THE GEORGE WASHINGTON UNIVERSITY

WASHINGTON, DC

# 2. Relational Model

## CSCI 2541 Database Systems & Team Projects

Wood & Chaufournier

Slides adapted from Prof. Bhagi Narahari; Silberschatz, Korth, and Sudarshan; and Ramakrishnan, Gerhke, and Lawrence

# Admin Stuff

HW 1 due yesterday

- Test cases should give you idea of your grade

Optional RPS HW due Jan 23

Read our materials/instructions carefully

Read the syllabus for course policies

Watch Slack for ways to **#engage** and ask **#questions**

# Last week…

**Structure** that is **independent** of the underlying file formats

**Queries** to flexibly read, update, and delete information

**Transactions** that provide guarantees about **multi-user consistency**

Relational Model Definitions

Constraints and Relationships

Lab!

…this week.

# Data

Let's store some information about professors

- How?

# Tables

A Table is a set of rows and columns…

- A column defines an attribute that can have different values
- A row represents related attributes that together represent a data element

Instructor Table

| *ID* | *name* | *dept_name* | *salary* |
|---|---|---|---|
| 10101 | Srinivasan | Comp. Sci. | 65000 |
| 12121 | Wu | Finance | 90000 |
| 15151 | Mozart | Music | 40000 |
| 22222 | Einstein | Physics | 95000 |
| 32343 | El Said | History | 60000 |
| 33456 | Gold | Physics | 87000 |
| 45565 | Katz | Comp. Sci. | 75000 |
| 58583 | Califieri | History | 62000 |
| 76543 | Singh | Finance | 80000 |
| 76766 | Crick | Biology | 72000 |
| 83821 | Brandt | Comp. Sci. | 92000 |
| 98345 | Kim | Elec. Eng. | 80000 |

Course Table

| *course_id* | *title* | *dept_name* | *credits* |
|---|---|---|---|
| BIO-101 | Intro. to Biology | Biology | 4 |
| BIO-301 | Genetics | Biology | 4 |
| BIO-399 | Computational Biology | Biology | 3 |
| CS-101 | Intro. to Computer Science | Comp. Sci. | 4 |
| CS-190 | Game Design | Comp. Sci. | 4 |
| CS-315 | Robotics | Comp. Sci. | 3 |
| CS-319 | Image Processing | Comp. Sci. | 3 |
| CS-347 | Database System Concepts | Comp. Sci. | 3 |
| EE-181 | Intro. to Digital Systems | Elec. Eng. | 3 |
| FIN-201 | Investment Banking | Finance | 3 |
| HIS-351 | World History | History | 3 |
| MU-199 | Music Video Production | Music | 3 |
| PHY-101 | Physical Principles | Physics | 4 |

# Tables = Relations

A **Relation** is a **set** of **tuples** and **attributes**

- Set: an unordered list of unique elements
- Tuple: a sequence of values
- Attribute: a named type with values in a domain

Why?

Instructor Relation

| *ID* | *name* | *dept_name* | *salary* |
|---|---|---|---|
| 10101 | Srinivasan | Comp. Sci. | 65000 |
| 12121 | Wu | Finance | 90000 |
| 15151 | Mozart | Music | 40000 |
| 22222 | Einstein | Physics | 95000 |
| 32343 | El Said | History | 60000 |
| 33456 | Gold | Physics | 87000 |
| 45565 | Katz | Comp. Sci. | 75000 |
| 58583 | Califieri | History | 62000 |
| 76543 | Singh | Finance | 80000 |
| 76766 | Crick | Biology | 72000 |
| 83821 | Brandt | Comp. Sci. | 92000 |
| 98345 | Kim | Elec. Eng. | 80000 |

