointure avec la collection `films` pour obtenir les informations des films diffusés dans chaque salle avec la méthode `$lookup`.
- Puis, nous faisons une jointure avec la collection `note_g` pour obtenir les notes d'appréciation des films diffusés dans chaque salle, encore une fois, avec la méthode `$lookup`.
- Ensuite, nous filtrons les films qui ont une note d'appréciation supérieure ou égale à 4 avec la méthode `$match`.
- Enfin, nous regroupons les cinémas par leur identifiant avec la méthode `$group`, nous comptons le nombre de films diffusés dans chaque cinéma puis nous trions les cinémas par le nombre de films diffusés avec `$sort` et nous limitons le résultat à 10 cinémas avec `$limit`.
- Jusqu'ici, nous avons obtenu les identifiants des 10 cinémas diffusant le plus de films les mieux notés. Pour obtenir les noms de ces cinémas, nous faisons une jointure avec la collection `cinemas` avec la méthode `$lookup`. La méthode `$unwind` nous permet de dérouler les tableaux résultants de la jointure, puis nous projetons les noms des cinémas et le nombre de films diffusés avec `$project`. Nous donnant ainsi le nom des 10 cinémas diffusant le plus de films les mieux notés triés par ordre décroissant du nombre de films diffusés.

Ajouts à la base de données

Les ajouts à la base de données se feront de manière manuelle, en utilisant les méthodes `insert` et `insertMany` de MongoDB. Les données à ajouter devront être formatées correctement et cohérentes avec les données déjà présentes dans la base de données.

Si plusieurs collections sont concernées par l'ajout, il faudra veiller à mettre à jour les collections dans le bon ordre, en commençant par les collections primaires et en finissant par les collections qui font référence à ces collections primaires afin de garantir le bon référencement des éléments ajoutés.

```
// toujours dans cet ordre...
db.genres_films.insertMany([ ... ]);
db.personnes.insertMany([ ... ]);
db.films.insertMany([ ... ]);
db.avis.insertMany([ ... ]);
db.villes.insertMany([ ... ]);
db.cinemas.insertMany([ ... ]);
```

Traitements des erreurs

Détection des erreurs

Pour la détection des erreurs, nous procéderons comme suit:

- Tout d'abord, nous allons créer une collection `erreurs` dans laquelle nous stockerons les éléments erronés que nous rencontrerons. Cette collection contiendra les champs suivants:
  - `type`: la collection d'origine de l'élément erroné (films, genres_films, personnes, cinemas, villes, avis)
  - `raison`: la raison pour laquelle l'élément est erroné
  - `element`: l'élément erroné
- Ensuite, nous vérifierons que les données sont bien formatées et cohérentes progressivement dans chaque collections (des collections primaires aux collections faisant référence à ces collections primaires).
- Si une erreur est détectée, nous retirerons l'élément de la collection d'origine et nous l'ajouterons à la collection `erreurs`.

N.B: Si l'élément vérifié contient une référence erronée, il sera retiré de la collection d'origine. Ainsi, un élément erroné dans une collection primaire entraînera la détection de tous les éléments qui y font référence comme des erreurs.

Nous détaillerons les détails de l'implémentation de la détection des erreurs dans la troisième partie.

Correction des erreurs et réintégration des données

La correction des erreurs devra se faire manuellement collection par collection, des éléments issus des collections primaires aux éléments. Une fois le traitement de toutes les erreurs d'une collection terminé, nous pourrons essayer de réintégrer les éléments corrigés dans la collection d'origine, ainsi que de tester si le reste des éléments présents dans la collection `erreurs` peuvent être réintégrés sans erreur (ex: correction d'un élément référencé par un autre élément).

Une fois les erreurs corrigées, nous pourrons réintégrer les éléments corrigés dans leur collection d'origine de la façon suivante:

```
// dans le même ordre que pour les ajouts...
e_genres_films = db.errors.find({type: "genres_films"});
e_genres_films.forEach(function(e) {
  db.genres_films.insert(e.element);
  db.errors.remove(e);
});
e_personnes = db.errors.find({type: "personnes"});
e_personnes.forEach(function(e) {
  db.personnes.insert(e.element);
  db.errors.remove(e);
});
e_films = db.errors.find({type: "films"});
e_films.forEach(function(e) {
  db.films.insert(e.element);
  db.errors.remove(e);
});
e_avis = db.errors.find({type: "avis"});
e_avis.forEach(function(e) {
  db.avis.insert(e.element);
  db.errors.remove(e);
});
e_villes = db.errors.find({type: "villes"});
e_villes.forEach(function(e) {
  db.villes.insert(e.element);
  db.errors.remove(e);
});
e_cinemas = db.errors.find({type: "cinemas"});
e_cinemas.forEach(function(e) {
  db.cinemas.insert(e.element);
  db.errors.remove(e);
});
```

Une fois les erreurs corrigées et les éléments réintégrés dans leur collection d'origine, nous pourrons répéter le processus de détection des erreurs pour

vérifier que les éléments réintégrés ne sont plus erronés et ont été corrigés correctement.

Partie 3: Python et pyMongo

Requêtes simples

Les requêtes simples sont en grande partie les mêmes que celles de la partie 2, mais cette fois-ci en Python avec pyMongo.

cf. `queries.py`

Ajouts à la base de données

Les ajouts à la base de données sont effectués selon les mêmes principes que dans la partie 2, mais cette fois-ci en Python avec pyMongo et sa méthode `insert_many` (remplaçant la méthode `insertMany` de MongoDB).

Le contenu remplissant la base de donnée a été généré dans un but de démonstration, et ne contient pas de données pertinentes.

cf. `fill.py`

Traitements des erreurs

Détection des erreurs

Pour la détection des erreurs, nous avons utilisé un script Python avec pyMongo. Ce script parcourt chaque collection de la base de données et leur arborescence et vérifie, dans l'ordre, que:

- le type de donnée de chaque champ est correct.
- les valeurs de chaque champ sont correctes, non nulles et non vides selon les critères définis dans le shéma de la base de données, cohérentes entre elles et cohérentes avec les autres données de la collection, et que les dates sont au bon format.
- les références entre les collections sont correctes et correspondent à des documents existants.

Si une erreur est détectée, le document est ajouté à la collection `errors` avec, comme décrit en partie 2, les champs suivants:

- 'type' étant le nom de la collection d'origine du document.
- 'raison' étant une description de l'erreur.
- 'element' étant le document erroné.

Pour chaque collection, le script `detect_errors.py` récupère l'ensemble des documents de la collection, les parcourt et les vérifie un par un. Si une erreur est détectée, le document est ajouté à la collection `errors` et le script passe au document suivant. Une fois toutes les collections parcourues, le script notitifie de sa fin, si aucun autre affichage ne s'est produit précédemment, cela signifie que le script s'est terminé sans erreur.

cf. `detect_errors.py`

Correction des erreurs et réintégration des données

Comme décrit dans la partie 2, les erreurs sont contenues dans la collection `errors` et doivent être corrigées manuellement par l'utilisation de la méthode `updateOne` de l'interface de MongoDB.

Une fois les erreurs corrigées, les données peuvent être réintégrées dans leur collection d'origine et supprimées de la collection `errors`.

Notre script `insert_errors.py` permet de réintégrer les données dans leur collection d'origine. Cependant, ce script ne gère pas la détéction des erreurs et intègrera toules les données de la collection `errors` dans leur collection d'origine sans distinction. Il est donc conseillé d'utiliser le script `detect_errors.py` à la suite de `insert_errors.py` pour vérifier que d'autres erreurs ne subsistent pas.

cf. `insert_errors.py`