Final Project

CST8333 19F

Professor: Stanley Pieda

By: Lucas Estienne (esti0011)

040 819 959

## Evidence of Learning Final Project Feature

// function to search/filter and display records

func searchRecords(database \*sql.DB) {

    fmt.Printf("\nSearch for a record...\n\n")

    colOne, valOne := searchRecordHelper()

    colTwo, valTwo := searchRecordHelper()

    colThree, valThree := searchRecordHelper()

    fmt.Printf("\nDisplaying all records matching your filters, multithreaded...\n")

    var wg sync.WaitGroup

    rs := filterRecords(database, colOne, valOne, colTwo, valTwo, colThree, valThree)

    // Tell the waitgroup how many threads are about to run concurrently.

    wg.Add(len(rs))

    for i := 0; i < len(rs); i++ {

        // Spawn a thread (goroutine) for each iteration in the loop.

        go func(id int, r Record) {

            // At the end of the goroutine, tell the waitgroup that the thread has completed

            defer wg.Done()

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

        }(i, rs[i])

    }

}

func filterRecords(database \*sql.DB, colOne string, valOne string, colTwo string, valTwo string, colThree string, valThree string) []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

    )

    var rs []Record

    // workaround for where clause

    q := fmt.Sprintf(`

        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 %s = $1 AND %s = $2 AND %s = $3

    `, colOne, colTwo, colThree)

    // prepare select

    statement, \_ := database.Prepare(q)

    // query select

    rows, \_ := statement.Query(valOne, valTwo, valThree)

    // loop through resultset

    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)

        }

        // append to our resulting record slice

        rs = append(rs, 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 rs

}

// helper function for search

func searchRecordHelper() (string, string) {

    c := ""

    s := ""

    columns :=  []string {

        "cheese\_name", "manufacturer\_name", "manufacturer\_prov\_code", "manufacturing\_type", "website",

        "particularities", "flavour", "characteristics", "ripening", "category\_type", "milk\_type",

        "milk\_treatment\_type", "rind\_type", "last\_update\_date",

    }

    for c == "" {

        fmt.Printf("\n Please pick one of the following columns to filter records on:")

        fmt.Printf("\n%v: ", columns)

        \_, err := fmt.Scanf("%s", &c)

        if err != nil {

            c = ""

            fmt.Println("\nPlease enter a valid selection.")

        } else if !stringInSlice(c, columns) {

            c = ""

            fmt.Printf("\nPlease enter a valid (from the list) column name to filter on.\n")

        }

    }

    s = readString(c)

