# Application Development

Towards real programs in Go

## Admin

- Questions
- Project
- Exam
- Feedback?
- Lets look at the news
  - Lets get a few people to weight in on it.

## Agile

- For those of you who haven’t done recent industry work, Agile is an overused term for a development methodology where
  - you work on a small feature set
  - Ship often – work is done in ‘sprints’ usually 1-2 weeks long
  - See how everything went
  - Then repeat
  - Lets examine some of the good/bad for this

## Retrospective

- The “retrospective” is the ‘See how everything went’ part
  - Traditional questions/topics
    - What went well?
    - What didn’t go well?
    - Questions?

# Retrospective

- The “retrospective” is the ‘See how everything went’ part
  - Because this is an academic setting I’m making a slight change
    - What went well?
    - What didn’t go well?
    - What do you wish you (maybe you plural) knew before starting the sprint/project.
    -

## Retrospective

- So now lets take 10-15 minutes
  - (I'll duck in somewhere in the 11 minute mark to some groups to see if we are more or less done)
  - And then we'll have a representative report out to the whole group.

## Comments

- Lets review
  - How do we create a one line comment in go?
    - Lucky volunteer?

# Comments

- Lets review
  - How do we create a one line comment in go?
    - Lucky volunteer?
  - How about a multi line comment in go?
    - Another lucky volunteer?
  -

# Comments

- Lets review
  - How do we create a one line comment in go?
    - Lucky volunteer?
  - How about a multi line comment in go?
    - Another lucky volunteer?
  - What do I mean by inline comments in go?
    - You know what is coming

# Comments

- Lets review
  - How do we create a one line comment in go?
    - Lucky volunteer?
  - How about a multi line comment in go?
    - Another lucky volunteer?
  - What do I mean by inline comments in go?
    - `defer os.Exit(0) //if we leave this function then  exit with success`
  - Which should you use when commenting your code after you are good at go and why?
    - Distinguish learners from practitioners.

# GoDoc

- GoDoc is a little like javadoc
  - Allows comments in a particular format to be used for library documentation.
  - All uses multiple line comments
    - //this is the first
    - //this is the second
    - Code being commented.
  -

## Run godoc

- Run godoc from command line
  - If needed
  - go get golang.org/x/tools/cmd/godoc
  - It will start a web server serving web pages with documentation

# Application Development

- These days most applications need
  - Data
  - To show that data in a useful format
  - Anything else?
  - Of course there are also games, but maybe we'll look at those in the future
  -

# GUI

- We are showing our data in a GUI
  - Fyne lets us do that in a window on the desktop
  - Or in a web browser
    - So that is more or less covered.

## Data

- Where can we get the data from?
  - Thank-you my wonderful crew of volunteers

## Data

- Where can we get the data from?
  - Thank-you my wonderful crew of volunteers
  - Hopefully your found all of them
    - Maybe files
    - Maybe the network (API calls like we’ve been doing)
    - And maybe databases (we haven’t talked about that yet)

# Data

- Where else do we find a lot of data other than APIs and databases?
  -

## Data

- Where else do we find a lot of data other than APIs and databases?
  - How about excel?
  - Used everywhere in the world – and has lots of data
  -

# Excel with go

- There is a go library that works really well with excel files.
- (unknown unknowns to known unknowns)
- Gone through different maintainers over the years,
- Current:
  - "github.com/xuri/excelize/v2"
- Grab the sample excel file from website

## Simple Excel read

```go
package main
import (
	"fmt"
	"github.com/xuri/excelize/v2"
	"log"
)
func main() {
	file, err := excelize.OpenFile("MedianIncome.xlsx")
	if err != nil {
		log.Fatal(err)
	}
	all_rows, err := file.GetRows("h08")
	if err != nil {
		fmt.Println("Error getting Rows", err)
	}
	for _, row := range all_rows {
		//the titles all have one real cell, but the data we want all has many cells, so only do work
		//when there is more than one cell in the row
		if len(row) > 1 {
			fmt.Println(row[0], " \t:\t ", row[1])
		}
	}
```

# Going beyond reading

- Here are a few links to go beyond reading data
  - Unknown unknowns → known unknowns
- https://www.kelche.co/blog/go/excel/
- https://blog.logrocket.com/building-spreadsheets-go-excelize/
- https://xuri.me/excelize/en/base/installation.html
- Will you be able to cut and paste into your project?
  - No – but if you understand it, you will be good.

