3 Structured Query Language (SQL) Pada Bahasa Pemrograman Go

Objektif :

- **Mahasiswa Diharapkan Mampu Mengetahui dan Memahami Konsep SQL**
- **Mahasiswa Diharapkan Mampu Menggunakan Perintah SQL**
- **Mahasiswa Diharapkan Mampu Menggunakan SQL pada Bahasa pemrograman Golang**

3.1 Sekilas Tentang SQL

SQL (*Structured Query Language*) adalah sebuah bahasa yang digunakan untuk berinteraksi dengan database secara terstruktur. Bahasa SQL ini dibuat sebagai bahasa yang dapat merelasikan beberapa tabel dalam database maupun merelasikan antar database. Pada bab ini hanya akan sekilas membahas mengenai sintaks yang ada pada SQL dan akan lebih membahas mengenai cara menggunakan *Database* dengan Bahasa pemrograman Golang.

SQL dibagi menjadi 3 bentuk, yaitu :

1. Data Definition Language (DDL)

   DDL adalah sebuah metode *Query* SQL yang berguna untuk mendefinisikan struktur pada sebuah Database, sintaks yang dimiliki DDL adalah *Create, Drop* dan *Alter*.

2. Data Manipulation Language (DML)

   DML adalah sebuah metode *Query* SQL yang berguna untuk melakukan pemanipulasian data pada *database* yang telah dibuat, DML selalu berhubungan dengan data, sintaks yang dimiliki DML adalah *Select, Insert, Update* dan *Delete*.

3. Data Control Language (DCL)

   DCL adalah sebuah metode *Query* SQL yang digunakan untuk memberikan hak otorisasi mengakses Database, mengalokasikan *space*, pendefinisian *space*, dan pengauditan penggunaan *database*, sintaks yang dimiliki DCL adalah *Grant, Revoke, Commit*, dan *Rollback*.

3.2 Penggunaan SQL pada Bahasa Pemrograman Golang

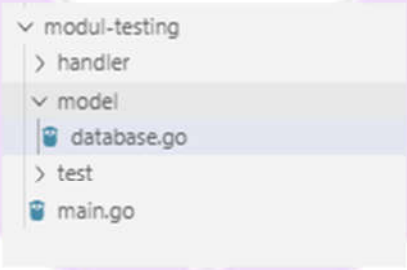
Ketika baru melakukan instalasi golang pada komputer yang digunakan tidak dapat langsung menggunakan query SQL, hal ini dikarenakan *package database/sql* yang berisikan *generic interface* untuk keperluan penggunaan SQL tidak dapat berjalan jika belum tersedia *driver database engine* pada komputer, hal yang harus dilakukan adalah dengan mengunduh *package driver database engine* terlebih dahulu, contoh dari driver *engine* disini adalah Mysql, Oracle, MS SQL Server, dll. Pada modul ini driver yang akan digunakan adalah Mysql. Dibawah ini adalah contoh untuk mengunduh driver mysql.

```
c:\golang\src>go get github.com/go-sql-driver/mysql
```

Gambar 3.1 Mysql Driver

3.3 Data Definition Language (DDL) menggunakan Golang

Pada umumnya pembuatan *database* dan *table* dilakukan langsung menggunakan *mysql console* atau pun *mysql gui* seperti *phpMyAdmin*, namun agar lebih memaksimalkan penggunaan golang, di modul ini pembuatan *database* dan *table* dilakukan melalui kode dan menjalankan *testing* untuk memastikan bahwa *database* dan *table* sudah terbuat. Sebelum memulai pastikan sudah membuat struktur *folder* seperti dibawah ini.



