Exercise 4

CST8333 19F

Professor: Stanley Pieda

By: Lucas Estienne (esti0011)

040 819 959

## Evidence of Learning

* 1. Database Init & Create table:

// open db

    database, \_ := sql.Open("sqlite3", filePath)

    // create table if not exist

    statement, \_ := database.Prepare(`

        CREATE TABLE IF NOT EXISTS cheeses (

            id INTEGER PRIMARY KEY,

            cheese\_id INTEGER,

            cheese\_name TEXT,

            manufacturer\_name TEXT,

            manufacturer\_prov\_code TEXT,

            manufacturing\_type TEXT,

            website TEXT,

            fat\_content\_percent REAL,

            moisture\_percent REAL,

            particularities TEXT,

            flavour TEXT,

            characteristics TEXT,

            ripening TEXT,

            organic INTEGER,

            category\_type TEXT,

            milk\_type TEXT,

            milk\_treatment\_type TEXT,

            rind\_type TEXT,

            last\_update\_date TEXT

        )

    `)

    statement.Exec()

* 1. Database Insert:

// prepare insert

        statement, \_ := database.Prepare(`

            INSERT INTO cheeses (

                cheese\_id, cheese\_name, manufacturer\_name, manufacturer\_prov\_code,

                manufacturing\_type, website, fat\_content\_percent, moisture\_percent,

                particularities, flavour, characteristics, ripening,

                organic, category\_type, milk\_type, milk\_treatment\_type,

                rind\_type, last\_update\_date

            ) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)

        `)

        // exec insert

        statement.Exec(

            records[i].CheeseId, records[i].CheeseName, records[i].ManufacturerName, records[i].ManufacturerProvCode,

            records[i].ManufacturingType, records[i].WebSite, records[i].FatContentPercent, records[i].MoisturePercent,

            records[i].Particularities, records[i].Flavour, records[i].Characteristics, records[i].Ripening,

            records[i].Organic, records[i].CategoryType, records[i].MilkType, records[i].MilkTreatmentType,

            records[i].RindType, records[i].LastUpdateDate,

        )

* 1. Database select:

// prepare select

    statement, \_ := database.Prepare(`

        SELECT cheese\_id, cheese\_name, manufacturer\_name, manufacturer\_prov\_code,

        manufacturing\_type, website, fat\_content\_percent, moisture\_percent,

        particularities, flavour, characteristics, ripening,

        organic, category\_type, milk\_type, milk\_treatment\_type,

        rind\_type, last\_update\_date FROM cheeses WHERE id = $1

    `)

    // query select

    rows, \_ := statement.Query(id)

* 1. Unit Test:

// create a record with the proper data

    firstRecord := Record {

        CheeseId: 228,

        CheeseName: "Sieur de Duplessis (Le)",

        ManufacturerName: "Fromages la faim de loup",

        ManufacturerProvCode: "NB",

        ManufacturingType: "Farmstead",

        WebSite: "N/A",

        FatContentPercent: 24.2,

        MoisturePercent: 47,

        Particularities: "N/A",

        Flavour: "Sharp, lactic",

        Characteristics: "Uncooked",

        Ripening: "9 Months",

        Organic: false,

        CategoryType: "Firm Cheese",

        MilkType: "Ewe",

        MilkTreatmentType: "Raw Milk",

        RindType: "Washed Rind",

        LastUpdateDate: "2016-02-03",

    }

retrievedFirstRecord := getCheeseByRecordId(rid, database)