Course Relation

| *course_id* | *title* | *dept_name* | *credits* |
|---|---|---|---|
| BIO-101 | Intro. to Biology | Biology | 4 |
| BIO-301 | Genetics | Biology | 4 |
| BIO-399 | Computational Biology | Biology | 3 |
| CS-101 | Intro. to Computer Science | Comp. Sci. | 4 |
| CS-190 | Game Design | Comp. Sci. | 4 |
| CS-315 | Robotics | Comp. Sci. | 3 |
| CS-319 | Image Processing | Comp. Sci. | 3 |
| CS-347 | Database System Concepts | Comp. Sci. | 3 |
| EE-181 | Intro. to Digital Systems | Elec. Eng. | 3 |
| FIN-201 | Investment Banking | Finance | 3 |
| HIS-351 | World History | History | 3 |
| MU-199 | Music Video Production | Music | 3 |
| PHY-101 | Physical Principles | Physics | 4 |

# Schema

Defines the structure of one or more Relations

- **A1, A2, …, An** are attributes
- **R = (A1, A2, …, An )** is a relation schema

Example: **instructor = (ID, name, dept_name, salary)**

- A relation instance **r** defined over schema **R** is denoted by **r (R)**.
- The current values of a relation are specified by a table
- An element **t** of relation **r** is called a tuple and is represented by a row in a table

# Example DB Schema

**STUDENT**

| Name | Student_number | Class | Major |
|---|---|---|---|

**COURSE**

| Course_name | Course_number | Credit_hours | Department |
|---|---|---|---|

**PREREQUISITE**

| Course_number | Prerequisite_number |
|---|---|

**SECTION**

| Section_identifier | Course_number | Semester | Year | Instructor |
|---|---|---|---|---|

**GRADE_REPORT**

| Student_number | Section_identifier | Grade |
|---|---|---|

# Relational Model Definitions

A **relation** is a table with columns and rows.

An **attribute** is a named column of a relation.

A **tuple** is a row of a relation.

A **domain** is a set of allowable values for one or more attributes.

The **degree** of a relation is the number of attributes it contains.

The **cardinality** of a relation is the number of tuples it contains.

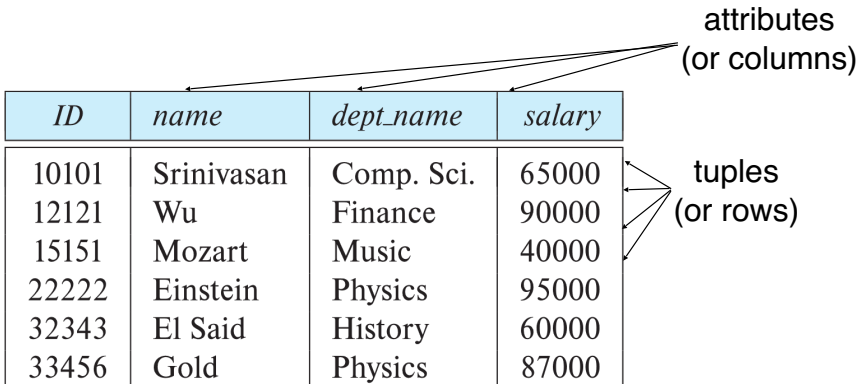
A **relational database** is a collection of normalized relations with distinct relation names.

# Definitions

Degree =

Cardinality =

Help me fill these in!




| *ID* | *name* | *dept_name* | *salary* |
|---|---|---|---|
| 10101 | Srinivasan | Comp. Sci. | 65000 |
| 12121 | Wu | Finance | 90000 |
| 15151 | Mozart | Music | 40000 |
| 22222 | Einstein | Physics | 95000 |
| 32343 | El Said | History | 60000 |
| 33456 | Gold | Physics | 87000 |

# Relation Property Summary

1. Each relation name is unique
   - No two relations have the same name

2. Each cell of the relation (value of a domain) contains exactly one atomic (single) value and cannot be empty… in practice SQL allows NULL

3. Each attribute of a relation has a distinct name

4. Values of an attribute are all from the same domain

5. Each tuple is distinct. There are no duplicate tuples
   - Theoretically… in practice, SQL supports "bags" (allow duplicates)