    return c, s

}

// helper to check if a string is in a string slice

func stringInSlice(a string, list []string) bool {

    for \_, b := range list {

        if b == a {

            return true

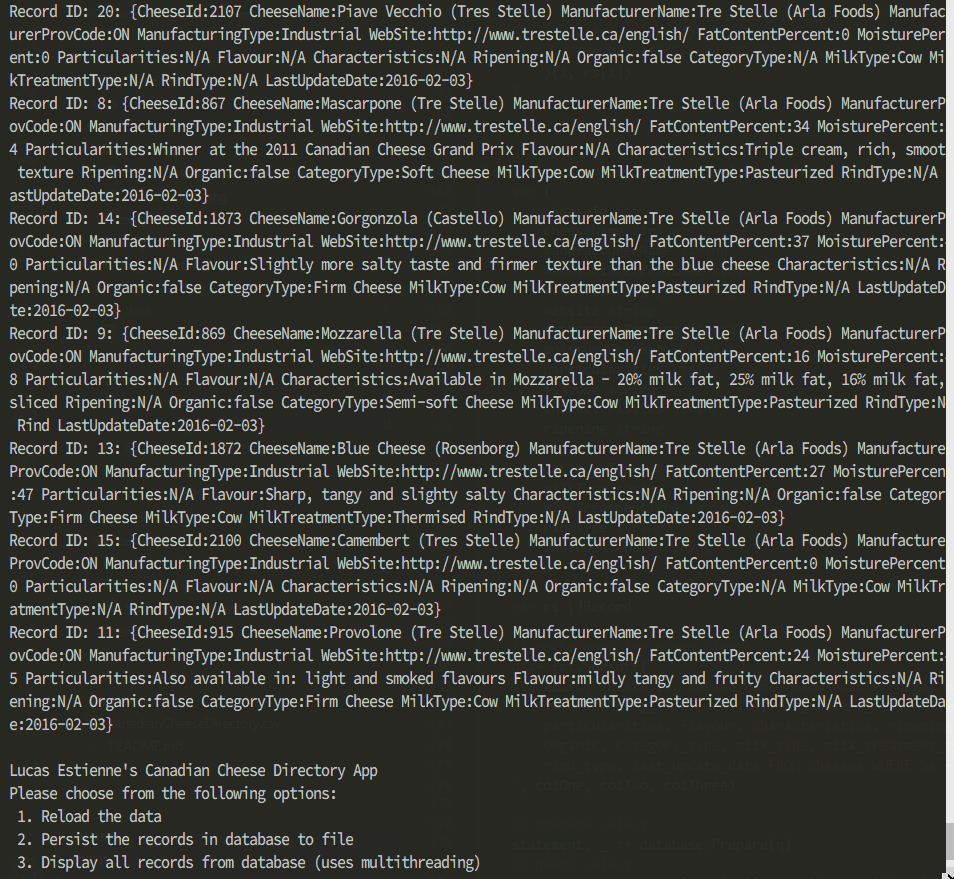
        }

    }

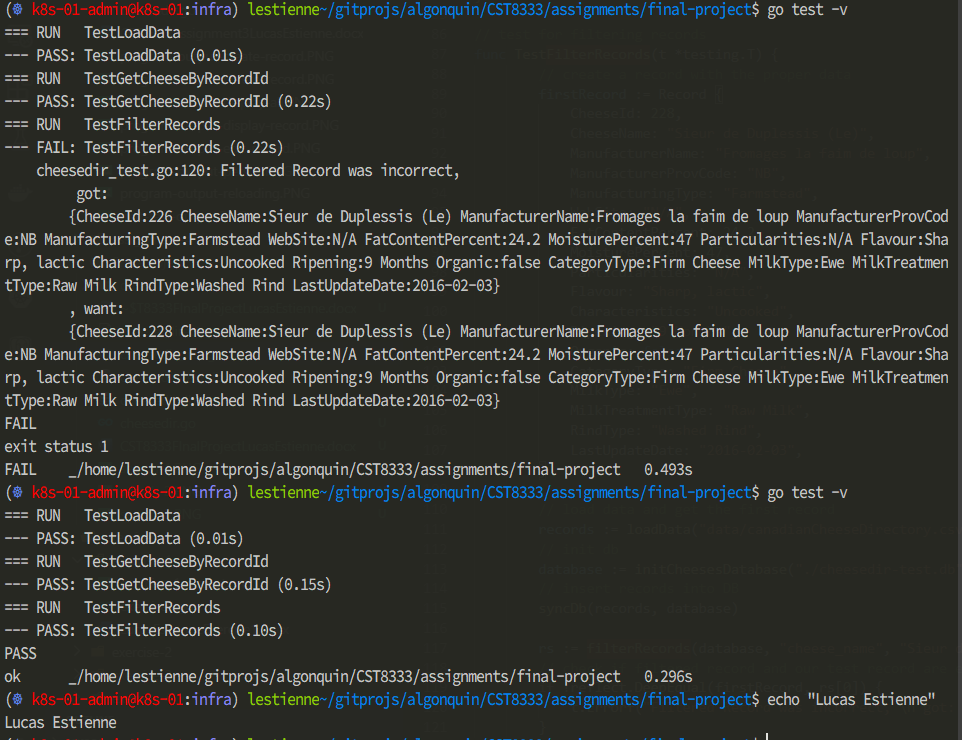
    return false

}

## Program Demonstration via Screen Shot



## Unit Testing Demonstration via Screen Shots



## Learning Resources

[1]"TIOBE Index | TIOBE - The Software Quality Company", *Tiobe.com*, 2019. [Online]. Available: https://www.tiobe.com/tiobe-index/. [Accessed: 12- Sep- 2019].

