Université de Bourgogne
UFR sciences et Techniques
Département IEM

Module : *SGD*

Rapport projet

Systèmes de gestion de données

Responsables :

— Nadine Cullot

Réalisé par :

— JOULAIN Matthieu
— TROU Yann

Année : 2021-2022

Table des matières

1 Introduction

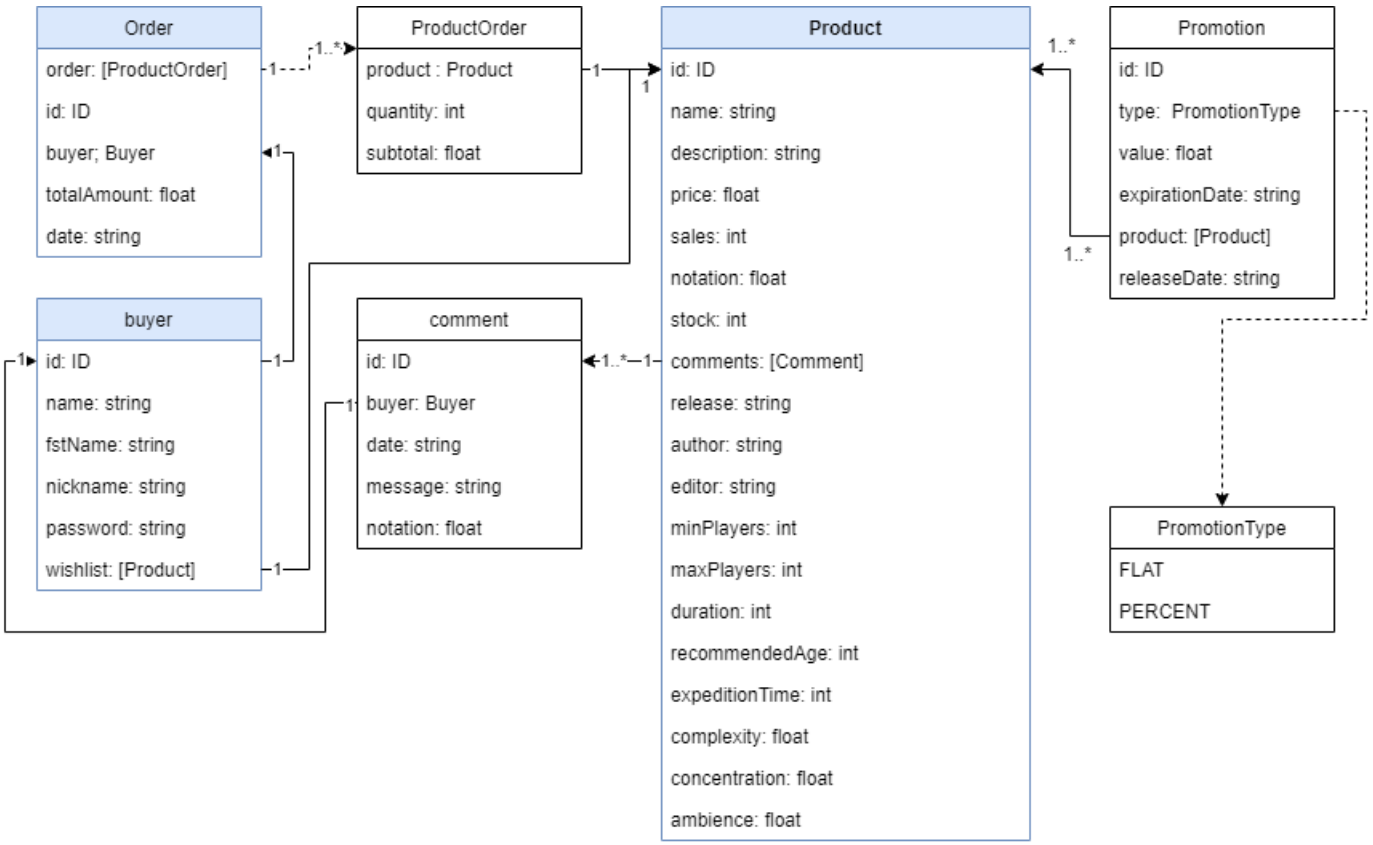
Ce projet et ce rapport présentent la conception et l'implémentation d'une base de données MongoDB pour le développement d'un site de vente en ligne de jeux de sociétés.