## Databases

- How many of you have worked with databases?
  - In Dr. Jung’s class (or in some other undergrad class)
  - In your jobs?
  - Playing around?

# Databases

- How many of you have worked with databases?
- Based on a lot of my other surveys my guess is going to be that about a quarter of you have seen this already and the rest are pretty new
- So for that quarter bear with me for a moment
- References for my images and further reading:
  - https://medium.com/analytics-vidhya/programming-with-databases-in-python-using-sqlite-4cecbef51ab9
-

# Databases in 20 minutes

- So with that
  - We really need to expose all students to some basic data handling in a required class. (you are only required one programming heavy class in the MS program.)
  - In this slideset I'm going to try to summarize a database course in a week or less.
    - This is not a substitute for a real database class!!!!!
    - That said, if you have questions ask them.
  - Part of what I'm trying to do in this class is to give you a taste of some of the programming bits that every graduate student should have

## Database History

- Historically Databases come in lots of difference types
  - Hierarchical databases
  - Network databases
  - Relational databases
  - Object-oriented databases
  - Graph databases
  - ER model databases
  - Document databases

# Databases and SQL

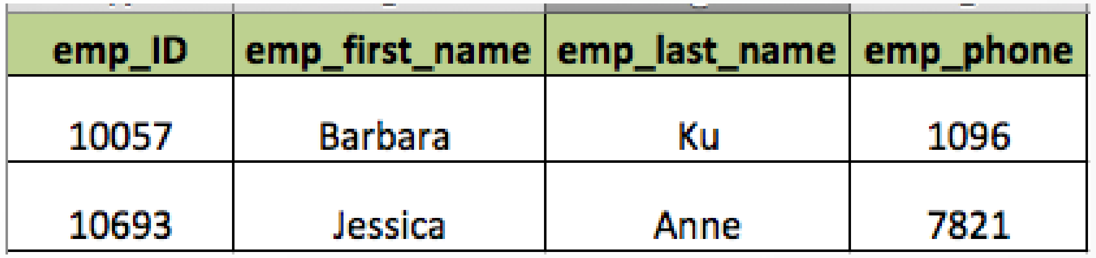
- Realistically, today most people care about 2 of those
  - Relational Databases (SQL)
  - Document Databases (NoSQL)
  - Note on Pronouncing these:
    - Lots of disucussion:
      - https://www.khanacademy.org/computing/computer-programming/sql/sql-basics/v/s-q-l-or-sequel
      - https://database.guide/is-it-pronounced-s-q-l-or-sequel/
    - sqlServer vs MySQL
    - In general I find Americans more likely to use the old IBM “Sequel” pronunciation and the rest of the world to use S-Q-L
    - I learned SQL from a recent immigrant so I often switch back and forth

# Relational Databases

- Relational Databases
  - Developed in the early 1970s
  - By 1980’s were all but the only type used
  - (document databases making a big inroad in this dominance in last decade)

# Relational Database in nutshell

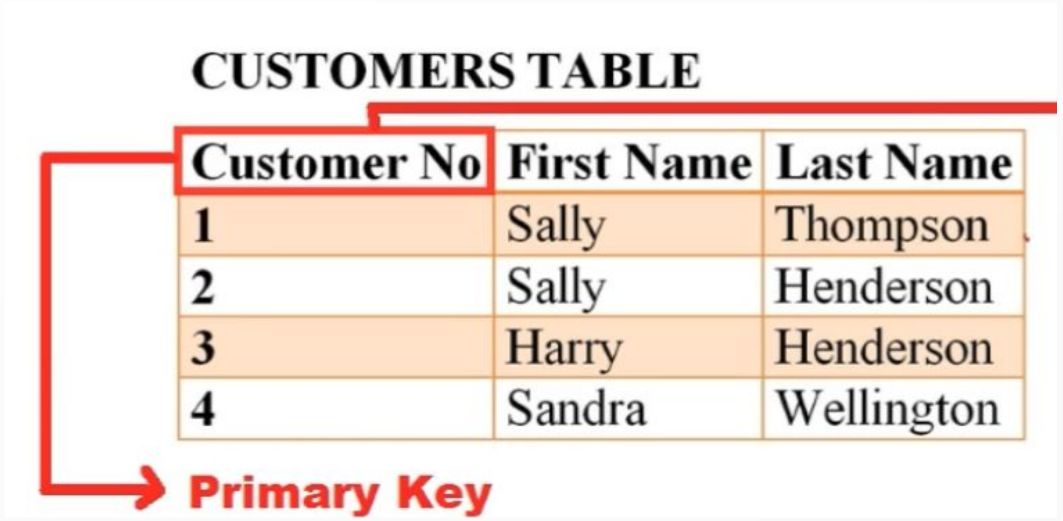
- Database is made up of Tables
  - Tables consist of rows and columns such that:
    - There is no significance to the order of the columns or rows.
    - Each row contains one and only one value for each column.
    - Each value for a given column has the same type.
      - Caveat: blob field equivalent of void* in most DBMS
    - Each table in the database should hold information about one specific thing, such as employees, products, or customers.
    - Each row should hold a unique value
      - At least one column value (or combo) for each row should be unique