    // check if loaded first record and our test record are equal

    if !reflect.DeepEqual(firstRecord, retrievedFirstRecord) {

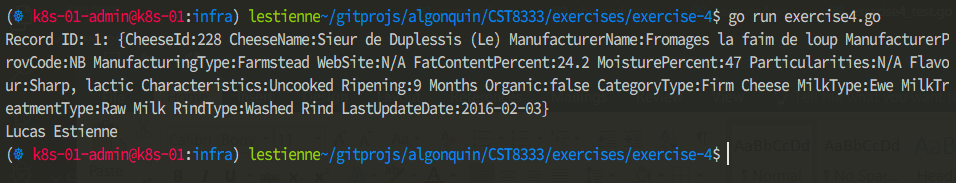
       t.Errorf("Loaded First Record was incorrect, \n got: \n%+v\n, want: \n%+v\n", firstRecord, retrievedFirstRecord)

    }

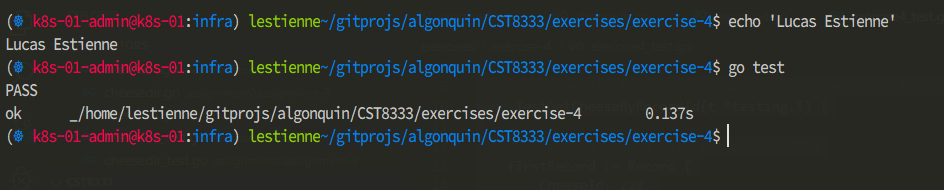
## Research

1]N. Raboy, "Using A SQLite Database For Local Data In A Golang Application", *The Polyglot Developer*, 2017. [Online]. Available: https://www.thepolyglotdeveloper.com/2017/04/using-sqlite-database-golang-application/. [Accessed: 11- Nov- 2019].

## Program Demonstration via Screen Shot



## Unit Testing Demonstration via Screen Shot



## Source Code

### exercise4.go

// CST8333 Exercise 4 - Lucas Estienne

package main

import (

    "fmt"

    "database/sql"

    "os"

    "log"

    "strings"

    "strconv"

    "encoding/csv"

    \_ "github.com/mattn/go-sqlite3"

)

const NumRecordsToLoad = 10

type Record struct {

    CheeseId int

    CheeseName string

    ManufacturerName string

    ManufacturerProvCode string

    ManufacturingType string

    WebSite string

    FatContentPercent float32

    MoisturePercent float32

    Particularities string

    Flavour string

    Characteristics string

    Ripening string

    Organic bool

    CategoryType string

    MilkType string

    MilkTreatmentType string

    RindType string

    LastUpdateDate string

}

// main function, this is the entrypoint

func main() {

    // load data

    records := loadData("data/canadianCheeseDirectory.csv", NumRecordsToLoad)

    // init db

    database := initCheesesDatabase("./exercise4.db")

    // record id to retrieve at the end

    rid := 1

    // insert records into DB

    insertRecords(records, database)

    // get record at rid in cheeses table

    r := getCheeseByRecordId(rid, database)

    fmt.Printf("Record ID: %d: %+v\n", rid, r)

    fmt.Println("Lucas Estienne")

}

// helper function to do error handling

func check(e error) {

    if e != nil {

        log.Fatal("Error", e)

        panic(e)

    }

}

// function to open/initialize cheeses database

func initCheesesDatabase(filePath string) \*sql.DB {

    // open db

    database, \_ := sql.Open("sqlite3", filePath)

    // create table if not exist

    statement, \_ := database.Prepare(`

        CREATE TABLE IF NOT EXISTS cheeses (

            id INTEGER PRIMARY KEY,

            cheese\_id INTEGER,

            cheese\_name TEXT,

            manufacturer\_name TEXT,

            manufacturer\_prov\_code TEXT,

            manufacturing\_type TEXT,

            website TEXT,

            fat\_content\_percent REAL,

            moisture\_percent REAL,

            particularities TEXT,

            flavour TEXT,

            characteristics TEXT,

            ripening TEXT,

            organic INTEGER,

            category\_type TEXT,

            milk\_type TEXT,

            milk\_treatment\_type TEXT,

            rind\_type TEXT,

            last\_update\_date TEXT

        )

