LABORATORIUM NIERELACYJNE BAZY DANYCH

Data wykonania ćwiczenia:	26.02.2023
Rok studiów:	3
Semestr:	6
Grupa studencka:	2
Grupa laboratoryjna:	2B

Ćwiczenie nr.	2

Temat: Modelowanie danych w MongoDB: Tworzenie Struktury Kolekcji (Zadanie nr.2)

Osoby wykonujące ćwiczenia:

1. Igor Gawłowicz
2. Andrzej Macura
3. Patryk Ihas

Katedra Informatyki i Automatyki

Opis zadania

System Oceny Studentów:

Kolekcja "courses" z zagnieżdżonymi ocenami dla każdego studenta.

Aby stworzyć kolekcje odpowiedzialną za system oceny studentów musieliśmy rozważyć strukturę naszej tabeli, więc zaczęliśmy od napisania przykładowej struktury rekordu

```
[
  {
    "student_id": 1,
    "student_name": "Anna",
    "student_surname": "Nowak",
    "subjects": [
      {
        "subject": "math",
        "grade": 4,
        "final": true
      },
      {
        "subject": "history",
        "grade": 5,
        "final": true
      }
    ]
  }
]
```

Postanowiliśmy że potrzebujemy osobnego rekordu dla każdego studenta, gdzie każdy student będzie miał swoją tabelę ocen dla każdego przedmiotu, w ten sposób doszliśmy do zdefiniowania kolekcji w następujący sposób

```
db.createCollection("courses", {
   validator: {
      $jsonSchema: {
         bsonType: "object",
         required: ["student_id", "student_name", "student_surname", "subjects"],
         properties: {
            student_id: {
               bsonType: "int",
               description: "ID of the student"
            },
            student_name: {
               bsonType: "string",
               description: "Name of the student"
            },
            student_surname: {
               bsonType: "string",
```

```
                description: "Surname of the student"
            },
            subjects: {
                bsonType: "array",
                description: "List of subjects",
                items: {
                    bsonType: "object",
                    required: ["subject", "grade", "final"],
                    properties: {
                        subject: {
                            bsonType: "string",
                            description: "Name of the subject"
                        },
                        grade: {
                            bsonType: ["int", "double"],
                            description: "Grade of the student for the subject"
                        },
                        final: {
                            bsonType: "bool",
                            description: "Whether the grade is final or not"
                        }
                    }
                }
            }
        }
    }
});
```

Operacje CRUD

Przykładowe dane

Do celów testowych przygotowaliśmy skrypt generujący duże zestawy losowych danych który możemy następnie uruchomić przez mongo shell, tworząc automatycznie naszą tabelę i zapełniając ją podaną ilością rekordów, z w pełni losowymi sztucznymi danymi.

```
out.write("db.courses.insertMany([\n")
for i in range(100):
    out.write("{\n")
    out.write(f'"student_id": {i},\n')
    out.write(f'"student_name": "{fake.first_name()}",\n')
    out.write(f'"student_surname": "{fake.last_name()}",\n')
    out.write(f'"subjects": [\n')
    for sub in subjects:
        out.write('{\n')
        out.write(f'"subject": "{sub}",\n')
        out.write(f'"grade": "{random.randint(2,5)}",\n')
        out.write(f'"final": "{random.choice(states)}"\n')
        out.write('},\n')
    out.write('],\n')
```

```
    out.write("},\n")
out.write("])\n")
```

```
db.courses.insertMany([
{
"student_id": 0,
"student_name": "Shannon",
"student_surname": "Dennis",
"subjects": [
{
"subject": "Mathematics",
"grade": "4",
"final": "false"
},
{
"subject": "English Language",
"grade": "5",
"final": "true"
},
{
"subject": "History",
"grade": "2",
"final": "false"
},
{
"subject": "Science",
"grade": "2",
"final": "true"
},
{
"subject": "Computer Science",
"grade": "3",
"final": "false"
},
{
"subject": "Physical Education",
"grade": "3",
"final": "true"
},
],
},
...itd
```

Pytania kontrolne

1. Opisz, dlaczego zastosowanie zagnieżdżonych dokumentów może być korzystne w modelowaniu danych w MongoDB. Podaj przykłady sytuacji, w których to rozwiązanie jest preferowane.