6. Order of attributes is not important
   - Note difference from mathematical def of relations
     - Tuple (x,y) is not the same as (y,x) in mathematical definition
   - Reason: attribute names represent domain and can be reordered

7. Order of tuples is not important

Relational Model Definitions

**Constraints and Relationships**

Lab!

onwards…

# Constraints

Relation scheme defines the types and domain of all attributes

- Can enforce constraints whenever tuples are added/modified

This can enforce many constraints to protect the integrity of your data

- Can't insert a string into an Integer type attribute
- A State field could limit domain to (AL, AK, AZ…WY)
- An SSN attribute must follow form (xxx-xx-xxxx)
- Price must be > 0.00

… but not all!

- Application or "business logic" may not be feasible
- Example: "An employee can't work more than 40 hours per week across all jobs"

# Keys

Superkey of R:

- A set of attributes that is sufficient to uniquely identify each tuple in r(R)

| *ID* | *name* | *dept_name* | *salary* |
|---|---|---|---|
| 22222 | Einstein | Physics | 95000 |
| 12121 | Wu | Finance | 90000 |
| 32343 | El Said | History | 60000 |
| 45565 | Katz | Comp. Sci. | 75000 |
| 98345 | Kim | Elec. Eng. | 80000 |
| 76766 | Crick | Biology | 72000 |
| 10101 | Srinivasan | Comp. Sci. | 65000 |
| 58583 | Califieri | History | 62000 |
| 83821 | Brandt | Comp. Sci. | 92000 |
| 15151 | Mozart | Music | 40000 |
| 33456 | Gold | Physics | 87000 |
| 76543 | Singh | Finance | 80000 |

The *professor* relation

What is a superkey for this relation?

# Keys

Superkey of R:

- A set of attributes that is sufficient to uniquely identify each tuple in r(R)

What is a superkey for this relation?

| *course_id* | *sec_id* | *semester* | *year* | *building* | *room_number* | *time_slot_id* |
|---|---|---|---|---|---|---|
| BIO-101 | 1 | Summer | 2017 | Painter | 514 | B |
| BIO-301 | 1 | Summer | 2018 | Painter | 514 | A |
| CS-101 | 1 | Fall | 2017 | Packard | 101 | H |
| CS-101 | 1 | Spring | 2018 | Packard | 101 | F |
| CS-190 | 1 | Spring | 2017 | Taylor | 3128 | E |
| CS-190 | 2 | Spring | 2017 | Taylor | 3128 | A |
| CS-315 | 1 | Spring | 2018 | Watson | 120 | D |
| CS-319 | 1 | Spring | 2018 | Watson | 100 | B |
| CS-319 | 2 | Spring | 2018 | Taylor | 3128 | C |
| CS-347 | 1 | Fall | 2017 | Taylor | 3128 | A |
| EE-181 | 1 | Spring | 2017 | Taylor | 3128 | C |
| FIN-201 | 1 | Spring | 2018 | Packard | 101 | B |
| HIS-351 | 1 | Spring | 2018 | Painter | 514 | C |
| MU-199 | 1 | Spring | 2018 | Packard | 101 | D |
| PHY-101 | 1 | Fall | 2017 | Watson | 100 | A |

The *section* relation

# Candidate and Primary Keys

**Superkey** of R:

- A set of attributes that is sufficient to uniquely identify each tuple in r(R)

**Candidate Key** of R: A “minimal” superkey

- A Candidate Key is a superkey K such that removal of any attribute from K results in a set of attributes that is not a superkey (does not possess the superkey uniqueness property)
- A Candidate Key is a Superkey but opposite may not be true

**Primary Key**: The Candidate Key chosen to represent a relation/table

# Super vs Candidate Key

Possible superkeys:

- <ID, name>,
- <ID, dept_name>,
- <ID, name, dept_name, salary>

Candidate Key must be minimal:

- <ID>
- <course_id, sec_id, semester, year>

Primary keys are listed first and underlined when showing the schema

*classroom*(*building*, *room_number*, *capacity*)
*department*(*dept_name*, *building*, *budget*)
*course*(*course_id*, *title*, *dept_name*, *credits*)
*instructor*(*ID*, *name*, *dept_name*, *salary*)

| *ID* | *name* | *dept_name* | *salary* |
|---|---|---|---|
| 22222 | Einstein | Physics | 95000 |
| 12121 | Wu | Finance | 90000 |
| 32343 | El Said | History | 60000 |
| 45565 | Katz | Comp. Sci. | 75000 |
| 98345 | Kim | Elec. Eng. | 80000 |
| 76766 | Crick | Biology | 72000 |
| 10101 | Srinivasan | Comp. Sci. | 65000 |
| 58583 | Califieri | History | 62000 |
| 83821 | Brandt | Comp. Sci. | 92000 |
| 15151 | Mozart | Music | 40000 |
| 33456 | Gold | Physics | 87000 |
| 76543 | Singh | Finance | 80000 |

| *course_id* | *sec_id* | *semester* | *year* | *building* | *room_number* | *time_slot_id* |
|---|---|---|---|---|---|---|
| BIO-101 | 1 | Summer | 2017 | Painter | 514 | B |
| BIO-301 | 1 | Summer | 2018 | Painter | 514 | A |
| CS-101 | 1 | Fall | 2017 | Packard | 101 | H |
| CS-101 | 1 | Spring | 2018 | Packard | 101 | F |
| CS-190 | 1 | Spring | 2017 | Taylor | 3128 | E |
| CS-190 | 2 | Spring | 2017 | Taylor | 3128 | A |
| CS-315 | 1 | Spring | 2018 | Watson | 120 | D |
| CS-319 | 1 | Spring | 2018 | Watson | 100 | B |
| CS-319 | 2 | Spring | 2018 | Taylor | 3128 | C |
| CS-347 | 1 | Fall | 2017 | Taylor | 3128 | A |
| EE-181 | 1 | Spring | 2017 | Taylor | 3128 | C |
| FIN-201 | 1 | Spring | 2018 | Packard | 101 | B |
| HIS-351 | 1 | Spring | 2018 | Painter | 514 | C |
| MU-199 | 1 | Spring | 2018 | Packard | 101 | D |
| PHY-101 | 1 | Fall | 2017 | Watson | 100 | A |

# Picking a Primary Key

Every Relation must have a Primary Key

How to pick from the candidates?

- Based on business logic
- Is “Name” unique? depends on your business/ application!
- Ideally Primary Key should be something that never/rarely changes

Why?

Primary Key is another type of **constraint**

- DB will enforce uniqueness of the Primary Key attributes

# The magic of Databases

A database helps us **connect** multiple Relations

**STUDENT**

| Name | Student_number | Class | Major |
|---|---|---|---|

**COURSE**

| Course_name | Course_number | Credit_hours | Department |
|---|---|---|---|

**PREREQUISITE**

| Course_number | Prerequisite_number |
|---|---|

**SECTION**

| Section_identifier | Course_number | Semester | Year | Instructor |
|---|---|---|---|---|

**GRADE_REPORT**

| Student_number | Section_identifier | Grade |
|---|---|---|

How are these Relations connected to each other?

# Foreign Keys

Defines a relationship connecting tuples in two relations

- The **referencing relation** and the **referenced relation**
- Defines another type of constraint - **Referential Integrity**
- **Foreign Key constraints** must connect to the Primary Key in the referenced relation

**GRADE_REPORT.Student_number** must match a value in **STUDENT.Student_number**

**PREREQUSITE.Course_number** and **Prequisite_number** must match value in **COURSE.Course_number, etc**

etc

**STUDENT**

| Name | Student_number | Class | Major |
|---|---|---|---|

**COURSE**

| Course_name | Course_number | Credit_hours | Department |
|---|---|---|---|

**PREREQUISITE**

| Course_number | Prerequisite_number |
|---|---|

**SECTION**

| Section_identifier | Course_number | Semester | Year | Instructor |
|---|---|---|---|---|

**GRADE_REPORT**

| Student_number | Section_identifier | Grade |
|---|---|---|

# Referential Integrity

Only students listed in the Students relation should be allowed to enroll for courses.

