SQLBuilder Mini Project Setup

This document explains how to set up a Go project using `github.com/deoxxa/sqlbuilder` with examples of each file.

---

1. Project Folder Structure

```
sqlbuilder-demo/
├─ main.go
├─ db/
│   └─ db.go
├─ models/
│   ├─ user.go
│   ├─ order.go
│   └─ product.go
└─ queries/
    ├─ user_queries.go
    └─ order_queries.go
```

---

2. Initialize Go Module

```
go mod init sqlbuilder-demo
go get github.com/deoxxa/sqlbuilder
go get github.com/mattn/go-sqlite3  # for SQLite driver
```

---

3. Database Setup (SQLite example)

```
CREATE TABLE users (
    id INTEGER PRIMARY KEY,
    name TEXT,
    age INTEGER,
    status TEXT
);

CREATE TABLE products (
    id INTEGER PRIMARY KEY,
    name TEXT,
    price REAL
);

CREATE TABLE orders (
    id INTEGER PRIMARY KEY,
    user_id INTEGER,
    product_id INTEGER,
    quantity INTEGER,
    FOREIGN KEY(user_id) REFERENCES users(id),
    FOREIGN KEY(product_id) REFERENCES products(id)
);
```

---

4. DB Connection (`db/db.go`)

```go
package db

import (
    "database/sql"
    _ "github.com/mattn/go-sqlite3"
)

var DB *sql.DB

func Connect() error {
    var err error
    DB, err = sql.Open("sqlite3", "demo.db")
    if err != nil {
        return err
    }
    return DB.Ping()
}
```

---

5. Models (Optional)

`models/user.go`

```go
package models

type User struct {
    ID      int
    Name    string
    Age    int
    Status string
}
```

`models/order.go`

```
package models

type Order struct {
    ID        int
    UserID    int
    ProductID int
    Quantity  int
}
```

`models/product.go`

```
package models

type Product struct {
    ID    int
    Name  string
    Price float64
}
```

---

6. Queries Using SQLBuilder

`queries/user_queries.go`

```
package queries

import (
    "fmt"
    "sqlbuilder-demo/db"
    "github.com/deoxxa/sqlbuilder"
)

func GetActiveUsers() {
    qb := sqlbuilder.New("users").
            Select("id", "name", "age").
            Where(sqlbuilder.Col("status").EQ("active").
                And(sqlbuilder.Col("age").GT(18)))
    sqlStr, args, _ := qb.SQLArgs()
    fmt.Println("SQL:", sqlStr)
    fmt.Println("Args:", args)

    rows, err := db.DB.Query(sqlStr, args...)
    if err != nil {
        panic(err)
    }
    defer rows.Close()

    for rows.Next() {
        var id int
        var name string
        var age int
        rows.Scan(&id, &name, &age)
        fmt.Println(id, name, age)
    }
}
```

queries/order_queries.go

```
package queries

import (
    "fmt"
    "sqlbuilder-demo/db"
    "github.com/deoxxa/sqlbuilder"
)

func GetOrdersWithDetails() {
    ordersBuilder := sqlbuilder.New("orders o").
        Select("o.id", "u.name AS user_name", "p.name AS product_name",
"o.quantity").
        Join("users u", "u", "o.user_id = u.id").
        Join("products p", "p", "o.product_id = p.id")

    sqlStr, args, _ := ordersBuilder.SQLArgs()
    fmt.Println("SQL:", sqlStr)
    fmt.Println("Args:", args)

    rows, err := db.DB.Query(sqlStr, args...)
    if err != nil {
        panic(err)
    }
    defer rows.Close()
    for rows.Next() {
        var id int
        var userName, productName string
        var qty int
        rows.Scan(&id, &userName, &productName, &qty)
        fmt.Println(id, userName, productName, qty)
    }
}
```

Subquery / Reference Example

```
subQuery := sqlbuilder.New("orders").
            Select("user_id", "COUNT(*) AS order_count").
            GroupBy("user_id").
            Having(sqlbuilder.Col("order_count").GT(1)).
            As("user_orders")

qb := sqlbuilder.New("users u").
        Select("u.id", "u.name", "uo.order_count").
        Join(subQuery, "uo", "u.id = uo.user_id")

sqlStr, args, _ := qb.SQLArgs()
fmt.Println("SQL:", sqlStr)
fmt.Println("Args:", args)
```

---

7. Main File (`main.go`)

```
package main

import (
    "fmt"
    "sqlbuilder-demo/db"
    "sqlbuilder-demo/queries"
)

func main() {
    err := db.Connect()
    if err != nil {
        panic(err)
    }

    fmt.Println("Active users:")
    queries.GetActiveUsers()

    fmt.Println("Orders with details:")
    queries.GetOrdersWithDetails()
}
```

---

8. Key Points

- `sqlbuilder` only generates SQL queries safely.
- Execution is done with `database/sql` (or other drivers).
- Supports joins, references/subqueries, dynamic WHERE clauses, GROUP BY, ORDER BY, LIMIT.
- Helps prevent SQL injection by separating query and arguments.

This setup allows you to experiment with SQLBuilder to build complex queries in a structured way.