Gambar 3.2 Strukur 1

Setelah itu pada bagian database.go ketikan seperti kode berikut ini

```go
package model

import (
    "database/sql"
    "fmt"
    _ "github.com/go-sql-driver/mysql"
)

type Table interface {
    Name() string
    Field() ([]string, []interface{})
}

func Connect(username, password, host, database string) (*sql.DB, error) {
    conn := fmt.Sprintf("%s:%s@tcp(%s:3306)/%s", username, password, host, database)
    db, err := sql.Open("mysql", conn)
    return db, err
}

func CreateDB(db *sql.DB, name string) error {
    query := fmt.Sprintf("CREATE DATABASE %v", name)
    _, err := db.Exec(query)
    return err
}

func CreateTable(db *sql.DB, query string) error {
    _, err := db.Exec(query)
    return err
}

func DropDB(db *sql.DB, name string) error {
    query := fmt.Sprintf("DROP DATABASE  %v", name)
    _, err := db.Exec(query)
    return err
}

```

Gambar 3.3 Database.GO

- Library

  Pada *file* database.go diatas terdapat 3 library yang digunakan yaitu

  1. database/sql

     Package ini berisikan generic interface untuk keperluan penggunaan SQL.

  2. Fmt

     Package ini mengimplementasikan input atau ouput yang dapat diformat dengan fungsi analog dengan menggunakan printf dan scanf. Penulisan ini mirip seperti pada keluarga Bahasa C namun lebih sederhana.

  3. go-sql-driver/mysql.

     Package ini menyediakan driver MySQL untuk *package database/sq*l. Driver ini harus digunakan bersamaan dengan paket database / sql.

- Fungsi **Connect**

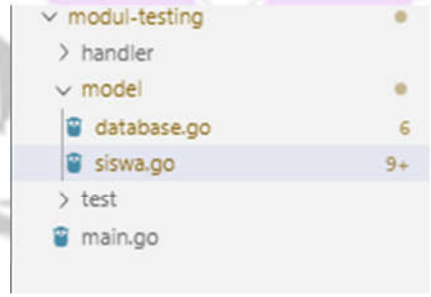
Fungsi ini dibuat untuk melakukan koneksi dengan database, terdapat 4 parameter utama yang harus dimasukan ke dalam fungsi ini, yaitu username, password, database, host. Pada baris ke 15 terdapat penggunaan *fmt.Sprintf* yang berfungsi untuk *formatting* yang akan mengeluarkan hasil string tanpa melakukan cetak apapun, string ini Merupakan *Data Source Name* yang berfungsi untuk melakukan koneksi dengan database, berikut ini adalah bentuk dari koneksi string **(user:pass@tipeprotokol(host:port)/namadb)** sebagai contoh dapat dilihat seperti berikut ini **(root:root@tcp(localhost:3306)/testDatabase),** lalu menggunakan fungsi dari *package database/sql* yaitu **sql.Open** untuk menjalankan koneksi string.

- Fungsi **CreateDB** dan **DropDB**

  Fungsi ini digunakan untuk membuat *database* dan menghapus *database*. Fungsi yang dipakai dari *package database/sql* adalah **db.Exec(query),** fungsi ini digunakan untuk menjalankan query sql yang dimasukan, sebagai contoh dapat dilihat pada baris 21 dan 32 pada Gambar 3.3, baris tersebut adalah baris untuk membuat dan menghapus database, %v yang ada baris tersebut akan digantikan dengan nama database.

3.4 Data Manipulation Language (DML) menggunakan Golang

Pada bagian ini akan mempelajari bagaimana cara menggunakan operasi DML pada Bahasa pemrograman golang, query dml yang digunakan yaitu Select, Insert, Update dan Delete. Sebelum melanjutkan ke dalam program pastikan struktur *folder* sudah seperti dibawah ini.



Gambar 3.4 Struktur 2

Pada modul ini penggunaan Operasi DML di golang akan selalu berhubungan dengan method yang memiliki *method receiver* berupa *struct* dengan *property field-field* yang memiliki nama yang sama dengan *field* dari table di database yang akan digunakan. Pada Gambar 5 terdapat variable TableSiswa yang berisikan suatu string yang akan dimasukan ke