[2]"PYPL PopularitY of Programming Language index", *Pypl.github.io*, 2019. [Online]. Available: http://pypl.github.io/PYPL.html. [Accessed: 12- Sep- 2019].

[3]"Stack Overflow Developer Survey 2019", *Stack Overflow*, 2019. [Online]. Available: https://insights.stackoverflow.com/survey/2019#technology. [Accessed: 12- Sep- 2019].

[4]"testing - The Go Programming Language", *Golang.org*, 2019. [Online]. Available: https://golang.org/pkg/testing/. [Accessed: 12- Sep- 2019].

[5]A. Ellis, "Golang basics - writing unit tests", *alex ellis' blog*, 2017. [Online]. Available: https://blog.alexellis.io/golang-writing-unit-tests/. [Accessed: 12- Sep- 2019].

[6]M. Richards, "Software architecture patterns", *O'Reilly Media*, 2015. [Online]. Available: https://www.oreilly.com/ideas/software-architecture-patterns/page/2/layered-architecture. [Accessed: 13- Sep- 2019].

[7]"What is Agile Software Development?", *Agile Alliance*, 2019. [Online]. Available: https://www.agilealliance.org/agile101/. [Accessed: 23- Sep- 2019].

[8]"Agile Methodologies", *Blueprintsys.com*, 2019. [Online]. Available: https://www.blueprintsys.com/agile-development-101/agile-methodologies. [Accessed: 23- Sep- 2019].

[9]"Agile Model & Methodology: Guide for Developers and Testers", *Guru99.com*, 2019. [Online]. Available: https://www.guru99.com/agile-scrum-extreme-testing.html. [Accessed: 23- Sep- 2019].

[10]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].

[11]D. Gorby, "Multi-thread For Loops Easily and Safely in Go", *Medium*, 2016. [Online]. Available: https://medium.com/@greenraccoon23/multi-thread-for-loops-easily-and-safely-in-go-a2e915302f8b. [Accessed: 20- Nov- 2019].

[12] "Effective Go - The Go Programming Language", *Golang.org*, 2019. [Online]. Available: https://golang.org/doc/effective\_go.html. [Accessed: 12- Sep- 2019].

[13]"golang/go – Code Review Comments", *GitHub*, 2019. [Online]. Available: https://github.com/golang/go/wiki/CodeReviewComments. [Accessed: 12- Sep- 2019].

[14]A. Gerrand, "Godoc: documenting Go code - The Go Blog", *Blog.golang.org*, 2011. [Online]. Available: https://blog.golang.org/godoc-documenting-go-code. [Accessed: 12- Sep- 2019].  
[15]"Command: godoc", *Godoc.org*, 2019. [Online]. Available: https://godoc.org/golang.org/x/tools/cmd/godoc. [Accessed: 12- Sep- 2019].

[16]"Memory Blocks - Go 101 (an online book for Go programming language + Golang knowledge base)", *Go101.org*, 2019. [Online]. Available: https://go101.org/article/memory-block.html. [Accessed: 16- Sep- 2019].

[17]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].

