Тема 13. Модель данных в MongoDB.

Цель занятия:

Получить представление о моделях данных в MongoDB.

Учебные вопросы:

1. Типы моделей данных в MongoDB.
2. Сравнение встроенной и нормализованной моделей.
3. Основные принципы проектирования данных в MongoDB.
4. Валидирование схемы в MongoDB.

1. Типы моделей данных в MongoDB.

Встроенная модель данных (Embedded Data Model) в MongoDB.

Встроенная модель данных (Embedded Data Model) в MongoDB предполагает хранение связанных данных в одном документе путем вложения одного документа внутрь другого.

Это позволяет избежать разбиения данных по разным коллекциям и объединения их с помощью сложных запросов.

Особенности хранения вложенных документов:

- Вложенные документы хранятся внутри основного документа, что делает данные более компактными и самодостаточными.
- MongoDB позволяет вложенным документам содержать другие вложенные документы или массивы, что делает структуру данных гибкой.
- Размер одного документа в MongoDB ограничен 16 МБ, что необходимо учитывать при использовании данной модели.

Примеры использования. Модель пользователей и их адресов: Если каждому пользователю можно задать несколько адресов, эти данные можно хранить в одном документе с вложенными адресами:

```
{
  "user_id": 1,
  "name": "John Doe",
  "email": "john.doe@example.com",
  "addresses": [
    { "street": "123 Main St", "city": "New York", "zip": "10001" },
    { "street": "456 Oak St", "city": "Los Angeles", "zip": "90001" }
  ]
}
```

Модель заказов и товаров: При хранении информации о заказе можно встроить детали заказанных товаров прямо в документ заказа:

```
{
  "order_id": 101,
  "order_date": "2024-10-01",
  "customer": { "customer_id": 1, "name": "John Doe" },
  "items": [
    { "item_id": 1, "name": "Laptop", "price": 1200, "quantity": 1 },
    { "item_id": 2, "name": "Mouse", "price": 50, "quantity": 2 }
  ]
}
```

Преимущества встроенной модели данных:

- Минимизация количества запросов: Все данные по связанным объектам (например, пользователю и его заказам) находятся в одном документе, что снижает необходимость делать дополнительные запросы для получения связанных данных.
- Более быстрая работа с данными: MongoDB может быстро получить весь документ целиком, что ускоряет выполнение операций, особенно при частом чтении.
- Простота управления данными: Вложенная модель упрощает структуру данных, особенно для простых одноуровневых связей (например, "один к одному" или "один ко многим").
- Отсутствие сложных операций объединения (JOIN): Нет необходимости использовать сложные JOIN-запросы, как в реляционных базах данных, что снижает нагрузку на сервер и ускоряет выполнение запросов.

Недостатки встроенной модели данных:

- Ограничение на размер документа: В MongoDB существует ограничение в 16 МБ на размер одного документа. Если данные становятся слишком большими, встроенная модель становится непрактичной.
- Дублирование данных: При изменении вложенных данных в нескольких документах может возникнуть необходимость обновлять дублирующийся данные в разных местах. Это увеличивает риск несогласованности данных.
- Сложности с изменением структуры: Вложенная модель усложняет операции обновления и удаления вложенных данных, особенно если вложенные документы изменяются часто.
- Снижение гибкости при сложных отношениях: Если объекты имеют сложные связи, например "многие ко многим", встроенная модель становится неэффективной. Для таких случаев лучше использовать нормализованную модель с ссылками между документами.

Заключение:

Встроенная модель данных в MongoDB удобна для простых иерархических структур данных, которые часто читаются целиком. Она упрощает работу с данными и ускоряет запросы, но имеет свои ограничения при работе с большими объемами данных или сложными связями.

Нормализованная модель данных (Normalized Data Model) в MongoDB.

Нормализованная модель данных в MongoDB предполагает разделение данных по разным коллекциям с использованием ссылок (референсов) между документами.

Это похоже на подход реляционных баз данных, где данные хранятся в отдельных таблицах, связанных через внешние ключи.

Вместо вложения данных одного объекта в другой, используется схема ссылок, при которой один документ хранит идентификатор другого документа для его связи.

Использование ссылок между документами (схема на основе ссылок):

В нормализованной модели вместо вложения связанных данных используются ссылки (_id), с помощью которых документы из разных коллекций могут быть связаны между собой.

Например, если есть коллекции для пользователей и заказов, вместо хранения информации о заказах внутри документа пользователя, каждый заказ может храниться отдельно, ссылаясь на идентификатор пользователя.