- If a value of sid appears in Enrollment relation then it MUST appear in Student relation
  - "Only students can take courses"
  - Database is automatically enforcing application requirements for you… can your Array do that?

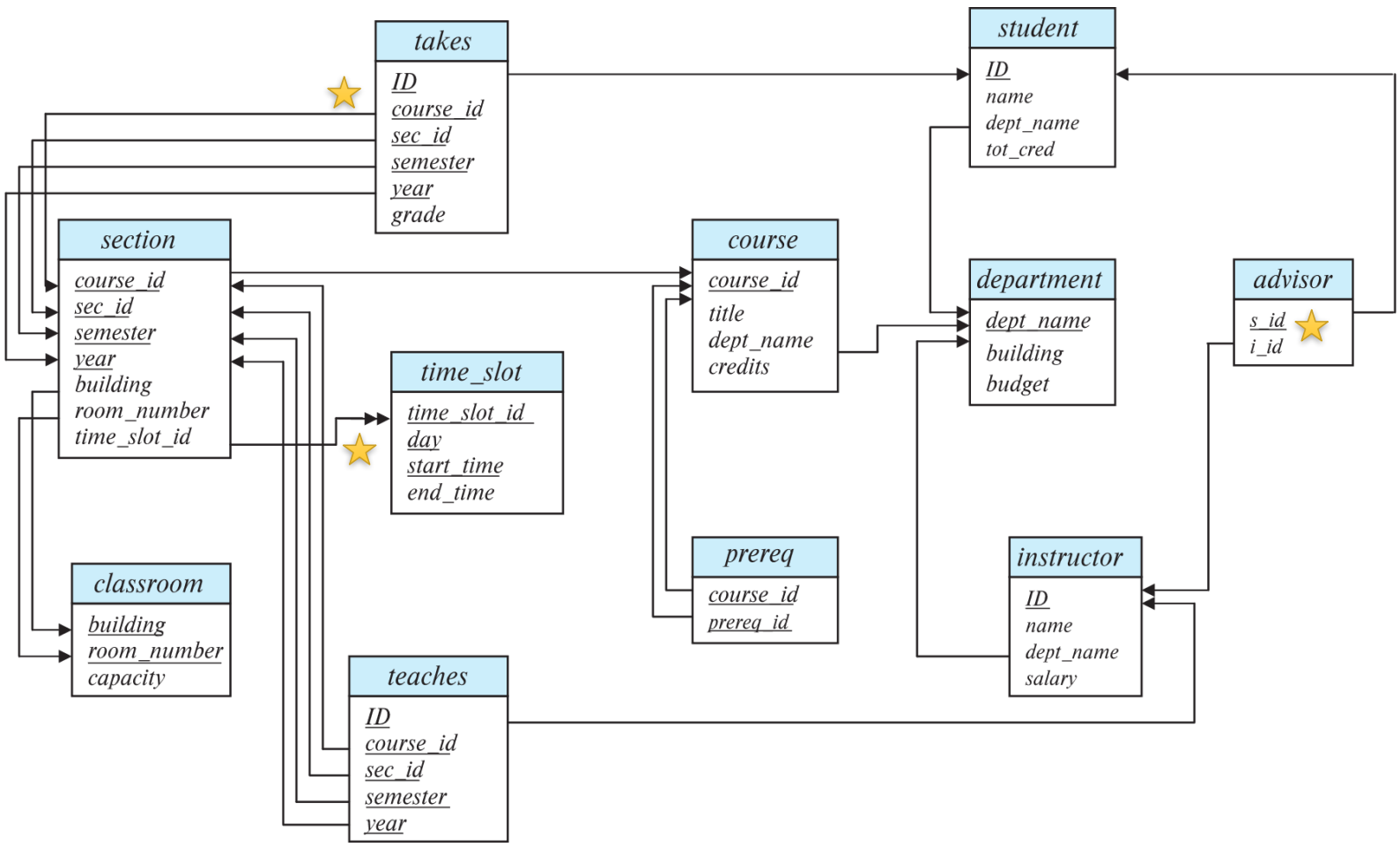
Enrollment

| sid | cid | grade |
|---|---|---|
| 53666 | Jazz101 | C |
| 53666 | Reggae203 | B |
| 53650 | Topology112 | A |
| 53666 | History105 | B |

Student

| sid | name | login | age | gpa |
|---|---|---|---|---|
| 53666 | Jones | jones@cs | 18 | 3.4 |
| 53688 | Smith | smith@eecs | 18 | 3.2 |