## Source Code

### cheesedir.go

// CST8333 Cheese Directory App - Lucas Estienne

package main

import (

    "fmt"

    "os"

    "database/sql"

    "log"

    "time"

    "bufio"

    "strings"

    "strconv"

    "encoding/csv"

    "sync"

    \_ "github.com/mattn/go-sqlite3"

)

const NumRecordsToLoad = 10000

const (

    OptionReload = 1

    OptionPersist = 2

    OptionDisplayAll = 3

    OptionCreate = 4

    OptionDisplay = 5

    OptionEdit = 6

    OptionDelete = 7

    OptionSearch = 8

    OptionExit = 9

)

// simple data structure containing a string

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("./cheesedir.db")

    // sync in-memory records data structure with database

    syncDb(records, database)

    // loop until exit

    for true {

        // display menu and process choice

        switch selection := showMenu(); selection {

            case OptionReload:

                fmt.Println("Reloading data...")

                // reload records

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

                // sync in-memory records data structure with database

                syncDb(records, database)

            case OptionPersist:

                persistToFile(database, "cheese\_directory\_output.csv")

            case OptionDisplayAll:

                displayAllRecords(database)

            case OptionCreate:

                // create record

                records = createRecord(records)

                // sync in-memory records data structure with database

                syncDb(records, database)

            case OptionDisplay:

                displayRecord(database)

            case OptionEdit:

                // edit record

                editRecord(records)

                // sync in-memory records data structure with database

                syncDb(records, database)

            case OptionDelete:

                // delete record

                records = deleteRecord(records)

                // sync in-memory records data structure with database

                syncDb(records, database)

            case OptionSearch:

                searchRecords(database)

            case OptionExit:

                fmt.Println("Goodbye")

                return

        }

        time.Sleep(1 \* time.Second)

    }

}

// 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 sync in-memory records data structure with database table

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

    // prepare delete statement for records currently in table

    statement, \_ := database.Prepare(`

        DELETE FROM cheeses;

    `)

    // exec delete

    statement.Exec()

    // 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,

        )

    }

}

// helper function to do error handling

func check(e error) {

    if e != nil {

        log.Fatal("Error", e)

        panic(e)

    }

}

// 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"

    }

}

// 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],

    }

}

// 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

}

// function to show menu and return the user selection

func showMenu() int {

    selection := 0

    fmt.Println("\nLucas Estienne's Canadian Cheese Directory App")

    fmt.Println("Please choose from the following options:")

    fmt.Printf(" %d. Reload the data\n", OptionReload)

    fmt.Printf(" %d. Persist the records in database to file\n", OptionPersist)

    fmt.Printf(" %d. Display all records from database (uses multithreading)\n", OptionDisplayAll)

    fmt.Printf(" %d. Create a new record\n", OptionCreate)

    fmt.Printf(" %d. Display a record from database\n", OptionDisplay)

    fmt.Printf(" %d. Edit a record\n", OptionEdit)

    fmt.Printf(" %d. Delete a record\n", OptionDelete)

    fmt.Printf(" %d. Search a record\n", OptionSearch)

    fmt.Printf(" %d. Exit\n", OptionExit)

    // loop until selection is valid

    for selection == 0 {

        fmt.Printf("Please choose an option: ")

        \_, err := fmt.Scanf("%d", &selection)

        if err != nil {

            selection = 0

            fmt.Println("\nPlease enter a valid option.")

        } else if selection < OptionReload || selection > OptionExit {

            selection = 0

            fmt.Println("\nPlease enter a valid integer between 1 and 8.")

        }

    }

    return selection

}

// function to display all records

func displayAllRecords(database \*sql.DB) {

    fmt.Printf("\nDisplaying all records from database, multithreaded...\n\n")

    var wg sync.WaitGroup

    rs := getAllCheeses(database)

    // Tell the waitgroup how many threads are about to run concurrently.

    wg.Add(len(rs))

    for i := 0; i < len(rs); i++ {

        // Spawn a thread (goroutine) for each iteration in the loop.

        go func(id int, r Record) {

            // At the end of the goroutine, tell the waitgroup that the thread has completed

            defer wg.Done()

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

        }(i, rs[i])