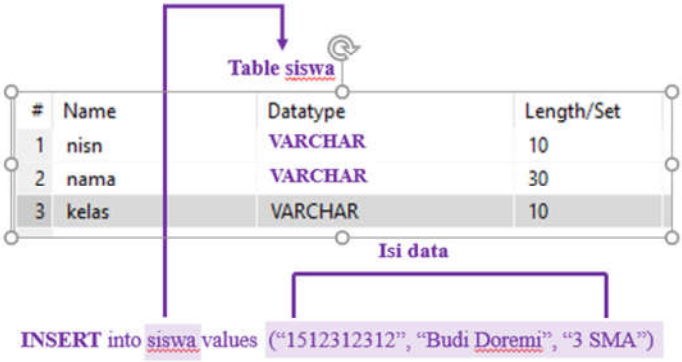
dalam query **db.Exec()** yang terletak pada fungsi **CreateTable** *file* **database.go** untuk membuat suatu tabel, lalu terdapat Struct Siswa yang berisi property berupa field yang ada pada tabel siswa, lalu terdapat method *field* dan *structure* yang berfungsi untuk memberikan pen definiasian dari struct Siswa. Untuk lebih jelasnya dapat dilihat pada Gambar 3.5 dibawah ini.

```go
package model

import (
    "database/sql"
    "fmt"
    "strings"
)

var TabelSiswa string = `
    CREATE TABLE siswa(
        nisn VARCHAR(10) PRIMARY KEY,
        nama VARCHAR(30),
        kelas VARCHAR(10)
    );
`

type Siswa struct {
    NISN  string `json:"NISN"`
    Nama  string `json:"Nama"`
    Kelas string `json:"Kelas"`
}

func (m *Siswa) Fields() ([]string, []interface{}) {
    fields := []string{"nisn", "nama", "kelas"}
    temp := []interface{}{&m.NISN, &m.Nama, &m.Kelas}
    return fields, temp
}

func (m *Siswa) Structur() *Siswa {
    return &Siswa{}
}
```

Gambar 3.5 Siswa.GO

- Fungsi Insert

  Untuk memasukkan data ke dalam sebuah tabel, dapat menggunakan perintah SQL Insert. Berikut adalah anatomi sintaksis untuk perintah *insert* di SQL.



Gambar 3.6 Anatomi Insert

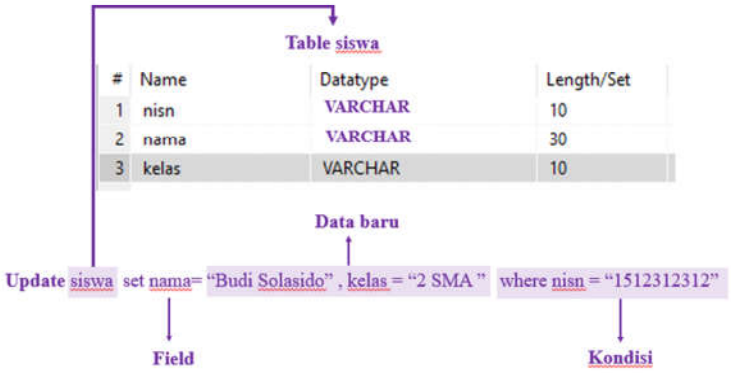
Untuk melakukan operasi *Insert* pada golang, Fungsi yang dipakai dari *package database/sql* adalah **db.Exec(),** seperti pada contoh-contoh sebelumnya, namun dapat dilihat pada baris ke 4, query yang dituliskan berbeda dengan yang ada pada anatomi sintaksis diatas, isi dari values hanya berupa tanda tanya, hal ini berhubungan dengan fungsi **db.Exec()** yang dimana tanda tanya akan digantikan dengan parameter yang ada pada fungsi **db.Exec()**. Untuk lebih jelasnya dapat dilihat pada Gambar 3.7 dibawah ini.

```
33  func (m *Siswa) Insert(db *sql.DB) error {
34      query := fmt.Sprintf("INSERT INTO %v values(?,?, ?)", "siswa")
35      _, err := db.Exec(query, &m.NISN, &m.Nama, &m.Kelas)
36      return err
37  }
38
```

Gambar 3.7 Insert Pada Golang

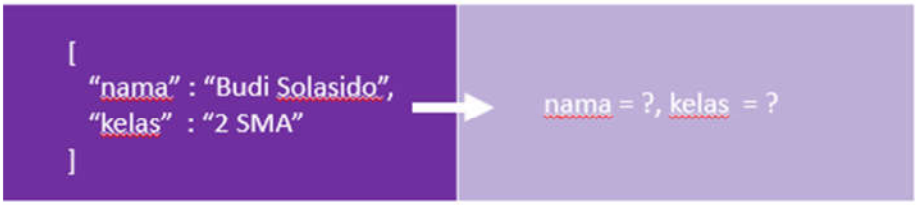
- Fungsi Update

  Untuk melakukan ubah data yang ada pada sebuah table dapat menggunakan perintah SQL Update. Berikut adalah anatomi sintaksis untuk perintah *update* di SQL.



