GO-08 05: Работа с NoSql (Mongo)

Описание:

Базы данных делятся на два основных типа: реляционные(SQL) и нереляционные(NoSQL). До этого момента вы работали только с SQL базами данных. В этой главе мы рассмотрим самую популярную NoSQL базу данных.
Mongo:
MongoDB реализует новый подход к построению баз данных, где нет таблиц, схем, запросов SQL, внешних ключей и многих других вещей, которые присущи объектно-реляционным базам данных.
В отличие от реляционных баз данных MongoDB предлагает документо-ориентированную модель данных, благодаря чему MongoDB работает быстрее, обладает лучшей масштабируемостью, ее легче использовать.
Всю историю разработки использовали только SQL базы данных, не задумываясь о том, подходит ли эта модель под их нужды. Но, даже учитывая все недостатки традиционных баз данных и достоинства MongoDB, важно понимать, что задачи бывают разные и методы их решения бывают разные. В какой-то ситуации MongoDB действительно улучшит производительность вашего приложения, например, если надо хранить сложные по структуре данные. В другой же ситуации лучше будет использовать традиционные реляционные базы данных. Кроме того, можно использовать смешенный подход: хранить один тип данных в MongoDB, а другой тип данных - в традиционных БД.
Когда лучше использовать Mongo:
Основной плюс Mongo это не жестка структура данных, хранение в json формате. Это позволяет нам не менять таблицу каждый раз, когда появилось новое поле в структуре. Вся работа с Mongo, это в чистом виде ORM. Легко соотносить ваши структуры данных из кода, в структуру json.
Так когда-же лучше использовать Mongo:

- Когда ваше приложение быстро развивается и структура часто меняться.
- Когда вам не нужно делать сложных запросов с join и group by.
- Когда у вас сложная вложенная структура данных

Перейдем к практике.
Работа с Mongo:
Одним из популярных стандартов обмена данными и их хранения является JSON (JavaScript Object Notation). JSON эффективно описывает сложные по структуре данные. Способ хранения данных в MongoDB в этом плане похож на JSON, хотя формально JSON не используется. Для хранения в MongoDB применяется формат, который называется BSON (БиСон) или сокращение от binary JSON.
BSON позволяет работать с данными быстрее: быстрее выполняется поиск и обработка. Хотя надо отметить, что BSON в отличие от хранения данных в формате JSON имеет небольшой недостаток: в целом данные в JSON-формате занимают меньше места, чем в формате BSON, с другой стороны, данный недостаток с лихвой окупается скоростью.

Основные команды в Mongo:

- find
- update
- insert

Примеры команд из консоли:

```
db.users.insertOne({"name": "Tom", "age": 28, languages: ["english", "spanish"]})
db.users.find()
db.users.update({name : "Tom"}, {name: "Tom", age : 25}, {upsert: true})
```

Команды в mongo и работа с ними, крайне обширная тема. Что бы описать ее всю, потребуется отдельный практикум. Поэтому со всеми командами, которые вам понадобятся вы сможете ознакомится в официальной документации.

MGO:

Для работы с mongo из golang кода мы будем использовать mgo. На данный момент это самая простая библиотека для работы с mongo.

Аналоги можно найти по ссылке(так же советую сохранить эту ссылку себе, там список все доступных библиотек для работы).

Рассмотрим основные способы взаимодействия:

Сохранение данных:

```
package main

import (
	"fmt"
	"github.com/globalsign/mgo/bson"
	"time"
	mongo "github.com/globalsign/mgo"
)

// создаеам структуру, которую будем хранить в базе данных
type User struct {
	//смотрим на тэги
	//специальный тэг bson отвечает за имя в mongo
	Name string `json:"name" bson:"name"`
	Age int `json:"age" bson:"age"`
	Documents struct{
		PassportNumber string `json:"passport_number" bson:"passport_number"`
		INN int `json:"inn" bson:"inn"`
	}
```

```
	CreateAt int64 `json:"create_at" bson:"create_at"`
}

func main(){
	u := User{
		Name: "Наземнов Глеб Андреевич",
		Age: 22,
		CreateAt: time.Now().Unix(),
	}

	u.Documents.PassportNumber = "1111111"
	u.Documents.INN = 111111

	// Подключение к монго
	sess,err := mongo.Dial("mongo://192.168.0.107:27017/cata")
	if err != nil{
		panic(err)
	}

	err=Store(sess,u)
	if err!=nil{
		fmt.Println(err)
	}

}

//функция сохранения документа
func Store(sess *mongo.Session,user User)error{
	err := sess.DB("").C("users").Insert(user)
	if err != nil{
		return err
	}
	return nil
}
```

Важно заметить, что записи в mongo называются документами.
Данные мы сохранили, но как же первичный ключ? Первичный ключ в mongo выглядит как "_id", и если его не указать специально, то mongo задаст его автоматически.
Давайте теперь попробуем считать, наши данные:

```
func Find(sess *mongo.Session ,name string)(User,error){
```

```
	var u User
	q := bson.M{
		"name" : "Наземнов Глеб Андреевич",
	}
	err := sess.DB("").C("users").Find(q).One(&u)
	if err != nil{
		return u,err
	}
	return u,nil
}
```

Во всех случаях, указывается:

- DB - имя базы данных
- C - коллекция(аналог таблиц)

В случаи поиска, q - это запрос. Он тоже составляется по средствам bson.

```
	q := bson.M{
		"name" : "Наземнов Глеб Андреевич",
	}
```

Полезные ссылки:

- Официальная документация mongo
- MongoDB Go Driver туториал

Задание:

1. Создайте репозиторий module08_05 в группе с вашими репозиториями - golang_users_repos/<your_gitlab_id>.
2. Для выполнения задания вам понадобиться экземпляр mongodb. Для выполнения задания достаточно docker containeer со стандартными настройками, который можно запустить командой - docker run bitnami/mongodb:latest. более подробно о запуске по ссылке - bitnami-docker-mongodb.
3. Создайте коллекцию users и заполните ее случайными значениями (заполнять можете как вручную так и програмно, достаточно 10-15 записей), соответствующими структуре:

```
type User struct {
	Name       string `bson:"name"`
	Sex        string `bson:"sex"`
	Smartphone struct {
```

```
        Model string `bson:"model"`
        // производитель телефона
        Vendor string `bson:"vendor"`
    } `bson:"smartphone"`
    DateOfBirth int64 `bson:"date_of_birth"`
}
```

4. Напишите программу, которая:
    - подключается к mongo
    - агреггирует данные по модели телефона и полу.
    - при выполнения аггрегирования используйте aggregate.
    - полученные данные должны сохраняться в файл data.json.
5. В ответе пришлите ссылку на ваш репозиторий, в котором должны быть:
    - код программы
    - Файл с результатом работы программы - data.json