| emp_ID | emp_first_name | emp_last_name | emp_phone |
|---|---|---|---|
| 10057 | Barbara | Ku | 1096 |
| 10693 | Jessica | Anne | 7821 |

# RDMS: Primary Key

- Every table has a ‘primary key’
  - The value in every row (column or columns) that allows the row to be uniquely identified
  - Eg:



CUSTOMERS TABLE

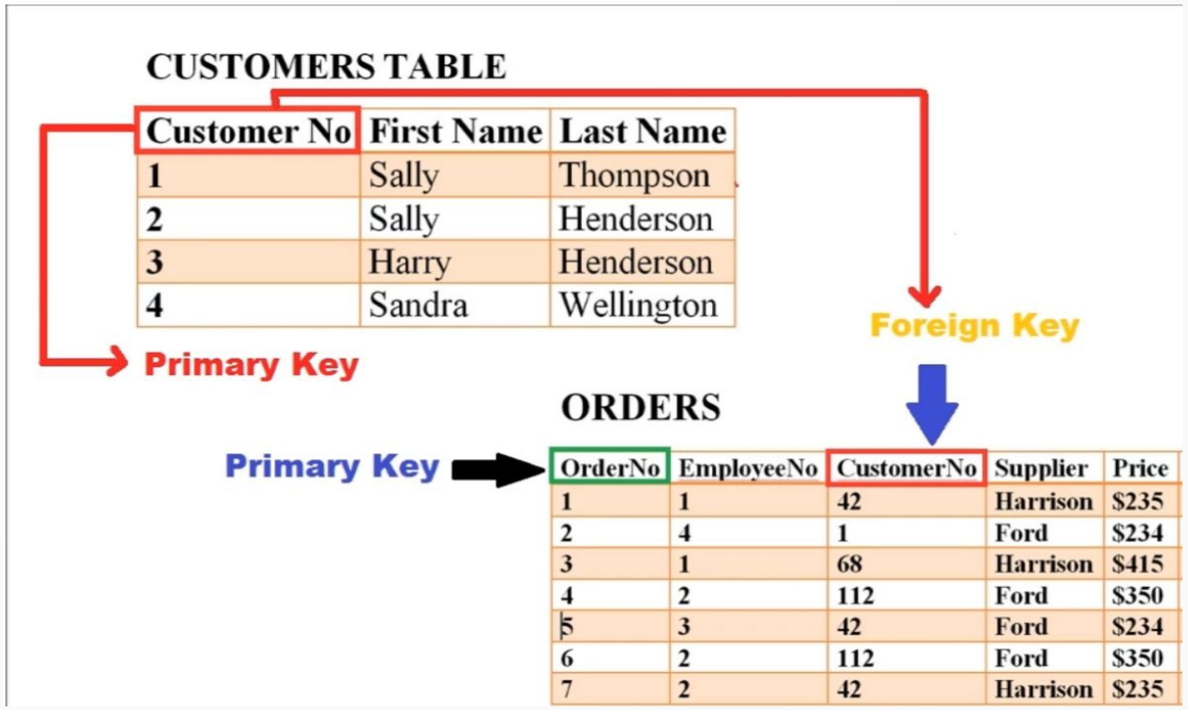
| Customer No | First Name | Last Name |
|---|---|---|
| 1 | Sally | Thompson |
| 2 | Sally | Henderson |
| 3 | Harry | Henderson |
| 4 | Sandra | Wellington |