    }

}

// function to search/filter and display records

func searchRecords(database \*sql.DB) {

    fmt.Printf("\nSearch for a record...\n\n")

    colOne, valOne := searchRecordHelper()

    colTwo, valTwo := searchRecordHelper()

    colThree, valThree := searchRecordHelper()

    fmt.Printf("\nDisplaying all records matching your filters, multithreaded...\n")

    var wg sync.WaitGroup

    rs := filterRecords(database, colOne, valOne, colTwo, valTwo, colThree, valThree)

    // Tell the waitgroup how many threads are about to run concurrently.

    wg.Add(len(rs))

    for i := 0; i < len(rs); i++ {

        // Spawn a thread (goroutine) for each iteration in the loop.

        go func(id int, r Record) {

            // At the end of the goroutine, tell the waitgroup that the thread has completed

            defer wg.Done()

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

        }(i, rs[i])

    }

}

func filterRecords(database \*sql.DB, colOne string, valOne string, colTwo string, valTwo string, colThree string, valThree string) []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

    )

    var rs []Record

    // workaround for where clause

    q := fmt.Sprintf(`

        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 %s = $1 AND %s = $2 AND %s = $3

    `, colOne, colTwo, colThree)

    // prepare select

    statement, \_ := database.Prepare(q)

    // query select

    rows, \_ := statement.Query(valOne, valTwo, valThree)

    // loop through resultset

    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)

        }

        // append to our resulting record slice

        rs = append(rs, 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 rs

}

// helper function for search

func searchRecordHelper() (string, string) {

    c := ""

    s := ""

    columns :=  []string {

        "cheese\_name", "manufacturer\_name", "manufacturer\_prov\_code", "manufacturing\_type", "website",

        "particularities", "flavour", "characteristics", "ripening", "category\_type", "milk\_type",

        "milk\_treatment\_type", "rind\_type", "last\_update\_date",

    }

    for c == "" {

        fmt.Printf("\n Please pick one of the following columns to filter records on:")

        fmt.Printf("\n%v: ", columns)

        \_, err := fmt.Scanf("%s", &c)

        if err != nil {

            c = ""

            fmt.Println("\nPlease enter a valid selection.")

        } else if !stringInSlice(c, columns) {

            c = ""

            fmt.Printf("\nPlease enter a valid (from the list) column name to filter on.\n")

        }

    }

    s = readString(c)

    return c, s

}

// helper to check if a string is in a string slice

func stringInSlice(a string, list []string) bool {

    for \_, b := range list {

        if b == a {

            return true

        }

    }

    return false

}

// function to display a specific record

func displayRecord(database \*sql.DB) {

    id := -1

    count := getCheeseCount(database)

    // loop until ID is valid

    for id == -1 {

        fmt.Printf("\n Please enter the # of the record you would like to display: ")

        \_, err := fmt.Scanf("%d", &id)

        if err != nil {

            id = -1

            fmt.Println("\nPlease enter a valid integer.")

        } else if id < 0 || id > count-1 {

            id = -1

            fmt.Printf("\nPlease enter a valid record ID between 0 and %d.\n", count-1)

        }

    }

    r := getCheeseByRecordId(id, database)

    // display record

    fmt.Printf("\n Displaying Record #%d from database: \n%+v\n", id, r)

}

// helper function to delete an element from a Record slice and keep order

func deleteRecordFromSlice(slice []Record, id int) []Record {

    return append(slice[:id], slice[id+1:]...)

}

func deleteRecord(records []Record) []Record {

    id := -1

    // loop until ID is valid

    for id == -1 {

        fmt.Printf("\n Please enter the # of the record you would like to delete: ")

        \_, err := fmt.Scanf("%d", &id)

        if err != nil {

            id = -1

            fmt.Println("\nPlease enter a valid integer.")