Gambar 3.8 Anatomi Update

Untuk melakukan operasi Update masih menggunakan Fungsi **db.Exec()** seperti pada contoh-contoh sebelumnya, namun terdapat beberapa proses tambahan, seperti pada baris 47-54 yang Merupakan perulangan untuk mengubah format penulisan map menjadi string seperti contoh anatomi dibawah ini.



Gambar 3.9 Convert Data

Tanda tanya diatas akan digantikan dengan isi yang dimasukan ke dalam variable args, lalu pada baris ke 56 kondisi *where* akan dimasukan ke dalam variable args seperti sebelumnya dan akan dieksekusi pada baris ke 58. Untuk lebih jelasnya dapat dilihat pada Gambar 3.10 dibawah ini.

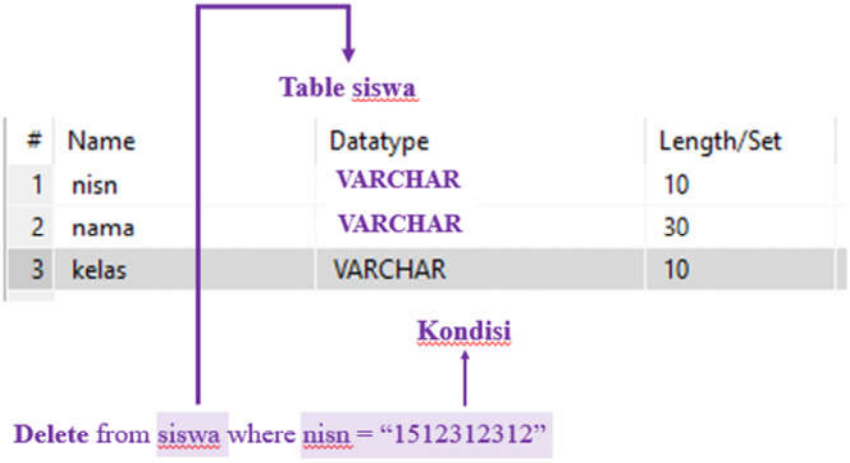
(args... adalah konsep variadic function atau pembuatan fungsi dengan parameter sejenis yang tak terbatas. Maksud tak terbatas disini adalah jumlah parameter yang disisipkan ketika pemanggilan fungsi bisa berapa saja, kalau dalam persamaan matematika seperti args1, args2 .... argsN).

```
39  func (m *Siswa) Update(db *sql.DB, data map[string]interface{}) error {
40
41      var kolom = []string{}
42      var args []interface{}
43
44      i := 1
45
46      // Ini loop data untuk dimasukan kedalam set,
47      for key, value := range data {
48          updateData := fmt.Sprintf("%v = ?", strings.ToLower(key))
49          kolom = append(kolom, updateData)
50          args = append(args, value)
51          i++
52      }
53      // Ubah array menjadi string dengan pemisah koma
54      dataUpdate := strings.Join(kolom, ",")
55      query := fmt.Sprintf("UPDATE %s SET %s WHERE %s = ?", "siswa", dataUpdate, "NISN")
56      args = append(args, m.NISN)
57      // Exec dengan query yang ada
58      _, err := db.Exec(query, args...)
59      return err
60  }
```

Gambar 3.10 Update Pada Golang

- Fungsi Delete

  Untuk melakukan hapus data yang ada pada sebuah tabel dapat menggunakan perintah SQL Delete. Berikut adalah anatomi sintaksis untuk perintah *delete* di SQL.