Primary Key

- Usually numeric,
  - But doesn’t have to be

# RDMS: Foreign Key

- Foreign Keys
  - Every table can have one or more foreign keys
  - Column in table B has the value of the primary key from Table A
    - Links the records/rows in tables A and B



CUSTOMERS TABLE

| Customer No | First Name | Last Name |
|---|---|---|
| 1 | Sally | Thompson |
| 2 | Sally | Henderson |
| 3 | Harry | Henderson |
| 4 | Sandra | Wellington |

ORDERS

| OrderNo | EmployeeNo | CustomerNo | Supplier | Price |
|---|---|---|---|---|
| 1 | 1 | 42 | Harrison | $235 |
| 2 | 4 | 1 | Ford | $234 |
| 3 | 1 | 68 | Harrison | $415 |
| 4 | 2 | 112 | Ford | $350 |
| 5 | 3 | 42 | Ford | $234 |
| 6 | 2 | 112 | Ford | $350 |
| 7 | 2 | 42 | Harrison | $235 |

## How do we represent the data

- Designing a database is a science and art of its own
  - What should be rows, what should be columns
  - Which things should be tables
    - One-one, one-many, many-many relationships among data in tables
    - Database managers have to make these decisions based on technological and business concerns
      - And these can change over time like programming
      - But harder to refactor the database.
  - Making these decisions are beyond the scope of this course
  - I'll assume any significant database is built for you.

## SQL

- When we interact with a Relational Databas SQL is the only game in town.
  - The full official standard is currently  ISO 9075
  - 14 documents from the ISO each 80+ pages and several cost a fair bit of money
    - Giant, in places somewhat contradictory
    - No one implements the whole thing
    - With stored procedures is Turing complete.
      - Turing complete?

## Quick Digression for project

- How do we put text on the screen with our Game Library?

# init in go

- The init function is special in go
  - It is called whenever a package is loaded
    - Either the main package or another package
    - Is called by the go system before any other function in the package
      - Including the main function
    - Is called only once per program (even if the package it is in is loaded multiple times)
  - Used to setup resources that need to exist before any other code is called
    - In this case our fonts need to be loaded in init.

# Font loading

```
func init() {
        tt, err := opentype.Parse(fonts.MPlus1pRegular_ttf)
        if err != nil {
        log.Fatal(err)
        }
        const dpi = 72
        mplusNormalFont, err = opentype.NewFace(tt, &opentype.FaceOptions{
        Size:    24,
        DPI:     dpi,
        Hinting: font.HintingFull,
        })
        if err != nil {
        log.Fatal(err)
        }
        mplusBigFont, err = opentype.NewFace(tt, &opentype.FaceOptions{
        Size:    48,
        DPI:     dpi,
        Hinting: font.HintingFull,
        })
        if err != nil {
        log.Fatal(err)
        }
}
```

From the Ebiten font demo : https://ebiten.org/examples/font.html

# sqlite

- There are many sql/RDMS databases out there
  - For my examples we'll use sqlite
    - Very small lightweight RDMS database
  - To use with go, most people suggest the go-sqlite3 driver by user mattn
  - If not using go modules then
  - go get github.com/mattn/go-sqlite3

# Getting SQLite3

- Any Debian based linux like Ubuntu/kubuntu/Mint etc
  - sudo apt install sqlite3
  - sudo apt install sqlitebrowser – graphical tool for looking at the data in the database
- And you are Done

- There are installers that you can download from the official site for the sqlite browser
- https://sqlitebrowser.org/

# Simple db access

```
import(
	"database/sql"
	_ "github.com/mattn/go-sqlite3" //import for side effects
	"log"
)

func main() {
	myDatabase := OpenDataBase("./Demo.db")
	defer myDatabase.Close()
	create_tables(myDatabase)
}

func OpenDataBase(dbfile string) *sql.DB{
	database, err := sql.Open("sqlite3", dbfile)
	if err != nil {
		log.Fatal(err)
	}
	return database
}
```

Note Unlike in some versions of the library, just opening the database won’t create it new.