    `)

    statement.Exec()

    return database

}

// function to insert records into DB

func insertRecords(records []Record, database \*sql.DB) {

    // loop through all records

    for i := 0; i < len(records); i++ {

        // prepare insert

        statement, \_ := database.Prepare(`

            INSERT INTO cheeses (

                cheese\_id, cheese\_name, manufacturer\_name, manufacturer\_prov\_code,

                manufacturing\_type, website, fat\_content\_percent, moisture\_percent,

                particularities, flavour, characteristics, ripening,

                organic, category\_type, milk\_type, milk\_treatment\_type,

                rind\_type, last\_update\_date

            ) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)

        `)

        // exec insert

        statement.Exec(

            records[i].CheeseId, records[i].CheeseName, records[i].ManufacturerName, records[i].ManufacturerProvCode,

            records[i].ManufacturingType, records[i].WebSite, records[i].FatContentPercent, records[i].MoisturePercent,

            records[i].Particularities, records[i].Flavour, records[i].Characteristics, records[i].Ripening,

            records[i].Organic, records[i].CategoryType, records[i].MilkType, records[i].MilkTreatmentType,

            records[i].RindType, records[i].LastUpdateDate,

        )

    }

}

func getCheeseByRecordId(id int, database \*sql.DB) Record {

    var (

        cheeseId int

        cheeseName string

        manufacturerName string

        manufacturerProvCode string

        manufacturingType string

        website string

        fatContentPercent float32

        moisturePercent float32

        particularities string

        flavour string

        characteristics string

        ripening string

        organic bool

        categoryType string

        milkType string

        milkTreatmentType string

        rindType string

        lastUpdateDate string

    )

    // prepare select

    statement, \_ := database.Prepare(`

        SELECT cheese\_id, cheese\_name, manufacturer\_name, manufacturer\_prov\_code,

        manufacturing\_type, website, fat\_content\_percent, moisture\_percent,

        particularities, flavour, characteristics, ripening,

        organic, category\_type, milk\_type, milk\_treatment\_type,

        rind\_type, last\_update\_date FROM cheeses WHERE id = $1

    `)

    // query select

    rows, \_ := statement.Query(id)

    for rows.Next() {

        err := rows.Scan(

            &cheeseId, &cheeseName, &manufacturerName, &manufacturerProvCode,

            &manufacturingType, &website, &fatContentPercent, &moisturePercent,

            &particularities, &flavour, &characteristics, &ripening,

            &organic, &categoryType, &milkType, &milkTreatmentType,

            &rindType, &lastUpdateDate,

        )

        if err != nil {

            log.Fatal(err)

        }

    }

    r := Record {

        CheeseId: int(cheeseId),

        CheeseName: cheeseName,

        ManufacturerName: manufacturerName,

        ManufacturerProvCode: manufacturerProvCode,

        ManufacturingType: manufacturingType,

        WebSite: website,

        FatContentPercent: float32(fatContentPercent),

        MoisturePercent: float32(moisturePercent),

        Particularities: particularities,

        Flavour: flavour,

        Characteristics: characteristics,

        Ripening: ripening,

        Organic: organic,

        CategoryType: categoryType,

        MilkType: milkType,

        MilkTreatmentType: milkTreatmentType,

        RindType: rindType,

        LastUpdateDate: lastUpdateDate,

    }

    return r

}

// function load or reload data

func loadData(filePath string, numRecords int) []Record {

    var records []Record

    // Load lines from CSV

    lines, err := getLinesFromCSV(filePath)

    check(err)

    // get rid of column names

    lines = lines[1:]

    // convert lines to records slice

    for i := 0; i < numRecords && i < len(lines); i++ {

        records = append(records, lineToRecord(lines[i]))

    }

    return records

}

// helper function to read CSV

func getLinesFromCSV(filePath string) (lines [][]string, err error) {

    // open file

    file, err := os.Open(filePath)

    check(err)

    defer file.Close() // defer closing the file until function returns

    // create CSV Reader from file

    reader := csv.NewReader(file)

    return reader.ReadAll()