| 53650 | Smith | smith@math | 19 | 3.8 |

# Full University Schema Diagram

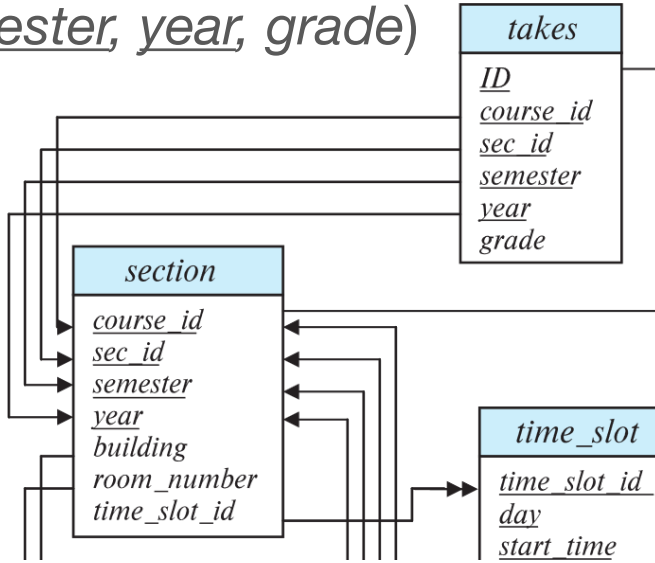


# Primary Key Selection

Why do we use multiple attributes in a Primary Key?

- section(*course_id*, *sec_id*, *semester*, *year*, *building*, …)
- takes(*ID*, *course_id*, *sec_id*, *semester*, *year*, *grade*)

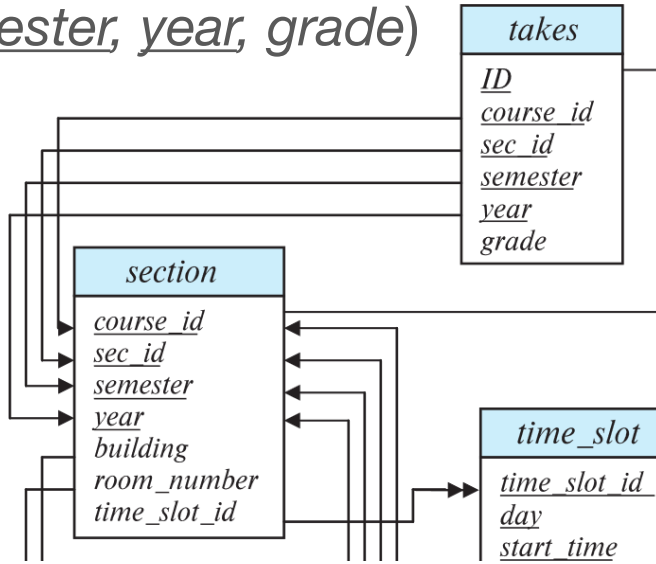
Why not just define a unique attribute like ***sec_id_number*** and use that by itself?



# Primary Key Selection

Why do we use multiple attributes in a Primary Key?