- Create table statement used to create a new table in the db
- Official sqlite docs:
  - CREATE TABLE [IF NOT EXISTS] [schema_name].table_name (
  - column_1 data_type PRIMARY KEY,
  - column_2 data_type NOT NULL,
  - column_3 data_type DEFAULT 0,
  - table_constraint
  - );
- Notice that SQL statements end in ‘;’

# Lets add a table

```
func create_tables(database *sql.DB){
	createStatement1 := "CREATE TABLE IF NOT EXISTS students(    " +
		"banner_id INTEGER PRIMARY KEY," +
		"first_name TEXT NOT NULL," +
		"last_name TEXT NOT NULL," +
		"gpa REAL DEFAULT 0," +
		"credits INTEGER DEFAULT 0);"
	database.Exec(createStatement1)
}
```

Lets add another table together. And since we have several SQL novices, lets examine what is going on here

```
CREATE TABLE IF NOT EXISTS course(
    course_prefix TEXT NOT NULL,
    course_number INTEGER NOT NULL,
    cap INTEGER DEFAULT 20,
    description TEXT,
    PRIMARY KEY(course_prefix, course_number)
```

## Finally the big one

- Lets talk about what it all means
- 
```
CREATE TABLE IF NOT EXISTS class_list(
registration_id INTEGER PRIMARY KEY,
course_prefix TEXT NOT NULL,
course_number INTEGER NOT NULL,
student_id INTEGER NOT NULL,
registration_date TEXT,
FOREIGN KEY (student_id) REFERENCES student (banner_id)
ON DELETE CASCADE ON UPDATE NO ACTION,
FOREIGN KEY (course_prefix, course_number) REFERENCES
courses (course_prefix, course_number)
ON DELETE CASCADE ON UPDATE NO ACTION
```

# Lets add some data

- statement := “INSERT INTO STUDENTS (banner_id, first_name, last_name, gpa, credits)
  VALUES (%d, %s, %s, %f, %d)”
- for firstName, lastName := range sampleNames{
-         randGPA := rand.Float32() + float32(rand.Intn(3))
-         randCredits := rand.Intn(120)
-         filled_statement := fmt.Sprintf(statement, count,
            firstName, lastName, randGPA, randCredits)
-         fmt.Println(filled_statement)
-         prepped_statement, err := database.Prepare(filled_statement)
- What could possibly go wrong?

# Bigger picture

```go
func add_sample_data(database *sql.DB){
	sampleNames := map[string]string{"John":"Santore", "Enping":"Li", "Margaret":"Black",
		"Seikyung":"Jung", "Haleh":"Khojasteh", "Abdul":"Sattar", "Paul":"Kim", "Yiheng":"Liang"}
	statement := "INSERT INTO STUDENTS (banner_id, first_name, last_name, gpa, credits)" +
		"  VALUES (%d, %s, %s, %f, %d)"
	count := 1001
	for firstName, lastName := range sampleNames{
		randGPA := rand.Float32() + float32(rand.Intn(3))
		randCredits := rand.Intn(120)
		filled_statement := fmt.Sprintf(statement, count, firstName, lastName, randGPA, randCredits)
		fmt.Println(filled_statement)
		prepped_statement, err := database.Prepare(filled_statement)
		if err != nil{
			//cowardly bail out since this is academia
			log.Fatal(err)
		}
		prepped_statement.Exec()
	}
}
```

- This data is hard coded in the functions -but  where might we be getting our data from in ‘real life’