        } else if id < 0 || id > len(records)-1 {

            id = -1

            fmt.Printf("\nPlease enter a valid record ID between 0 and %d.\n", len(records)-1)

        }

    }

    // display the record we are deleting

    fmt.Printf("\n Deleting the following record: \n%+v\n", records[id])

    // return a slice with the element removed

    return deleteRecordFromSlice(records, id)

}

// helper function to read a string from stdin

func readString(toRead string) string {

    fmt.Printf("Please enter the %s: ", toRead)

    // read from scanner

    scanner := bufio.NewScanner(os.Stdin)

    scanner.Scan()

    s := scanner.Text()

    if s == "" {

        s = "N/A"

    }

    return s

}

func createRecord(records []Record) []Record {

    var recordSlice []string

    fmt.Printf("\n Creating record...\n\n")

    // read values for our record

    recordSlice = append(recordSlice, readString("Cheese ID (int)")) //0

    recordSlice = append(recordSlice, readString("Cheese Name")) //1

    recordSlice = append(recordSlice, readString("Manufacturer Name")) //2

    recordSlice = append(recordSlice, readString("Manufacturer Prov Code")) //3

    recordSlice = append(recordSlice, readString("Manufacturing Type")) //4

    recordSlice = append(recordSlice, readString("Website")) //5

    recordSlice = append(recordSlice, readString("Fat Content Percent (float32)")) //6

    recordSlice = append(recordSlice, readString("Moisture Percent")) //7

    recordSlice = append(recordSlice, readString("Particularities")) //8

    recordSlice = append(recordSlice, readString("Flavour")) //9

    recordSlice = append(recordSlice, readString("Characteristics")) //10

    recordSlice = append(recordSlice, readString("Ripening")) //11

    recordSlice = append(recordSlice, readString("Organic (bool)")) //12

    recordSlice = append(recordSlice, readString("Category Type")) //13

    recordSlice = append(recordSlice, readString("Milk Type")) //14

    recordSlice = append(recordSlice, readString("Milk Treatment Type")) //15

    recordSlice = append(recordSlice, readString("Rind Type")) //16

    recordSlice = append(recordSlice, readString("Last Update Date")) //17

    // parse some values from strings

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

    if err != nil { cheeseId = 0 }

    fatContentPercent, err := strconv.ParseFloat(recordSlice[6], 32)

    if err != nil { fatContentPercent = 0.0 }

    moisturePercent, err := strconv.ParseFloat(recordSlice[7], 32)

    if err != nil { moisturePercent = 0.0 }

    organic, err := strconv.ParseBool(recordSlice[12])

    if err != nil { organic = false }

    // init record

    r := Record {

        CheeseId: int(cheeseId),

        CheeseName: recordSlice[1],

        ManufacturerName: recordSlice[2],

        ManufacturerProvCode: recordSlice[3],

        ManufacturingType: recordSlice[4],

        WebSite: recordSlice[5],

        FatContentPercent: float32(fatContentPercent),

        MoisturePercent: float32(moisturePercent),

        Particularities: recordSlice[8],

        Flavour: recordSlice[9],

        Characteristics: recordSlice[10],

        Ripening: recordSlice[11],

        Organic: organic,

        CategoryType: recordSlice[13],

        MilkType: recordSlice[14],

        MilkTreatmentType: recordSlice[15],

        RindType: recordSlice[16],

        LastUpdateDate: recordSlice[17],

    }

    fmt.Printf("\n Creating the following record: \n%+v\n", r)

    // return our records slice with the new record appended

    return append(records, r)

}

// helper function to convert a Record object to a slice

func recordToSlice(record Record) []string {

    var recordSlice []string

    recordSlice = []string{

        fmt.Sprintf("%d",record.CheeseId), record.CheeseName, record.ManufacturerName, record.ManufacturerProvCode,

        record.ManufacturingType, record.WebSite, fmt.Sprintf("%.2f", record.FatContentPercent),

        fmt.Sprintf("%.2f", record.MoisturePercent), record.Particularities, record.Flavour,

        record.Characteristics, record.Ripening, fmt.Sprintf("%t", record.Organic),

        record.CategoryType, record.MilkType, record.MilkTreatmentType, record.RindType, record.LastUpdateDate,

