New chat

Write me a code in Go language that shows how to interact with SQL database using GORM library

Sure! Here's an example code in Go language that uses the GORM library to interact with an SQL database:

go Copy code

```go
package main

import (
	"fmt"

	"gorm.io/driver/mysql"
	"gorm.io/gorm"
)

type User struct {
	ID        uint   `gorm:"primaryKey"`
	Name      string
	Age       int
	Email     string
	CreatedAt int64
	UpdatedAt int64
}

func main() {
	// Connect to the database
	dsn := "user:password@tcp(localhost:3306)/database_name?charset=utf8mb4&parse
	db, err := gorm.Open(mysql.Open(dsn), &gorm.Config{})
	if err != nil {
		panic(err)
	}
	fmt.Println("Successfully connected to the database!")

	// Auto-migrate the schema
	err = db.AutoMigrate(&User{})
	if err != nil {
        panic(err)
    }

    // Create a new user
    user := User{Name: "John", Age: 30, Email: "john@example.com"}
    result := db.Create(&user)
    if result.Error != nil {
        panic(result.Error)
    }
    fmt.Println("New user created with ID:", user.ID)

    // Update the user's age
    user.Age = 35
    result = db.Save(&user)
    if result.Error != nil {
        panic(result.Error)
    }
    fmt.Println("User's age updated to:", user.Age)

    // Delete the user
    result = db.Delete(&user)
    if result.Error != nil {
        panic(result.Error)
    }
    fmt.Println("User deleted")
}
```

In this example code, we define a `User` struct that maps to a table in our SQL database. We then use the GORM library to interact with the database.

First, we connect to the database using the `gorm.Open` function and pass in the connection details as a Data Source Name (DSN). We then use the `db.AutoMigrate` function to automatically create the `users` table in the database based on our `User` struct.

Next, we create a new user by instantiating a `User` struct and calling the `db.Create` function. We then update the user's age by modifying the `User` struct and calling the `db.Save` function. Finally, we delete the user by calling the `db.Delete` function.

Note that this is just a simple example code to demonstrate the basic usage of GORM. In a real-world application, you would likely want to define additional models, use transactions, and implement more complex queries.

‹ 2 / 2 ›

Rewrite the code to use SQLite instead

Sure! Here's the same code from the previous answer, modified to use SQLite instead of MySQL:

go Copy code

```go
package main

import (
    "fmt"

    "gorm.io/driver/sqlite"
    "gorm.io/gorm"
)

type User struct {
    ID        uint   `gorm:"primaryKey"`
    Name      string
    Age       int
    Email     string
    CreatedAt int64
    UpdatedAt int64
}

func main() {
    // Connect to the database
    db, err := gorm.Open(sqlite.Open("test.db"), &gorm.Config{})
    if err != nil {
        panic(err)
    }
    fmt.Println("Successfully connected to the database!")

    // Auto-migrate the schema
    err = db.AutoMigrate(&User{})
    if err != nil {
        panic(err)
    }

    // Create a new user
    user := User{Name: "John", Age: 30, Email: "john@example.com"}
    result := db.Create(&user)
    if result.Error != nil {
        panic(result.Error)
    }
    fmt.Println("New user created with ID:", user.ID)

    // Update the user's age
    user.Age = 35
    result = db.Save(&user)
    if result.Error != nil {
        panic(result.Error)
    }
    fmt.Println("User's age updated to:", user.Age)

    // Delete the user
    result = db.Delete(&user)
    if result.Error != nil {
        panic(result.Error)
    }
    fmt.Println("User deleted")
}
```

The main difference in this code compared to the previous one is that we now use the `sqlite` driver from GORM to connect to an SQLite database. We pass the path to the database file ("test.db") as an argument to the `sqlite.Open` function.

Other than that, the code remains the same. We define a `User` struct, use `db.AutoMigrate` to create the necessary table(s), and use GORM functions to create, update, and delete users.

Rewrite the code and add rest endpoints to mimic a basic CRUD application. Use only Go standard library for webserver part

‹ 2 / 2 ›

Sure! Here's the modified code that adds REST endpoints to the previous code using only the Go standard library:

go Copy code

```go
package main

import (
    "encoding/json"
    "fmt"
    "log"
    "net/http"
    "strconv"

    "gorm.io/driver/sqlite"
    "gorm.io/gorm"
)

type User struct {
    ID        uint   `gorm:"primaryKey"`
    Name      string
    Age       int
    Email     string
    CreatedAt int64
    UpdatedAt int64
}

func main() {
    // Connect to the database
    db, err := gorm.Open(sqlite.Open("test.db"), &gorm.Config{})
    if err != nil {
        log.Fatal(err)
    }
    fmt.Println("Successfully connected to the database!")

    // Auto-migrate the schema
    err = db.AutoMigrate(&User{})
    if err != nil {
        log.Fatal(err)
    }
    // Define the HTTP handlers
    http.HandleFunc("/users", func(w http.ResponseWriter, r *http.Request) {
        switch r.Method {
        case http.MethodGet:
            getUsersHandler(w, r, db)
        case http.MethodPost:
            createUserHandler(w, r, db)
        default:
            http.Error(w, "Method not allowed", http.StatusMethodNotAllowed)
        }
    })

    http.HandleFunc("/users/", func(w http.ResponseWriter, r *http.Request) {
        switch r.Method {
        case http.MethodGet:
            getUserHandler(w, r, db)
        case http.MethodPut:
            updateUserHandler(w, r, db)
        case http.MethodDelete:
            deleteUserHandler(w, r, db)
        default:
            http.Error(w, "Method not allowed", http.StatusMethodNotAllowed)
        }
    })

    // Start the HTTP server
    log.Fatal(http.ListenAndServe(":8080", nil))
}

func getUsersHandler(w http.ResponseWriter, r *http.Request, db *gorm.DB) {
    var users []User
    result := db.Find(&users)
    if result.Error != nil {
        http.Error(w, result.Error.Error(), http.StatusInternalServerError)
        return
    }
    json.NewEncoder(w).Encode(users)
}

func createUserHandler(w http.ResponseWriter, r *http.Request, db *gorm.DB) {
    var user User
    err := json.NewDecoder(r.Body).Decode(&user)
    if err != nil {
        http.Error(w, err.Error(), http.StatusBadRequest)
        return
    }
    result := db.Create(&user)
    if result.Error != nil {
        http.Error(w, result.Error.Error(), http.StatusInternalServerError)
        return
    }
    json.NewEncoder(w).Encode(user)
}

func getUserHandler(w http.ResponseWriter, r *http.Request, db *gorm.DB) {
    id, err := strconv.Atoi(r.URL.Path[len("/users/"):])
    if err != nil {
        http.Error(w, "Invalid user ID", http.StatusBadRequest)
        return
    }
    var user User
    result := db.First(&user, id)
    if result.Error != nil {
        http.Error(w, result.Error.Error(), http.StatusNotFound)
        return
    }
    json.NewEncoder(w).Encode(user)
}

func updateUserHandler(w http.ResponseWriter, r *http.Request, db *gorm.DB) {
    id, err := strconv.Atoi(r.URL.Path[len("/users/"):])
    if err != nil {
        http.Error(w, "Invalid user ID", http.StatusBadRequest)
        return
    }
    var user User
    result := db.First(&user, id)
    if result.Error != nil {
        http.Error(w, result.Error.Error(), http.StatusNotFound)
        return
    }
    err = json.NewDecoder(r.Body).Decode
```

The code is unfinished, finish it please

Sorry about that, here's the complete code:

go Copy code

```go
package main

import (
    "encoding/json"
    "fmt"
    "log"
    "net/http"
    "strconv"

    "gorm.io/driver/sqlite"
    "gorm.io/gorm"
)

type User struct {
    ID        uint   `gorm:"primaryKey"`
    Name      string
    Age       int
    Email     string
    CreatedAt int64
    UpdatedAt int64
}

func main() {
    // Connect to the database
    db, err := gorm.Open(sqlite.Open("test.db"), &gorm.Config{})
    if err != nil {
        log.Fatal(err)
    }
    fmt.Println("Successfully connected to the database!")

    // Auto-migrate the schema
    err = db.AutoMigrate(&User{})
    if err != nil {
        log.Fatal(err)
    }

    // Define the HTTP handlers
    http.HandleFunc("/users", func(w http.ResponseWriter, r *http.Request) {
        switch r.Method {
        case http.MethodGet:
            getUsersHandler(w, r, db)
        case http.MethodPost:
            createUserHandler(w, r, db)
        default:
            http.Error(w, "Method not allowed", http.StatusMethodNotAllowed)
        }
    })

    http.HandleFunc("/users/", func(w http.ResponseWriter, r *http.Request) {
        switch r.Method {
        case http.MethodGet:
            getUserHandler(w, r, db)
        case http.MethodPut:
            updateUserHandler(w, r, db)
        case http.MethodDelete:
            deleteUserHandler(w, r, db)
        default:
            http.Error(w, "Method not allowed", http.StatusMethodNotAllowed)
        }
    })

    // Start the HTTP server
    log.Fatal(http.ListenAndServe(":8080", nil))
}

func getUsersHandler(w http.ResponseWriter, r *http.Request, db *gorm.DB) {
    var users []User
    result := db.Find(&users)
    if result.Error != nil {
        http.Error(w, result.Error.Error(), http.StatusInternalServerError)
        return
    }
    json.NewEncoder(w).Encode(users)
}

func createUserHandler(w http.ResponseWriter, r *http.Request, db *gorm.DB) {
    var user User
    err := json.NewDecoder(r.Body).Decode(&user)
    if err != nil {
        http.Error(w, err.Error(), http.StatusBadRequest)
        return
    }
    result := db.Create(&user)
    if result.Error != nil {
        http.Error(w, result.Error.Error(), http.StatusInternalServerError)
        return
    }
    json.NewEncoder(w).Encode(user)
}

func getUserHandler(w http.ResponseWriter, r *http.Request, db *gorm.DB) {
    id, err := strconv.Atoi(r.URL.Path[len("/users/"):])
    if err != nil {
        http.Error(w, "Invalid user ID", http.StatusBadRequest)
        return
    }
    var user User
    result := db.First(&user, id)
    if result.Error != nil {
        http.Error(w, result.Error.Error(), http.StatusNotFound)
        return
    }
    json.NewEncoder(w).Encode(user)
}

func updateUserHandler(w http.ResponseWriter, r *http.Request, db *gorm.DB) {
    id, err := strconv.Atoi(r.URL.Path[len("/users/"):])
    if err != nil {
        http.Error(w, "Invalid user ID", http.StatusBadRequest)
        return
    }
    var user User
    result := db.First(&user, id)
    if result.Error != nil {
        http.Error(w, result.Error.Error(), http.StatusNotFound)
        return
    }
    err = json.NewDecoder
```

Ответ оборвался, закончи его

There was an error generating a response

Regenerate response

ChatGPT Feb 13 Version. Free Research Preview. Our goal is to make AI systems more natural and safe to interact with. Your feedback will help us improve.