Zastosowanie zagnieżdżonych dokumentów w MongoDB może być korzystne, ponieważ pozwala na przechowywanie związanych ze sobą danych w jednym dokumencie. To może ułatwić modelowanie danych,

zwłaszcza w przypadku danych, które są często używane razem lub są silnie powiązane. Przykłady sytuacji, w których zagnieżdżone dokumenty są preferowane:

- Relacje jeden do wielu, gdzie jedna encja zawiera wiele powiązanych danych (np. kursy w jednym zagnieżdżonym dokumencie, a oceny studentów w drugim).
- Dane, które nie są używane niezależnie od siebie, a ich powiązanie jest istotne dla aplikacji.
- Dane, które są zawsze pobierane razem, co może poprawić wydajność operacji odczytu.

2. W jaki sposób definiuje się strukturę kolekcji w MongoDB, uwzględniając zagnieżdżone dokumenty i tablice? Jakie elementy są wymagane w schemacie kolekcji?

Strukturę kolekcji w MongoDB definiuje się za pomocą dokumentu JSON, który określa schemat danych. Elementy wymagane w schemacie kolekcji obejmują:

- Nazwę pola i jego typ danych.
- Ewentualne ograniczenia, takie jak wymagane pola, unikalność wartości, zakresy wartości, etc.
- Jeśli kolekcja zawiera zagnieżdżone dokumenty lub tablice, należy określić ich strukturę w ramach właściwości obiektu głównego.

3. Dlaczego warto stosować indeksowanie w MongoDB? Jakie pola warto zindeksować, aby poprawić wydajność zapytań w przypadku zagnieżdżonych dokumentów?

Indeksowanie w MongoDB jest wartościowe, ponieważ poprawia wydajność zapytań poprzez przyspieszenie procesu wyszukiwania danych. Pola, które warto zindeksować w przypadku zagnieżdżonych dokumentów, to te, które są często używane w zapytaniach wyszukiwania lub sortowania. Indeksowanie pól, które służą jako klucze do łączenia kolekcji (np. pole student_id w przypadku ocen studentów w kolekcji kursów), może znacząco poprawić wydajność zapytań.

4. Wytłumacz, jakie są główne różnice między zagnieżdżonymi dokumentami a odwołaniami do innych kolekcji. Kiedy warto użyć jednego podejścia zamiast drugiego?

Główne różnice między zagnieżdżonymi dokumentami a odwołaniami do innych kolekcji to:

- Zagnieżdżone dokumenty przechowują dane w jednym dokumencie, podczas gdy odwołania do innych kolekcji przechowują dane w oddzielnych kolekcjach.
- Zagnieżdżone dokumenty są bardziej wydajne do odczytu, ponieważ wymagają jednego zapytania do bazy danych, podczas gdy odwołania do innych kolekcji mogą wymagać wielu zapytań.
- Odwołania do innych kolekcji mogą ułatwić aktualizację danych w przypadku, gdy te dane są używane w wielu miejscach.
- Warto użyć jednego podejścia zamiast drugiego zależy od specyfiki aplikacji i struktury danych, oraz wymagań dotyczących wydajności i spójności danych.

5. Przedstaw przykład struktury kolekcji dla systemu oceny studentów, gdzie kursy zawierają zagnieżdżone oceny dla każdego studenta. Opisz kluczowe elementy tej struktury.

```
    db.createCollection(“courses”,{
validator:{
$jsonSchema: {
bsonType: “object”,
required[“student_id”,”student_name”,”student_surname”,”subjects”],
```

```
properties: {
student_id: {
                bsonType: “int”,
                description: ”Id ucznia”
                },
            student_name: {
                bsonType: “string”,
description: “Imię studenta”
},
student_surname: {
    bsonType: “string”,
    description: “Nazwisko studenta”,
},
subjects: {
    bsonType: “array”,
    description: “Przedmioty szkolne”
    items: {
        bsonType: “object”,
        required: [“subject,”grade”,”final”],
        properties: {
            subject: {
                bsonType: “string”,
                description: “Nazwa przedmiotu”
                },
            grade: {
                bsonType: “int”,
                description: “Ocena ucznia”
                },
            final: {
                bsonType: “bool”
                description: “Końcowa ocena”
                },
                }
            }
        }
    }
}
}
});
```