Gambar 3.11 Anatomi Delete

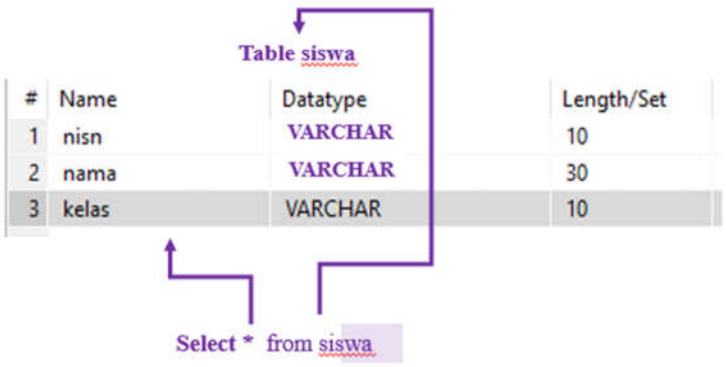
Untuk operasi *Delete* memiliki penggunaan yang sama seperti operasi *Insert*, hanya berbeda pada querynya saja dan parameternya saja yang dimana jika delete parameternya adalah suatu kondisi. Untuk lebih jelasnya dapat dilihat pada Gambar 3.12 dibawah ini.

```
61
62  func (m *Siswa) Delete(db *sql.DB) error {
63      query := fmt.Sprintf("DELETE FROM %s WHERE %s = ?", "siswa", "NISN")
64      // Exec dengan query yang ada
65      _, err := db.Exec(query, m.NISN)
66      return err
67  }
68
```

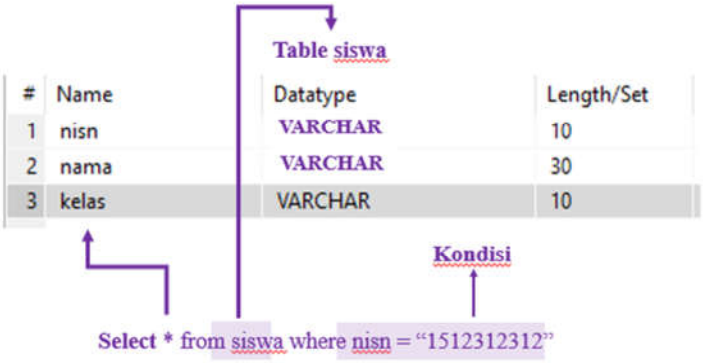
Gambar 3.12 Delete Pada Golang

- Fungsi Get

Untuk mengambil data pada sebuah tabel dapat menggunakan perintah SQL Select. Dibawah ini terdapat 2 contoh penggunaan Select, yaitu yang mengambil seluruh data dan untuk mengambil 1 data saja berdasarkan kondisi. Berikut adalah anatomi sintaksis untuk perintah *Select* di SQL.



Gambar 3.13 Anatomi Select Seluruh Data



Gambar 3.14 Anatomi Select Satu Data

Untuk operasi Select fungsi yang digunakan adalah **db.Query** dan **db.QueryRow**, perbedaanya adalah dari jumlah data yang akan diambil, jika db.Query akan mengambil seluruh data sedangkan db.QueryRow hanya akan mengambil 1 data saja.

a) Mengambil seluruh data

Untuk mengambil seluruh data, fungsi yang digunakan adalah db.Query, dapat dilihat pada baris ke 87 Gambar 3.15, lalu setelah query dieksekusi diperlukan 1 varibale baru bertipe data pointer Struct untuk menampung data yang akan dimasukan melalui perulangan data.Next(). Perulangan dengan cara ini dilakukan sebanyak jumlah total *record* yang ada, berurutan dari *record* pertama hingga akhir, satu per satu. *Method Scan()* milik *sql.Rows* berfungsi untuk mengambil nilai *record* yang sedang diiterasi, untuk disimpan pada variabel *pointer*. Pada contoh dibawah ini nilai *record* akan dimasukan ke dalam *method Fields* pada *struct* siswa. Untuk lebih jelasnya dapat dilihat pada Gambar 3.15 dibawah ini.

```go
func GetAllSiswa(db *sql.DB) ([]*Siswa, error) {
    m := &Siswa{}
    query := fmt.Sprintf("SELECT * FROM %s", "siswa")
    data, err := db.Query(query)
    if err != nil {
        return nil, err
    }

    defer data.Close()
    var result []*Siswa
    for data.Next() {
        each := m.Structur()
        _, dst := each.Fields()
        err := data.Scan(dst...)
        if err != nil {
            return nil, err
        }
        result = append(result, each)
    }
    return result, nil

}
```

Gambar 3.15 Select Seluruh Data Pada Golang

b) Mengambil salah satu data

Untuk *query* yang menghasilkan 1 baris *record* saja, bisa gunakan *method db.QueryRow()*, dan bisa langsung di *chaining* atau digabungkan dengan method Scan agar data yang didapatkan akan langsung dimasukan ke dalam *variable each*. Untuk lebih jelasnya dapat dilihat pada gambar dibawah ini.

