CECS 323

Section 6 / 7909

- Project 3 -
- Everything is Awesome -

Tasos Lilis

May 13th, 2024

- JSON Schema -

```
{
 $jsonSchema: {
  bsonType: 'object', title: 'User', properties: {
   _id: {bsonType: 'int'},
   name: {bsonType: 'string'},
   ownedSets: { bsonType: 'array',
    items: {
     bsonType: 'object',
     properties: {
      setName: {
       bsonType: 'string'
      },
      setNumber: {bsonType: 'string'},
      theme: {bsonType: 'string'},
      quantity: {bsonType: 'int', minimum: 1},
      acquiredDate: {
       bsonType: 'date'
      },
      inventoryId: {
       bsonType: 'int'
      }
     },
     required: [  'setName', 'setNumber', 'theme','quantity', 'acquiredDate',
'inventoryId']
    }
   }
  },
  required: [
   '_id',
   'name',
   'ownedSets'
  ]
 }
}
```

- User Document -


```
{"_id": 1, "name": "Neal", "ownedSets": [
  {
   "setName": "Luke Skywalker's X-Wing Fighter",
   "setNumber": "75301-1",
   "theme": "Star Wars",
   "quantity": 2,
   "acquiredDate": {
    "$date": "2023-09-23T00:00:00.000Z"
   },
   "inventoryId": 80073
  },
  {
   "setName": "NASA Apollo Saturn V",
   "setNumber": "92176-1",
   "theme": "LEGO Ideas and CUUSOO",
   "quantity": 1,
   "acquiredDate": {
    "$date": "2023-07-22T00:00:00.000Z"
   },
   "inventoryId": 78168
  },
  {
   "setName": "NASA Space Shuttle Discovery",
   "setNumber": "10283-1",
   "theme": "Icons",
   "quantity": 1,
   "acquiredDate": {
    "$date": "2024-05-07T00:00:00.000Z"
   },
   "inventoryId": 94718
  },
  {
   "setName": "NASA Apollo 11 Lunar Lander",
   "setNumber": "\"10266-1\"",
   "theme": "Creator Expert",
   "quantity": 1,
   "acquiredDate": {
    "$date": "2024-04-01T00:00:00.000Z"
   },
   "inventoryId": 21862
  }
 ]
}
```

- Schema Justification -

In our schema design, we went with a One-To-Few design implementation. We embedded the “ownedSets” collection within the “users” collection while using the inventory's identification number to identify which specific inventories a user owns. This is able to happen due to the fact that the information stored in “ownedSets” is quite small only requiring, “aquiredDate”, “inventoryID”, and “quantity”. This is well within MongoDB’s max limit of 16 MB even if an individual owned every inventory. Due to that fact, we have the opportunity to denormalize and add three additional sets of data to help a user write a query and avoid an unnecessary join. This includes adding; “setName”, “setNumber”, and “theme”. Adding these extra fields can help users with getting important information about an owned set from a user without overextending. Given that those fields will very rarely experience changes, it would still be safe to add them.

Given that the average “setName” in lego contains about 21 characters (21 bytes), the average “setNumber” in lego contains about 7 characters (7 bytes), the average “theme” in lego contains about 12 characters (12 bytes), “quantity” holds an integer of 4 bytes, “acquiredDate” holds 8 bytes, and “inventoryID” holds another integer of 4 bytes, there is a total average of 56 bytes in our “ownedSets” collection. If an individual were to possess at least one of each of the 38,559 inventories, they will only hold an amount of 2,159,304 bytes or 2.159304 MB. Which is still within MongoDB’s max limit of 16 MB. In real life, the most sets owned by an individual is held by Miloš Křeček, with a total of 6,005 sets owned. If we assume that they are all unique. Miloš will only take up 0.33628 MB worth of data from his owned sets. Overall denormalizing our design is very much possible and is a benefit overall.

- Python Code -

```
from pymongo import MongoClient


def connect():
    connection_string = "mongodb://localhost:27017"
    client = MongoClient(connection_string)
    return client['project3']


if __name__ == "__main__":
    db = connect()

    print("Project 3:")
    problem = input("Would you like to do problem 1 or 2: ")

    if problem == '1':
        user_name = input("Enter the name of the user: ")

        pipeline = [
            {"$match": {"name": user_name}},
            {"$unwind": "$ownedSets"},
            {"$lookup": {"from": "inventories", "localField": "ownedSets.inventoryId",
"foreignField": "_id", "as": "inventory"}},
            {"$unwind": "$inventory"},
            {"$unwind": "$inventory.parts"},
            {"$group": {
                "_id": {"partName": "$inventory.parts.part.partName", "colorName":
"$inventory.parts.color.colorName"},
                "totalQuantity": {"$sum": "$inventory.parts.quantity"}
            }},
            {"$project": {
                "_id": 0,
                "partName": "$_id.partName",
                "colorName": "$_id.colorName",
                "totalQuantity": 1
            }},
```

```
        {"$sort": {"partName": 1, "colorName": 1}}
    ]

    results = db["users"].aggregate(pipeline)

    for result in results:
        print(
            f"Total Quantity: {result['totalQuantity']}, "
            f"Part Name: {result['partName']}, "
            f"Color Name: {result['colorName']}"
        )

elif problem == '2':
    set_number = input("Enter the set number: ")
    version_number = int(input("Enter the version number: "))

    pipeline = [
        {"$match": {"setNumber": set_number, "version": version_number}},
        {"$unwind": "$parts"},
        {"$match": {"parts.isSpare": False}},
        {"$project": {
            "_id": 0,
            "quantity": "$parts.quantity",
            "partName": "$parts.part.partName",
            "colorName": "$parts.color.colorName"
        }},
        {"$sort": {"partName": 1, "colorName": 1}}
    ]

    results = db["inventories"].aggregate(pipeline)

    for result in list(results):
        print(
            f"Total Quantity: {result['quantity']},"
            f" Part Name: {result['partName']}, "
            f"Color Name: {result['colorName']}")

else:
    print("Invalid selection. Please enter 1 or 2.")
```