    }

    return recordSlice

}

// function to write in-memory records to file

func persistToFile(database \*sql.DB, filePath string) {

    fmt.Printf("\n Writing all database records to %s.\n", filePath)

    rs := getAllCheeses(database)

    headers :=  []string {

        "CheeseId", "CheeseName", "ManufacturerName", "ManufacturerProvCode", "ManufacturingType",

        "WebSite", "FatContentPercent", "MoisturePercent", "Particularities", "Flavour", "Characteristics",

        "Ripening", "Organic", "CategoryType", "MilkType", "MilkTreatmentType", "RindType", "LastUpdateDate",

    }

    // create file

    file, err := os.Create(filePath)

    check(err)

    defer file.Close()

    // initialize csv writer

    writer := csv.NewWriter(file)

    defer writer.Flush()

    // write headers

    err = writer.Write(headers)

    check(err)

    // loop through records and write each one to the CSV

    for i := 0; i < len(rs); i++ {

        err = writer.Write(recordToSlice(rs[i]))

        check(err)

    }

    fmt.Printf("\n Done writing to %s.\n", filePath)

}

// helper function to read a new string or keep the provided default

func readNewOrKeepDefaultString(toRead string, def string) string {

    fmt.Printf("Please enter the %s [%s]: ", toRead, def)

    // read from scanner

    scanner := bufio.NewScanner(os.Stdin)

    scanner.Scan()

    s := scanner.Text()

    if s == "" {

        s = def

    }

    return s

}

// function to edit record

func editRecord(records []Record) []Record {

    var recordSlice []string

    id := -1

    // loop until ID is valid

    for id == -1 {

        fmt.Printf("\n Please enter the # of the record you would like to edit: ")

        \_, err := fmt.Scanf("%d", &id)

        if err != nil {

            id = -1

            fmt.Println("\nPlease enter a valid integer.")

        } else if id < 0 || id > len(records)-1 {

            id = -1

            fmt.Printf("\nPlease enter a valid record ID between 0 and %d.\n", len(records)-1)

        }

    }

    r := records[id]

    // edit record

    fmt.Printf("\n Editing Record #%d: \n%+v\n", id, r)

    fmt.Printf("\n Press Enter to keep the same value, otherwise input your value...\n")

    // read values for our record

    recordSlice = append(recordSlice, readNewOrKeepDefaultString("Cheese ID (int)", fmt.Sprintf("%d",r.CheeseId))) //0

    recordSlice = append(recordSlice, readNewOrKeepDefaultString("Cheese Name", r.CheeseName)) //1

    recordSlice = append(recordSlice, readNewOrKeepDefaultString("Manufacturer Name", r.ManufacturerName)) //2

    recordSlice = append(recordSlice, readNewOrKeepDefaultString("Manufacturer Prov Code", r.ManufacturerProvCode)) //3

    recordSlice = append(recordSlice, readNewOrKeepDefaultString("Manufacturing Type", r.ManufacturingType)) //4

    recordSlice = append(recordSlice, readNewOrKeepDefaultString("Website", r.WebSite)) //5

    recordSlice = append(recordSlice, readNewOrKeepDefaultString("Fat Content Percent (float32)", fmt.Sprintf("%.2f",r.FatContentPercent))) //6

    recordSlice = append(recordSlice, readNewOrKeepDefaultString("Moisture Percent", fmt.Sprintf("%.2f",r.MoisturePercent))) //7

    recordSlice = append(recordSlice, readNewOrKeepDefaultString("Particularities", r.Particularities)) //8

    recordSlice = append(recordSlice, readNewOrKeepDefaultString("Flavour", r.Flavour)) //9

    recordSlice = append(recordSlice, readNewOrKeepDefaultString("Characteristics", r.Characteristics)) //10

    recordSlice = append(recordSlice, readNewOrKeepDefaultString("Ripening", r.Ripening)) //11