2 Conception

Pour modéliser le site, on doit représenter plusieurs éléments distincts :
— des produits, qui seront les jeux
— des acheteurs représentant les utilisateurs du site ayant un compte
— des commentaires écrits par les acheteurs concernant les produits
— des promotions s'appliquant sur les produits
— des factures indiquant les achats de produits réalisés par les acheteurs.

2.1 UML

On représente ces éléments et leurs relations dans le diagramme de classes suivant :



On va stocker ces objets dans 3 tables :
— Products
— Orders
— Buyers
Ces trois tables vont contenir les objets de la façon suivante :
Buyers ne va contenir que les éléments buyers. les produits de la wishlist d'un utilisateur seront des références vers Product car stocker les éléments Product dupliquerait trop de données.

Product va contenir les éléments Product mais aussi les éléments Comment et Promotion qui seront imbriqués dans le Product car les deux éléments sont liés à un unique élément Product.

Enfin, Orders va contenir les éléments Orders, avec les éléments ProductOrder qui seront imbriqués. Ces éléments vont avoir une référence vers un élément Product, et les éléments Order auront une référence vers un élément Buyer. Il n'y aura pas d'imbrication.

2.2 JSON

Ci dessous, un exemple de fichier contenu dans la table Product

```
{
    "_id": "1",
    "name": "produit de test",
    "description": "un produit de test, rien de plus",
    "price": 100.0,
    "category": "test",
    "sales": 1,
    "release": "01/01/2020",
    "notation": 0.0,
    "stock": 99,
    "promotions": [
        {
            "uuid": ObjectID("625d7c32720c8eab3cc6b589")
            "type": "FLAT",
            "value": 10,
            "startDate": "2021-01-01T00:00:00.000+00:00",
            "endDate": "2022-10-01T00:00:00.000+00:00"
        }
    ],
    "comments": [
        {
            "buyerID": ObjectID("625c18d40ad854de821d0401"),
            "date": "2022-04-18T16:22:48.86769",
            "message": "Commentaire de test",
            "notation": 4.5
        }
    ],
    "seller": "IAN",
    "author": "IAN",
    "editor": "IAN",
    "minPlayers": 1,
    "maxPlayers": 4,
    "duration": 30,
    "recommendedAge": 14,
    "expeditionTime": 24,
    "complexity": 2.0,
    "concentration": 3.0,
    "ambience": 4.0
}
```

Ci-dessous, un exemple de document contenu dans la table Orders

```
{
    "_id": ObjectID("625d85e20330de5febc54ad0"),
    "order": [
        {
```

```
            "idProduct": "1",
            "quantity": 1,
            "subtotal": 100.0
        }
    ],
    "buyerID": ObjectID("625c18d40ad854de821d0401"),
    "date": "2022-04-18T17:38:10.216+00:00",
    "totalAmount": 100.0
}
```

Ci-dessous, un exemple de fichier contenu dans la table Orders

```
{
    "_id": ObjectID("625c18d40ad854de821d0401"),
    "name": "Yann",
    "fstName": "Trou",
    "nickname": "IanZer",
    "password": "abc123",
    "wishlist": [
        ObjectID("1")
    ]
}
```

3 Implémentation

ATTENTION : les scripts fournis pour le projet ont étés implémentés avec python **3.9** et les bibliothèques **PyMongo[srv]**, **Tkinter** et **matplotlib**. La base de données est implémentée dans le cloud grâce au service gratuit fourni par mongoDB. Cela permet de mettre en place une base de données de manière simple et gratuite. Nous avons choisi cette option plutôt que le serveur Mongo de la fac car il est bien plus facile d'accès et ne nécessite pas de VPN pour pouvoir fonctionner. Nous avons également choisi de tout implémenter en python directement, car certaines fonctions nécessitent d'enchainer plusieurs appels afin que l'état de la base reste valide, et cela nous a semblé plus intéressant. La partie pour se connecter à MongoDB et accéder à la base est la même pour tous les scripts, elle n'est pas indiquée ici.