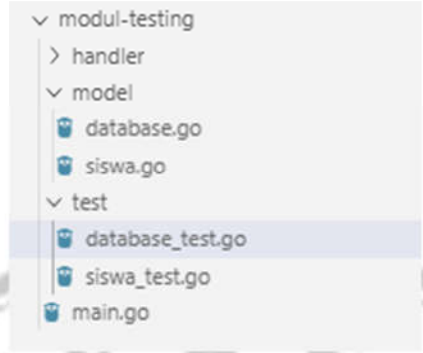
```go
func Get(db *sql.DB, id string) (*Siswa, error) {

    m := &Siswa{}
    each := m.Structur()
    _, dst := each.Fields()
    query := fmt.Sprintf("SELECT * FROM %s WHERE %s = ?", "siswa", "NISN")

    // isinya akan dimasukan kedalam var dst yang dideklarasikan diatas
    err := db.QueryRow(query, id).Scan(dst...)
    if err != nil {
        return nil, err
    }
    return each, nil
}
```

Gambar 3.16 Select Salah Satu Data Pada Golang

3.5 Membuat File *Testing*

Pada bab sebelumnya sudah dibahas bagaimana caranya melakukan *testing* pada suatu program, pada bagian ini *testing* akan dibuat dan dilakukan pada program yang lebih kompleks dari sebelumnya, yaitu melakukan *testing* pada fungsi-fungsi SQL yang sudah dibuat pada bagian sebelumnya, dimulai dari membuat database, membuat table, mengisi

data, melakukan update data, menghapus data, mengambil seluruh data, dan mengambil salah satu data saja. Untuk program *testing*nya dapat dilihat pada kode dibawah ini. Sebelum melanjutkan ke dalam program pastikan struktur *folder* sudah seperti dibawah ini.



Gambar 3.17 Struktur 3

Lalu isikan database_test.go seperti pada Gambar 3.18. Untuk cara menulis *testing* tidak akan dijelaskan kembali dikarenakan sudah dibahas pada bab sebelumnya, untuk isi *testing* sendiri hanya memanggil fungsi SQL yang sebelumnya sudah dibuat dan terdapat 1 fungsi initDatabase yang berguna untuk membuat koneksi secara global.

```go
package test

import (
    "database/sql"
    "fmt"
    "modul-testing/model"
    "testing"
)

var username, password, host, namaDB, defaultDB string

func init() {
    username = "root"
    password = ""
    host = "localhost"
    namaDB = "sekolah"
    defaultDB = "mysql"
}

run test | debug test
func TestDatabase(t *testing.T) {

    t.Run("Testing untuk membuat database", func(t *testing.T) {
        db, err := model.Connect(username, password, host, defaultDB)
        defer db.Close()
        if err != nil {
            t.Fatal(err)
        }
        err = model.CreateDB(db, namaDB)
        if err != nil {
            err = model.DropDB(db, namaDB)
            if err != nil {
                t.Fatal(err)
            }

            err = model.CreateDB(db, namaDB)
            if err != nil {
                t.Fatal(err)
            }
        }

    })

    t.Run("Testing untuk memeriksa koneksi database", func(t *testing.T) {

        db, err := model.Connect(username, password, host, defaultDB)

        defer db.Close()

        if err != nil {
            t.Fatal(err)
        }
    })

```

```
53
54      t.Run("Testing untuk membuat table Siswa", func(t *testing.T) {
55          db, err := model.Connect(username, password, password, namaDB)
56
57          if err != nil {
58              fmt.Println("Gagal melakukan koneksi")
59              t.Fatal(err)
60          }
61
62          if err = model.CreateTable(db, model.TabelSiswa); err != nil {
63              fmt.Println("Gagal membuat table Siswa")
64              t.Fatal(err)
65          }
66      })
67  }
68
69  func initDatabase() (*sql.DB, error) {
70      db, err := model.Connect(username, password, password, namaDB)
71      if err != nil {
72          fmt.Println("Gagal melakukan koneksi")
73          return nil, err
74      }
75      return db, nil
76  }
```