Пример использования. Коллекция пользователей (users):

```
{
  "_id": 1,
  "name": "John Doe",
  "email": "john.doe@example.com",
  "age": 28
}
```

Коллекция заказов (orders):

```
{
  "order_id": 101,
  "user_id": 1,  // Ссылка на пользователя
  "order_date": "2024-10-01",
  "items": [
    { "item_id": 1, "name": "Laptop", "price": 1200, "quantity": 1 },
    { "item_id": 2, "name": "Mouse", "price": 50, "quantity": 2 }
  ]
}
```

Для получения всех заказов пользователя, можно выполнить агрегированный запрос с использованием оператора **$lookup**, который соединяет коллекции по полю user_id.

```
db.orders.aggregate([
   {
     $lookup:
       {
         from: "users",
         localField: "user_id",
         foreignField: "_id",
         as: "user_info"
       }
   }
])
```

Преимущества нормализованной модели данных:

- Экономия памяти: Нормализованная модель позволяет избежать дублирования данных, так как общая информация (например, данные о пользователе) хранится в одном месте и на неё ссылаются другие коллекции.
- Гибкость в управлении данными: Изменение данных в одной коллекции автоматически распространяется на все связанные данные, так как ссылки обновлять не нужно (например, изменение данных пользователя не требует обновления документов заказов).
- Удобство при сложных отношениях: Нормализованная модель идеально подходит для сложных связей между данными, таких как "многие ко многим", "один ко многим" и т.д. С помощью ссылок можно легко моделировать такие отношения.
- Контроль над размерами документов: Поскольку связанные данные хранятся в отдельных коллекциях, общий размер документа уменьшается, и проблема с ограничением на размер документа в 16 МБ (в MongoDB) исчезает.

Недостатки нормализованной модели данных:

- Необходимость дополнительных запросов: Для получения полных данных приходится делать несколько запросов или использовать объединение коллекций с помощью $lookup. Это увеличивает время выполнения операций по сравнению с встроенной моделью данных, где все данные хранятся в одном документе.
- Сложность запросов: Запросы с использованием референсов могут быть сложнее в написании и выполнении по сравнению с запросами, работающими с встроенными данными. Например, агрегированные запросы с $lookup требуют больше ресурсов.
- Меньшая производительность при чтении: Чтение данных может быть медленнее, так как MongoDB должен загружать данные из нескольких коллекций и выполнять объединения. Это может стать проблемой в системах с высоким количеством операций чтения.
- Потенциальные проблемы с консистентностью: Если данные между коллекциями не синхронизированы, может возникнуть проблема консистентности, когда изменения в одной коллекции не отражаются корректно в другой.

Заключение:

Нормализованная модель данных в MongoDB используется для сложных структур данных и в тех случаях, когда важно избегать дублирования информации.

Она позволяет эффективно работать с большими объемами данных, экономить память и создавать сложные взаимосвязи между данными.

Однако, её использование требует дополнительных операций при чтении данных и может быть менее производительным, чем встроенная модель.

Гибридный подход в MongoDB: Комбинирование встроенной и нормализованной моделей

Гибридный подход в MongoDB предполагает сочетание встроенной (Embedded) и нормализованной (Normalized) моделей данных. Это означает, что некоторые данные могут храниться как вложенные документы внутри основного документа (встроенная модель), в то время как другие данные могут храниться в отдельных коллекциях с использованием ссылок (нормализованная модель). Такой подход позволяет гибко адаптировать структуру данных под конкретные задачи, учитывая как производительность, так и удобство работы с данными.

Когда использовать гибридную модель?

Гибридный подход удобен в ситуациях, когда часть данных является тесно связанной и часто используется вместе, а другая часть данных может быть более обособленной или редко изменяемой. Использование этого подхода позволяет комбинировать преимущества встроенной и нормализованной моделей для повышения производительности, гибкости и удобства работы с данными.

Примеры ситуаций для использования гибридной модели:

E-commerce: заказы и товары

Встроенная модель: Можно использовать встроенные данные для информации о товарах в заказе. Например, данные о товаре (название, цена) можно встроить в документ заказа, так как они редко изменяются после оформления заказа и всегда используются вместе с информацией о заказе.

Нормализованная модель: Ссылки могут использоваться для хранения информации о пользователе, который оформил заказ. Ссылка на пользователя позволяет управлять информацией о нём в отдельной коллекции, так как эти данные могут часто изменяться (например, адрес доставки или контактная информация).