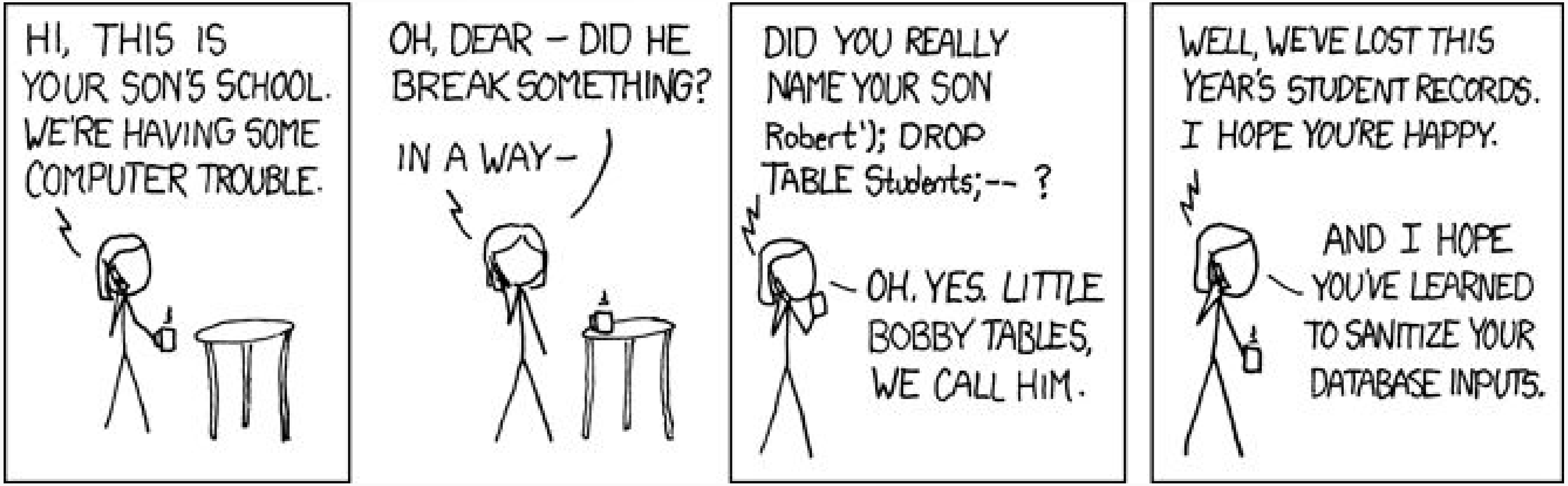
## What could possibly go wrong

- Where does data come from in real life?
  - Often users
  - If we Sprintf data from the users in we could end up with what?

# What could possibly go wrong

- Where does data come from in real life?
  - Often users
  - If we Sprintf data from the users in we could end up with what?
  - “SQL injection attack”
  -

## Little bobby tables anyone


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

# DB security

- App security
  - These days it isn't just for Dr. Li's classes, need to be thinking about secure apps all the time.
  - So when inserting into databases use the Values (?,?,...) notation

```
statement := "INSERT INTO STUDENTS (banner_id, first_name, last_name, gpa, credits)" +
    "  VALUES (?, ?, ?, ?, ?)"
```

  - Now we prepare the statement and execute it.
  - Lets do that now

# For posterity

- On slide in case you need it later

```
func add_sample_data(database *sql.DB){
	sampleNames := map[string]string{"John":"Santore", "Enping":"Li", "Margaret":"Black",
		"Seikyung":"Jung", "Haleh":"Khojasteh", "Abdul":"Sattar", "Paul":"Kim", "Yiheng":"Liang"}
	statement := "INSERT INTO STUDENTS (banner_id, first_name, last_name, gpa, credits)" +
		"  VALUES (?, ?, ?, ?, ?)"
	count := 1001
	for firstName, lastName := range sampleNames{
		randGPA := rand.Float32() + float32(rand.Intn(4))
		randCredits := rand.Intn(120)
		prepped_statement, err := database.Prepare(statement)
		if err != nil{
			//cowardly bail out since this is academia
			log.Fatal(err)
		}
		prepped_statement.Exec(count, firstName, lastName,randGPA,randCredits)
		count +=1
	}
}
```

## Adding courses

- Now we need some courses.
- I’m going to grab my sample data map from an old demo:
- https://github.com/jsantore/GioDemo/blob/master/DisplayData.go
- Our insert statement:

```
insert_statement := "INSERT INTO COURSES (course_prefix, course_number, description) VALUES (?,?,?);"
```

- What do we need next?

# Adding courses

- Now we need some courses.
- I’m going to grab my sample data map from old demo:
- https://github.com/jsantore/GioDemo/blob/master/DisplayData.go
- Our insert statement:

```
insert_statement := "INSERT INTO COURSE (course_prefix, course_number, description) VALUES (?,?,?);"
```

- What do we need next?
  - Range through the data,
  - Break course into number and prefix

# Prepare the data

- How are we going to break the course into a prefix and a course number?

## Prepare the data

- How are we going to break the course into a prefix and a course number?
  - Python style slicing is available
  - So we can slice out “comp” and “510”
  - But what do we need to do now?