Gambar 3.18 *Testing* DDL

Serta isikan pula siswa_test.go seperti pada Gambar 3.19.

```
1   package test
2
3   import (
4       "modul-testing/model"
5       "testing"
6   )
7
    run test | debug test
8   func TestSiswa(t *testing.T) {
9
10      var dataInsertSiswa = []model.Siswa{
11          model.Siswa{
12              NISN:  "123456731",
13              Nama:  "Budi Doremi",
14              Kelas: "1 SMA",
15          },
```

```
        model.Siswa{
            NISN:  "192837231",
            Nama:  "Doremi Budi",
            Kelas: "2 SMA",
        },
    }

    db, err := initDatabase()
    if err != nil {
        t.Fatal(err)
    }

    t.Run("Testing insert  mahasiswwa", func(t *testing.T) {
        for _, dataInsert := range dataInsertSiswa {
            err := dataInsert.Insert(db)
            if err != nil {
                t.Fatal(err)
            }
        }
    })

    t.Run("Testing update  mahasiswwa", func(t *testing.T) {
        var updateData = map[string]interface{}{
            "Nama": "Susi SUanti",
        }

        data := dataInsertSiswa[0]
        if err := data.Update(db, updateData); err != nil {
            t.Fatal(err)
        }

    })

    t.Run("Testing Get mahasiswa", func(t *testing.T) {
        _, err := model.Get(db, "123456731")
        if err != nil {
            t.Fatal(err)
        }
    })

    t.Run("Testing Get mahasiswa", func(t *testing.T) {
        _, err := model.GetAllSiswa(db)
        if err != nil {
            t.Fatal(err)
        }

    })

    t.Run("Testing delete mahasiswa", func(t *testing.T) {
        data := dataInsertSiswa[1]
        if err := data.Delete(db); err != nil {
            t.Fatal(err)
        }
    })

    defer db.Close()
}
```

Gambar 3.19 *Testing* DML

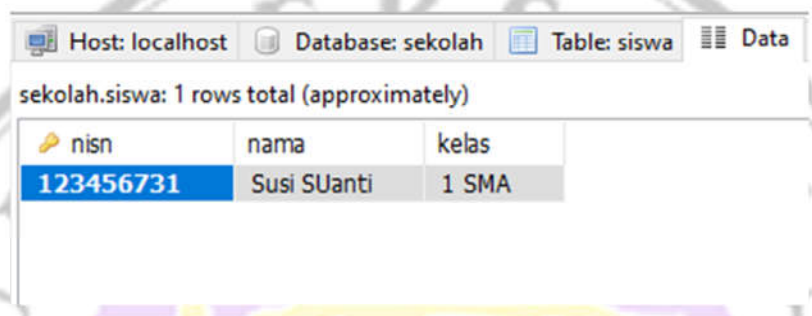
3.6 Melakukan *Testing* pada Fungsi SQL

Untuk melakukan *testing* pastikan terlebih dahulu seluruh *file testing* memiliki akhiran nama *file* berupa _test.go dan pastikan pula *file* berada pada *folder* test. Lalu untuk menjalankan *testing* dapat mengetikan go test -run [Nama Fungsi Yang akan Di *testing*], agar lebih jelasnya dapat dilihat pada Gambar 3.20 dibawah ini.

```
Inspiron        MINGW64 /c/golang/src/modul-testing/test
$ go test -run TestDatabase
PASS
ok      modul-testing/test      0.528s

Inspiron        MINGW64 /c/golang/src/modul-testing/test
$ go test -run TestSiswa
PASS
ok      modul-testing/test      0.564s
```

Gambar 3.20 *Command Testing*

Untuk hasilnya dapat dicek pada Gambar 3.21 dibawah ini, dapat dilihat tabel sudah terbuat dan data sudah di masukan, diupdate dan dihapus.



Gambar 3. 21 Hasil *Testing*