Пример:

```
{
  "order_id": 101,
  "user_id": 1,  // Ссылка на пользователя
  "order_date": "2024-10-01",
  "items": [
    { "item_id": 1, "name": "Laptop", "price": 1200, "quantity": 1 },
    { "item_id": 2, "name": "Mouse", "price": 50, "quantity": 2 }
  ]
}
```

Социальная сеть: пользователи и публикации

Встроенная модель: Можно встроить комментарии к посту прямо в документ поста, так как комментарии тесно связаны с контентом и редко используются отдельно.

Нормализованная модель: Пользователи и посты могут храниться в разных коллекциях, так как пользователи могут публиковать много постов, и при этом посты могут использоваться без привязки к пользователю при отображении на страницах с популярными записями.

```
{
  "post_id": 123,
  "user_id": 5,  // Ссылка на автора поста
  "content": "This is my first post!",
  "comments": [
    { "comment_id": 1, "user": "Jane", "text": "Great post!" },
    { "comment_id": 2, "user": "John", "text": "Thanks for sharing!"
  ]
}
```

Учебная платформа: студенты и курсы

```
{
  "student_id": 102,
  "name": "Alex Smith",
  "courses_enrolled": [
    { "course_id": 1, "progress": 80 },  // Ссылка на курс
    { "course_id": 2, "progress": 60 }
  ]
}
```

Заключение:

Гибридный подход в MongoDB предлагает эффективное сочетание встроенной и нормализованной моделей данных, позволяя адаптироваться к различным сценариям использования и требованиям к производительности. Он особенно полезен для приложений с разной степенью связанности данных, где некоторые данные могут храниться компактно и использоваться часто, а другие — обособленно и редко обновляться.

3. Основные принципы проектирования данных в MongoDB.

При проектировании базы данных в MongoDB важно учитывать особенности документно-ориентированной структуры данных, отличающейся от реляционных баз данных. Основные принципы проектирования MongoDB направлены на обеспечение гибкости, производительности и простоты управления данными.

1. Проектирование по доступу к данным.

В MongoDB дизайн базы данных начинается с анализа того, как данные будут использоваться приложением, а не с создания фиксированной схемы данных (как в реляционных БД). Это называется "проектирование на основе запросов".

Определите основные запросы, которые будут выполняться чаще всего.

Старайтесь проектировать структуру коллекций так, чтобы минимизировать количество запросов и соединений (особенно дорогостоящих операций $lookup).

Вложенные документы или массивы помогут вам запрашивать связанные данные быстрее.

Пример: Если данные пользователя и его заказы часто запрашиваются вместе, может быть полезно хранить заказы во вложенном массиве в документе пользователя.

2. Использование встроенных документов и массивов

В MongoDB поддерживаются вложенные документы и массивы, что позволяет гибко структурировать данные внутри одного документа.

Используйте встроенные документы для данных, которые имеют логическую связь и должны быть запрашиваемы вместе.

Используйте массивы для хранения связанных данных (например, список покупок, комментарии к постам).

Пример: Хранение адресов доставки пользователя как массива во встроенном документе:

```
{
  "_id": 1,
  "name": "John Doe",
  "addresses": [
    { "type": "home", "address": "123 Street" },
    { "type": "work", "address": "456 Avenue" }
  ]
}
```

3. Избегайте глубоких вложений

MongoDB позволяет создавать вложенные документы, но глубина вложенности не должна быть слишком большой. Рекомендуется ограничивать вложенность до 2-3 уровней.

Глубокая вложенность усложняет работу с данными, их обновление и удаление.

MongoDB ограничивает размер документа до 16 MB, что также ограничивает вложенность.

Совет: Если вложенные данные становятся слишком большими или изменяются независимо, возможно, стоит использовать ссылки на отдельные коллекции.

4. Баланс между встроенными и нормализованными структурами

Используйте встроенные документы для данных, которые запрашиваются вместе, и которые редко изменяются независимо.

Применяйте ссылки между коллекциями (нормализация), если данные часто меняются или используются в нескольких местах (например, товары в заказах).

Пример: Вложенные документы можно использовать для комментариев к посту, если они редко изменяются. А нормализованную структуру — для хранения информации о пользователе и его постах, если они часто редактируются.

5. Учет ограничений MongoDB

При проектировании базы данных учитывайте технические ограничения MongoDB:

Максимальный размер одного документа — 16 MB.

Ограничение на количество индексов в коллекции — 64.

Количество элементов в массиве — не более 1000 для агрегационных операций $unwind.