Pour les requêtes, on doit au minimum disposer des opérations CRUD (Create, Read, Update, Delete) sur tous les éléments, également les éléments imbriqués.

3.1 requêtes CREATE

```python
# créer un acheteur
buyer = {
        "name": input("name (string): "),
        "fstName": input("first name (string): "),
        "nickname": input("nickname (string): "),
        "password": input("password (string): "),
        "wishlist": []
    }

result = db.buyers.insert_one(buyer)
```

Ci-dessous, le script pour ajouter un commentaire à un produit. L'utilisateur qui ajoute le commentaire doit avoir acheté le produit mais également ne pas avoir déja posté de commentaire.

```python
# Ajout de commentaire a un produit avec vérification d'achat et de non duplication
# Check si l'utilisateur a acheté le produit
hasBought = db.orders.find_one(
    {"buyerID": ObjectId(buyerID)},
    {"order": {
        "$elemMatch": {
            "idProduct": ObjectId(productID)
        }
    }}
)
if hasBought is None:
    print("Erreur, l'utilisateur n'a pas acheté le produit")
    exit(1)

# Check si l'utilisateur a déjà commenté le produit
hasCommented = db.products.find_one(
    {
        "_id": ObjectId(productID),
        "comments.buyerID": ObjectId(buyerID)
    },
    {"comments": 1}
)

if hasCommented is not None:
    print("Erreur, l'utilisateur a déjà commenté sur le produit")
    exit(1)

comment = {
    "buyerID": ObjectId(buyerID),
    "date": datetime.now().isoformat(),
    "message": input("Message: "),
    "notation": int(input("Note (sur 5): "))
}

# mise a jour de la note du produit
pipeline = [
    {"$unwind": "$comments"},
    {"$group": {
        "_id": 0,
        "total": {"$sum": "$comments.notation"},
        "count": {"$count": {}}
    }}
]
notation = list(db.products.aggregate(pipeline))[0]
notation["count"] = notation["count"] + 1
notation["total"] = notation["total"] + comment["notation"]
productNotation = notation["total"] / notation["count"]

result = db.products.update_one(
    {"_id": ObjectId(productID)},
    {
        "$push": {"comments": comment},
        "$set": {"notation": productNotation}
    }
)
```

Nous avons également utilisé le framework d'agrégation pour faire certains appels spécifiques. Notamment, la gestion des commentaires qui sont imbriqués dans les produits mais aussi le script **DailySell** permettant d'obtenir le nombre de ventes journalières a l'échelle du site.

Ci dessous, une partie du script permettant la publication d'un commentaire avec la mise a jour de la note du produit.

```
comment = {
    "buyerID": ObjectId(buyerID),
    "date": datetime.now().isoformat(),
    "message": input("Message: "),
    "notation": int(input("Note (sur 5): "))
}

# mise a jour de la note du produit
pipeline = [
    {"$unwind": "$comments"},
    {"$group": {
        "_id": 0,
        "total": {"$sum": "$comments.notation"},
        "count": {"$count": {}}
    }}
]
notation = list(db.products.aggregate(pipeline))[0]
notation["count"] = notation["count"] + 1
notation["total"] = notation["total"] + comment["notation"]
productNotation = notation["total"] / notation["count"]

result = db.products.update_one(
    {"_id": ObjectId(productID)},
    {
        "$push": {"comments": comment},
        "$set": {"notation": productNotation}
    }
)
```