# Prepare the data

- How are we going to break the course into a prefix and a course number?
  - Python style slicing is available
  - So we can slice out “comp” and “510”
  - But what do we need to do now?
    - Now we need to convert “510” to 510 right? (lets look at the table description)
    -

# Lets add this sample data

- Work through it with the students
  - Now we have data – look at it in the database viewer

## Putting the R in RDMS

- So far none of our data is related to any other
  - Lets put a little data in the last table and fix that
  - Our statement

  -
    ```
    INSERT INTO CLASS_LIST (banner_id, course_prefix, course_number,
    registration_date)
        VALUES(1001, 'Comp', 510, DATE('now'))
    ```
  - Lets run this for several banner_id values
  - Lets have everyone register for comp510 – because of course
  - But setup replaceable values for the bannerID
    - Tell me how

## Now we have data

- Now we have data for all of the tables, but lets look at that last one

# Dates

- Some RDMS have a datetime type for fields
- Not sqlite
  - Dates can be stored as Strings
  - Dates can be stored as floats/Reals (use julianday function)
  - Dates can be stored as ints (unix time epoch)
- Two builtin functions you want to think about (for string version)
  - DATE
  - DATETIME
  - Both take params, most common value is ‘now’

# Oops I made a mistake

- Sometimes you make a mistake
  - Or you operate in Europe under the GDPR
  - GDPR???
    - So many lucky volunteers

## Oops I made a mistake

- Sometimes you make a mistake
  - Or you operate in Europe under the GDPR
  - And you need to delete data
  - DELETE Statement
    - Simple example: DELETE FROM table WHERE search_condition;

# We need to make a change

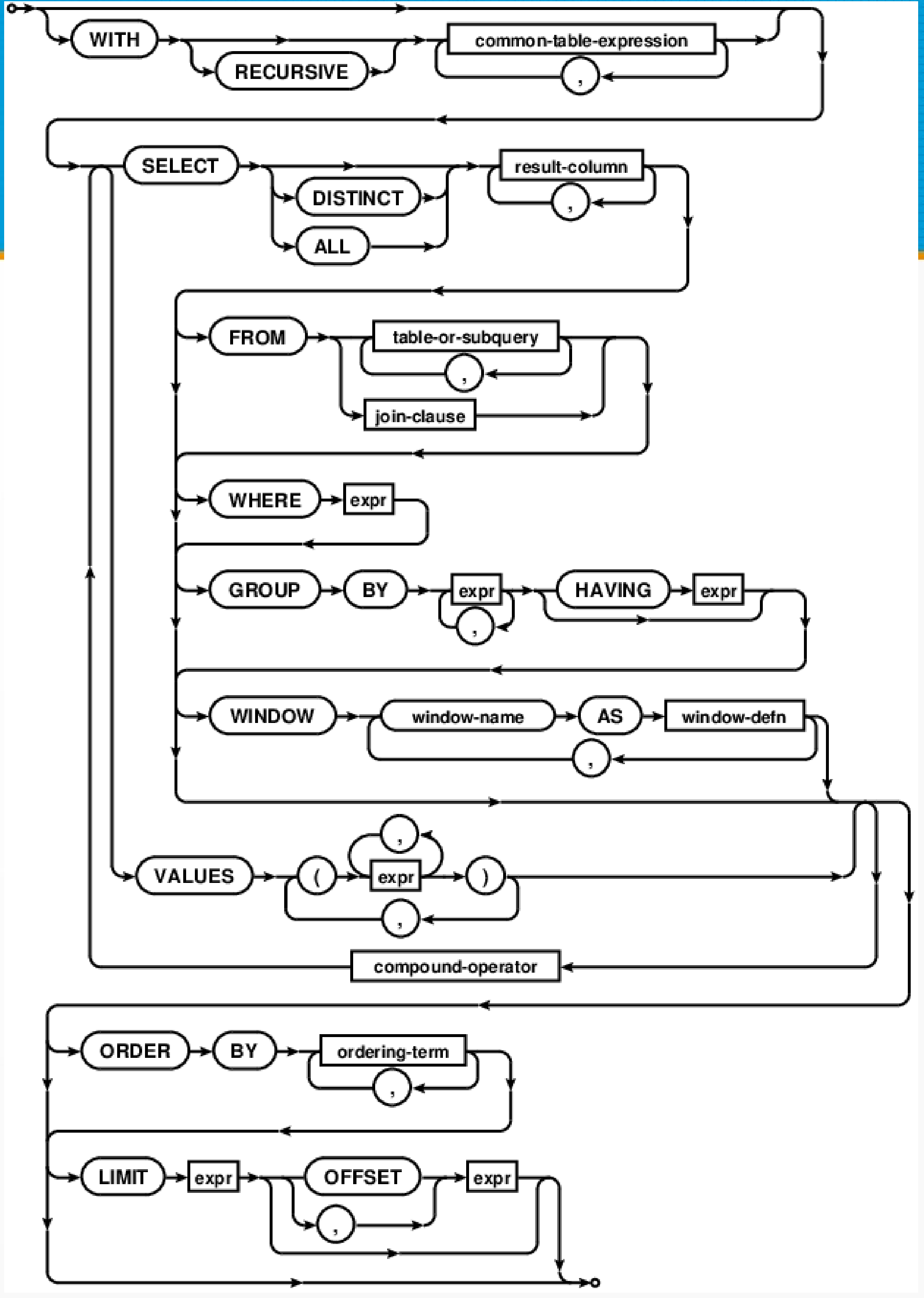
- If you need to make a change to the data use update
  - UPDATE data:
    - UPDATE table
    - SET column_1 = new_value_1,
    - column_2 = new_value_2
    - WHERE
    - search_condition
    - ORDER column_or_expression
    - LIMIT row_count OFFSET offset;

# More data maintenance

- Alter table
  - Can add or delete or rename columns and more
  - Fairly complex syntax, take comp580
- Drop table
  - Remove table (and all its data) from the database (schema)
- Here I'm giving you a taste of what is available
  - "Unknown unknowns" to "known unknowns"
  - And energetic grad students will fill in those "known unknowns"

# Using Data

- Using data in an RDMS
- Aka welcome to select
  - Even in small sqlite the syntax is complex
  - This is the official graph for the finite state machine for select



## Select at its most basic

- The simplest version of Select
  - SELECT <columns> FROM <table>
  - Or more interesting:
    - SELECT <columns> FROM <table> WHERE <column name> = <value>
    -

## Lets try

- Lets find all the students on probation
  - First ask the user what the probation cutoff is
    - (eg 2.0 for undergrads and 3.0 for grads etc.)
  - Guide me through this

## My solution

- For those who lose theirs and for later