6. Использование индексов для ускорения запросов

Индексы важны для повышения производительности. MongoDB поддерживает создание индексов на полях для ускорения поиска.

Создавайте индексы на полях, по которым часто выполняются запросы (например, поля, используемые в фильтрах find).

Используйте составные индексы, если запросы выполняются по нескольким полям одновременно.

Определите необходимость уникальных индексов (например, на email пользователей).

7. Избегайте избыточности, когда это возможно

MongoDB не требует строгой нормализации данных, но все же следует избегать дублирования данных, которое может привести к увеличению объема данных и сложности их обновления.

Используйте ссылки (нормализованную структуру), если это значительно уменьшает избыточность.

Гибкость схемы данных

В MongoDB схема данных не является фиксированной, что позволяет сохранять разные структуры документов в одной коллекции. Однако важно следить за поддержанием хотя бы минимальной унификации структуры документов.

Определите ключевые обязательные поля, которые должны присутствовать во всех документах.

Используйте механизм валидации схемы для того, чтобы контролировать соответствие структуры документов определенным правилам.

4. Валидирование схемы в MongoDB.

В MongoDB схема документа не строго фиксирована, однако для обеспечения целостности и структурной правильности данных можно использовать механизмы валидации схемы.

Этот инструмент позволяет задавать правила для проверки данных, которые вставляются или обновляются в коллекциях.

Валидация схемы — это процесс проверки, соответствуют ли данные определённым условиям перед их сохранением в базе данных.

MongoDB позволяет создавать правила валидации на уровне коллекций, чтобы гарантировать соблюдение требований к структуре и значениям полей документов.

В MongoDB можно настроить валидацию схемы при создании коллекции или на существующей коллекции с использованием оператора validator. В правилах валидации используются операторы для сравнения, логические выражения и операторы типа $jsonSchema.

```
db.createCollection("users", {
    validator: {
        $jsonSchema: {
            bsonType: "object",
            required: ["name", "age"],
            properties: {
                name: {
                    bsonType: "string"
                },
                age: {
                    bsonType: "int"
                }
            }
        }
    }
})
```

это параметр, который указывает правила проверки (валидации) данных

специальный оператор в MongoDB, который позволяет задавать структуру и ограничения для данных, используя JSON-схему

это массив обязательных полей, которые должны присутствовать в каждом документе.

это объект, который описывает поля документа и их типы.

Поле name должно быть строкой (тип данных string).

Поле age должно быть целым числом (тип данных int).

В правилах валидации MongoDB можно использовать различные операторы:

- bsonType — задаёт тип данных (например, string, int, object, array).
- required — определяет обязательные поля документа.
- minimum / maximum — ограничения для числовых значений.
- pattern — проверка на соответствие регулярным выражениям.
- enum — задаёт список допустимых значений для поля

Уровни строгой валидации.

MongoDB предоставляет возможность настройки уровня строгости валидации:

- strict — отклоняет любые документы, которые не соответствуют указанным правилам валидации.
- moderate — проверяет только документы, которые содержат проверяемые поля (остальные документы сохраняются).

```
db.createCollection("products", {
   validator: {
      $jsonSchema: {
         bsonType: "object",
         properties: {
            price: {
               bsonType: "double",
               minimum: 0,
               description: "Price must be a positive number."
            }
         }
      }
   },
   validationLevel: "moderate"
})
```

Преимущества валидации схемы:

- Контроль целостности данных: Валидация помогает гарантировать, что все вставляемые и обновляемые данные соответствуют заданным требованиям.
- Упрощение работы с данными: Валидация позволяет избежать ошибок, связанных с некорректными данными, и облегчает работу с данными для разработчиков.
- Гибкость и настройка: MongoDB позволяет гибко настраивать валидацию в зависимости от требований приложения и постепенно улучшать валидацию на уже существующих данных.

Недостатки валидации схемы:

- Снижение производительности: При интенсивных операциях записи дополнительная проверка данных может увеличивать задержки.
- Сложность поддержки: В сложных системах может потребоваться детальная настройка валидации, что увеличивает сложность конфигурации.

Домашнее задание:

1. Повторить материал лекции.

Список литературы:

1. В. Ю. Кара-ушанов SQL — язык реляционных баз данных
2. А. Б. ГРАДУСОВ. Введение в технологию баз данных
3. А.Мотеев. Уроки MySQL

Материалы лекций:

https://github.com/ShViktor72/Education

Обратная связь:

colledge20education23@gmail.com