Ci-dessous, une partie du script **DailySell** permettant de donner le nombre de ventes avant, après ou le jour précis par rapport à une date fournie par l'utilisateur.

```
#si on veut une date avant / après
if choice != "$eq" :
    result = db.orders.aggregate([
        {"$match": {"date": {choice: date}}},
        {"$unwind" : "$order"},
        {"$group": {
            "_id" :"$order.idProduct",
            "total": {
                "$sum": "$order.quantity"}
        }},
        {"$project": {
            "_id": 1,
            "total": 1,
            "date": 1}
        },
        {"$sort" : {"date" : -1}}])
#si on veut une date le jour même
else :
    date = date.replace(date.year,date.month,date.day,0,0,0,0)
    datef = date.replace(date.year,date.month,date.day+1,0,0,0,0)
    result = db.orders.aggregate([
        {"$match": {"date" : {"$gte": date, "$lt": datef}}},
        {"$unwind" : "$order"},
```

```
        {"$group": {
            "_id" :"$order.idProduct",
            "total": {
                "$sum": "$order.quantity"
            }
        }},
        {"$project": {"_id": 1, "total" : 1, "date": 1}},
        {"$sort" : {"date" : -1}}])
```

3.2 MapReduce

Nous n'avons pas pu tester la requête Map Reduce car l'API étant dépréciée dans les versions de MongoDB, la fonctionnalité a été désactivée rendant l'exécution de ce genre de requêtes impossible.

3.3 Graphiques depuis mongoDB

En utilisant la bibliothèque **matplotlib**, on peut utiliser python et mongoDB afin de présenter des données sous forme de graphiques. Par exemple, un script que nous avons réalisé montre l'évolution des notes des commentaires d'un produit.

```
pipeline = [
    {"$match": {"_id": ObjectId("625c23fc781677e6f2531ce5")}},
    {"$unwind": "$comments"},
    {"$replaceRoot": {"newRoot": "$comments"}},
    {"$project": {
        "month": {"$month": {"$dateFromString": {"dateString": "$date"}}},
        "day": {"$dayOfMonth": {"$dateFromString": {"dateString": "$date"}}},
        "notation": 1
    }},
]

#Pour chaque jour du mois, on note l'évolution des notes des avis (moyenné) et le nombre de nouveaux av
comments = db.products.aggregate(pipeline)
comments = list(comments)

notes = [0] * 31
nbCommentaires = [0] * 31
for comment in comments:
    notes[comment["day"]] += comment["notation"]
    nbCommentaires[comment["day"]] += 1

#moyennage des jours
for i in range(0, 31):
    if nbCommentaires[i] != 0:
        notes[i] = (notes[i] / nbCommentaires[i])

#https://matplotlib.org/3.5.0/gallery/subplots_axes_and_figures/two_scales.html

plt.bar(range(0, 31), notes, color="r", label="note moyenne")
plt.plot(nbCommentaires, color="g", label="Nombre de commentaires")
plt.ylabel("Notation")
plt.xlabel("Jour")
plt.legend()
plt.show()
```

On peut également utiliser MongoDB dans une application standard. Nous avons fait une adaptation du script de création d'utilisateur afin de lui donner une interface plus familière que de la ligne de commande. Dans ce cas, on a déplacé l'appel à MongoDB dans une fonction avec l'utilisation de paramètres externes.

```
def process () :
    buyer = {
        "name": name.get(),
        "fstName": fstname.get(),
        "nickname": nickname.get(),
        "password": password.get(),
        "wishlist": []
    }
    result = db.buyers.insert_one(buyer)
    print("[INFO] Acknowledged: " + str(result.acknowledged))
    print("[INFO] _id: " + str(result.inserted_id))
    showinfo("Réponse de la DB", "Acknowledged: " + str(result.acknowledged))



window = Tk()
window.title("Créer un compte")
window.geometry("300x100")

name = StringVar()
name.set("nom")
textinput1 = Entry(window, textvariable=name, width=50)
textinput1.pack()
```

4 Conclusion

Ce projet nous a montré la versatilité et la flexibilité de mongoDB, que ce soit pour des petits scripts, des applications, ou pour de la recheche sur des grands ensembles de données ainsi que la facilité d'utilisation avec le driver PyMongo