}

// function to convert CSV line to Record object

func lineToRecord(line []string) Record {

    // parse some values from strings

    cheeseId, err := strconv.ParseInt(line[0], 10, 64)

    if err != nil { cheeseId = 0 }

    fatContentPercent, err := strconv.ParseFloat(line[10], 32)

    if err != nil { fatContentPercent = 0.0 }

    moisturePercent, err := strconv.ParseFloat(line[11], 32)

    if err != nil { moisturePercent = 0.0 }

    organic, err := strconv.ParseBool(line[20])

    if err != nil { organic = false }

    return Record {

        CheeseId: int(cheeseId),

        CheeseName: getFirstNonEmptyStringOrNA(line[1], line[2]),

        ManufacturerName: getFirstNonEmptyStringOrNA(line[3], line[4]),

        ManufacturerProvCode: getFirstNonEmptyStringOrNA(line[5], "??"),

        ManufacturingType: getFirstNonEmptyStringOrNA(line[6], line[7]),

        WebSite: getFirstNonEmptyStringOrNA(line[8], line[9]),

        FatContentPercent: float32(fatContentPercent),

        MoisturePercent: float32(moisturePercent),

        Particularities: getFirstNonEmptyStringOrNA(line[12], line[13]),

        Flavour: getFirstNonEmptyStringOrNA(line[14], line[15]),

        Characteristics: getFirstNonEmptyStringOrNA(line[16], line[17]),

        Ripening: getFirstNonEmptyStringOrNA(line[18], line[19]),

        Organic: organic,

        CategoryType: getFirstNonEmptyStringOrNA(line[21], line[22]),

        MilkType: getFirstNonEmptyStringOrNA(line[23], line[24]),

        MilkTreatmentType: getFirstNonEmptyStringOrNA(line[25], line[26]),

        RindType: getFirstNonEmptyStringOrNA(line[27], line[28]),

        LastUpdateDate: line[29],

    }

}

// helper function to return the first of two non empty strings, or the string "N/A"

func getFirstNonEmptyStringOrNA(first string, second string) string {

    if strings.TrimSpace(first) != "" {

        return first

    } else if strings.TrimSpace(second) != "" {

        return second

    } else {

        return "N/A"

    }

}

### exercise4\_test.go

// CST8333 Exercise 4 - Unit Tests - Lucas Estienne

package main

import (

    "testing"

    "reflect"

)

// test to verify that our "loadData" function loads the first record from the dataset properly

func TestGetCheeseByRecordId(t \*testing.T) {

    // create a record with the proper data

    firstRecord := Record {

        CheeseId: 228,

        CheeseName: "Sieur de Duplessis (Le)",

        ManufacturerName: "Fromages la faim de loup",

        ManufacturerProvCode: "NB",

        ManufacturingType: "Farmstead",

        WebSite: "N/A",

        FatContentPercent: 24.2,

        MoisturePercent: 47,

        Particularities: "N/A",

        Flavour: "Sharp, lactic",

        Characteristics: "Uncooked",

        Ripening: "9 Months",

        Organic: false,

        CategoryType: "Firm Cheese",

        MilkType: "Ewe",

        MilkTreatmentType: "Raw Milk",

        RindType: "Washed Rind",

        LastUpdateDate: "2016-02-03",

    }

    // load data and get the first record

    records := loadData("data/canadianCheeseDirectory.csv", 5)

    // init db

    database := initCheesesDatabase("./exercise4-test.db")

    // insert records into DB

    insertRecords(records, database)

    // record id to test

    rid := 1

    retrievedFirstRecord := getCheeseByRecordId(rid, database)

    // check if loaded first record and our test record are equal

    if !reflect.DeepEqual(firstRecord, retrievedFirstRecord) {

       t.Errorf("Loaded First Record was incorrect, \n got: \n%+v\n, want: \n%+v\n", firstRecord, retrievedFirstRecord)

    }

}