```go
func getMinGPA() float64{
	reader := bufio.NewReader(os.Stdin)
	fmt.Println("What is the minimum grade for good standing?")
	value, err:= reader.ReadString("\n")
	if err != nil{
		log.Fatal("How did we fail reading standard in??",err)
	}
	min_gpa, err := strconv.ParseFloat(value, 32)
	if err != nil{
		log.Fatal("OK you seem to have typed in something that wasn't a float", err)
	}
	return min_gpa
}
```

# Now query

- Now lets make the query and use the data

```
• func findProbationStudents(database *sql.DB) {
    – var firstName, lastName string
    – var gpa float64
    – minGpa := getMinGPA()
    – selectStatement := "SELECT first_name, last_name, gpa FROM STUDENTS WHERE gpa < ?"
    – resultSet, err := database.Query(selectStatement, minGpa)
    – if err != nil {
        • log.Fatal("Bad Query", err)
    – }
    – defer resultSet.Close()
    – for resultSet.Next() {
    – err = resultSet.Scan(&firstName, &lastName, &gpa)
    – if err != nil {
        • log.Fatal(err)
    – }
    – fmt.Printf("%s %s is on probation with a GPA of %f\n", firstName, lastName, gpa)
    – }}
```

# Joins and select

- More interesting data is found by joining two tables
  - Terminology
    - Inner join
    - Outer join
    - See comp580
  - But rule of thumb: only join tables that have foreign keys, otherwise you will have giant mess on your hands

# Try it

- If we have time lets try this

-
  ```
  SELECT first_name, last_name, credits
      FROM STUDENTS
      INNER JOIN CLASS_LIST ON
      STUDENTS.banner_id = CLASS_LIST.banner_id
      WHERE (STUDENTS.credits < 10
          and CLASS_LIST.course_prefix = 'Comp'
          and CLASS_LIST.course_number = 510)
  ```
-
- Note: inner join on foreign key
- Where clause columns need not be in result set.

- Relational Databases are useful and important
  - Take comp580 or comp405
  - Learn more on your own
  - This finishes our quick and useful look at relational databases
  - Hopefully many “unknown unknowns → “known unknowns”