    recordSlice = append(recordSlice, readNewOrKeepDefaultString("Organic (bool)", fmt.Sprintf("%t",r.Organic))) //12

    recordSlice = append(recordSlice, readNewOrKeepDefaultString("Category Type", r.CategoryType)) //13

    recordSlice = append(recordSlice, readNewOrKeepDefaultString("Milk Type", r.MilkType)) //14

    recordSlice = append(recordSlice, readNewOrKeepDefaultString("Milk Treatment Type", r.MilkTreatmentType)) //15

    recordSlice = append(recordSlice, readNewOrKeepDefaultString("Rind Type", r.RindType)) //16

    recordSlice = append(recordSlice, readNewOrKeepDefaultString("Last Update Date", r.LastUpdateDate)) //17

    // parse some values from strings

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

    if err != nil { cheeseId = 0 }

    fatContentPercent, err := strconv.ParseFloat(recordSlice[6], 32)

    if err != nil { fatContentPercent = 0.0 }

    moisturePercent, err := strconv.ParseFloat(recordSlice[7], 32)

    if err != nil { moisturePercent = 0.0 }

    organic, err := strconv.ParseBool(recordSlice[12])

    if err != nil { organic = false }

    // replace record

    records[id] = Record {

        CheeseId: int(cheeseId),

        CheeseName: recordSlice[1],

        ManufacturerName: recordSlice[2],

        ManufacturerProvCode: recordSlice[3],

        ManufacturingType: recordSlice[4],

        WebSite: recordSlice[5],

        FatContentPercent: float32(fatContentPercent),

        MoisturePercent: float32(moisturePercent),

        Particularities: recordSlice[8],

        Flavour: recordSlice[9],

        Characteristics: recordSlice[10],

        Ripening: recordSlice[11],

        Organic: organic,

        CategoryType: recordSlice[13],

        MilkType: recordSlice[14],

        MilkTreatmentType: recordSlice[15],

        RindType: recordSlice[16],

        LastUpdateDate: recordSlice[17],

    }

    fmt.Printf("\n Changed the record to record: \n%+v\n", records[id])

    // return our amended records slice

    return records

}

// function to select cheese by record id from database

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)

    // loop through resultset

    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 to select all cheeses from database

func getAllCheeses(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

    )

    var rs []Record

    // 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 ORDER BY id ASC

    `)

    // query select

    rows, \_ := statement.Query()

    //loop through resultset

    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)

        }

        // append to our resulting record slice

        rs = append(rs, 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 rs

}

// function to select the count of cheeses from database

func getCheeseCount(database \*sql.DB) int {

    var count int

    // prepare select

    statement, \_ := database.Prepare(`

        SELECT count(\*) FROM cheeses

    `)

    // query select

    rows, \_ := statement.Query()

    // loop through resultset

    for rows.Next() {

        err := rows.Scan(

            &count,

        )

        if err != nil {

            log.Fatal(err)

        }

    }

    return count

}

### cheesedir\_test.go

// CST8333 Cheese Directory App - 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 TestLoadData(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)

    loadedFirstRecord := records[0]

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

    if !reflect.DeepEqual(firstRecord, loadedFirstRecord) {

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

    }

}

// test to verify that our "getCheeseByRecordId" properly retrieves the first record from the database

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("./cheesedir-test.db")

    // insert records into DB

    syncDb(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)

    }

}

// test for filtering records

func TestFilterRecords(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("./cheesedir-test.db")

        // insert records into DB

        syncDb(records, database)

        rs := filterRecords(database, "cheese\_name", "Sieur de Duplessis (Le)", "flavour", "Sharp, lactic", "ripening", "9 Months")

        // check if filtered record and our test record are equal

        if !reflect.DeepEqual(firstRecord, rs[0]) {

            t.Errorf("Filtered Record was incorrect, \n got: \n%+v\n, want: \n%+v\n", firstRecord, rs[0])

        }

}