6. Jakie są korzyści z uwzględniania podpowiedzi (hintów) w MongoDB podczas implementacji struktury kolekcji? Kiedy warto je stosować?

Korzyści z uwzględniania podpowiedzi (hintów) w MongoDB podczas implementacji struktury kolekcji obejmują poprawę wydajności zapytań poprzez kontrolę wyboru indeksu lub planu zapytania. Warto stosować podpowiedzi w przypadku, gdy wiesz, że określony indeks jest bardziej odpowiedni dla zapytania niż domyślnie wybrany przez silnik bazodanowy MongoDB.

7. W jaki sposób zaplanować strukturę kolekcji, aby umożliwić optymalne zapytania dotyczące konkretnej kategorii danych w zagnieżdżonych tablicach?

Aby umożliwić optymalne zapytania dotyczące konkretnej kategorii danych w zagnieżdżonych tablicach, należy odpowiednio zindeksować pola używane w zapytaniach wyszukiwania lub sortowania. Ważne jest również, aby rozważyć strukturę zagnieżdżonych dokumentów tak, aby umożliwić efektywne zapytania dotyczące danych wewnątrz tablic.

8. Opisz, jak można zastosować operacje CRUD (Create, Read, Update, Delete) w kontekście zagnieżdżonych dokumentów w MongoDB. Przedstaw przykłady dla każdej operacji.

Operacje CRUD (Create, Read, Update, Delete) w kontekście zagnieżdżonych dokumentów w MongoDB można stosować w taki sam sposób jak dla dokumentów na głównym poziomie. Przykłady:

- Create: db.collection.insertOne() lub db.collection.insertMany() dla dodawania nowych dokumentów.
- Read: db.collection.find() do odczytu dokumentów, db.collection.findOne() do odczytu pojedynczego dokumentu.
- Update: db.collection.updateOne() lub db.collection.updateMany() do aktualizacji dokumentów.
- Delete: db.collection.deleteOne() lub db.collection.deleteMany() do usuwania dokumentów.

9. Dlaczego db.createCollection() w przykładzie zawiera sekcję "validator", a co to za sekcja? Jakie jest jej znaczenie w kontekście modelowania danych w MongoDB?

W przypadku db.createCollection() sekcja "validator" zawiera definicję walidatora schematu dla danej kolekcji. Jest to mechanizm służący do sprawdzania, czy dokumenty dodawane do kolekcji spełniają określone kryteria co do struktury i zawartości danych. Walidator określa, jakie pola są wymagane, jakiego typu powinny być dane oraz inne reguły, które muszą być spełnione.

10. Co to jest Aggregation Pipeline w MongoDB, i w jaki sposób można go wykorzystać do przetwarzania danych zagnieżdżonych w kolekcji? Przedstaw prosty przykład zastosowania Aggregation Pipeline.

Aggregation Pipeline w MongoDB to mechanizm pozwalający na przetwarzanie danych w kolekcji w celu wykonywania złożonych operacji agregujących, takich jak grupowanie, sortowanie, łączenie danych i wiele innych. Może być wykorzystany do przetwarzania danych zagnieżdżonych w kolekcji poprzez operacje, takie jak $unwind, $lookup, $group, $project itp. Przykład zastosowania Aggregation Pipeline:

```
db.collection.aggregate([
  { $unwind: "$nested_array" }, // Rozwinięcie zagnieżdżonej tablicy
  { $group: { _id: "$nested_array.category", total: { $sum: 1 } } } // Grupowanie
i obliczanie sumy
])
```