- section(*course_id*, *sec_id*, *semester*, *year*, *building*, …)
- takes(*ID*, *course_id*, *sec_id*, *semester*, *year*, *grade*)



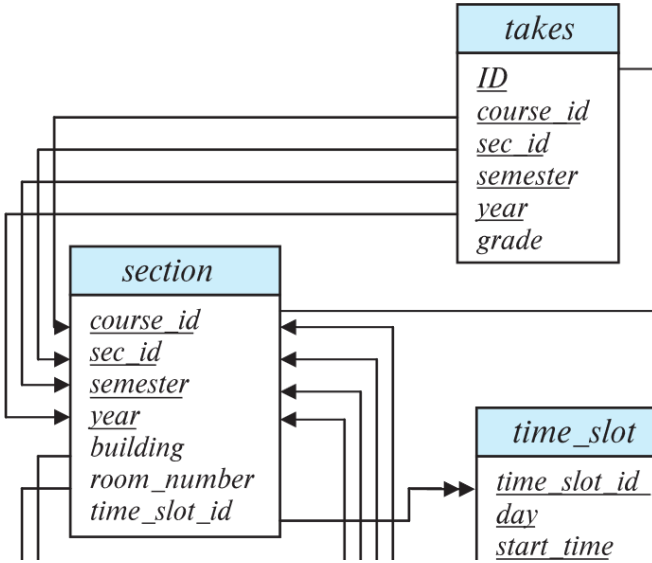

- Using a single field looks simpler, but it prevents the benefit of the DB enforcing uniqueness
- Using sec_id_number as foreign key requires us to look up info from multiple tables which may be less efficient

# Primary Key Selection

Consider this:

- takes(*ID*, *course_id*, *sec_id*, *semester*, *year*, *grade*)

Does this match the "business logic" we actually want for our university? (Hint: what uniqueness will this enforce?)

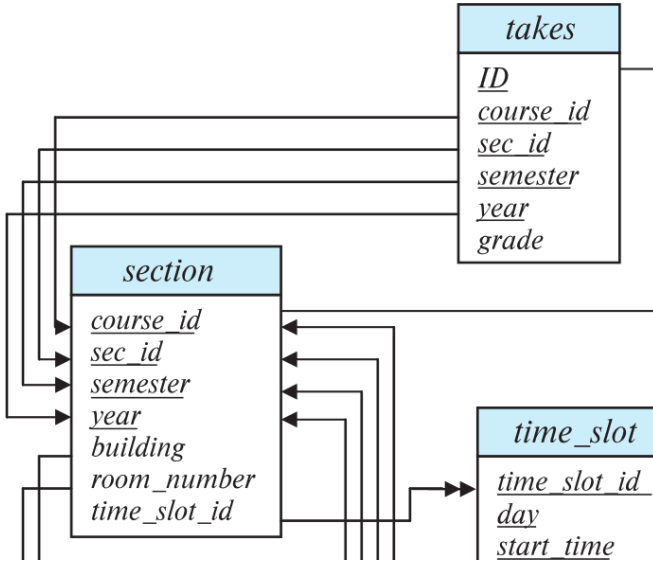


# Primary Key Selection

Consider this:

- takes(*ID*, *course_id*, *sec_id*, *semester*, *year*, *grade*)

This Primary Key allows a student to be registered for multiple sections of the same course at once!




But if we remove *sec_id*, then we will not have a complete Foreign Key!

- We must match all fields in the other relation's PK to qualify as a Foreign Key
- In practice, many SQL DBs don't support Referential Integrity without a complete Foreign Key

Relational Model Definitions

Constraints and Relationships

**Lab!**

onwards…

# Lab 2: Python Flask

**GW CS 2541: Database Systems and Team Projects - 2022**
Prof. Tim Wood, Ethan Baron, and Catherine Meadows

# **Front End vs Backend**: What is the difference?

# Front End vs Back End

Front End:

- The art and design of websites and web applications that render on the client side. Everything from the look and feel to the way you interact with a website.

Back End:

- The server side logic for an application controlling what happens with the data, how the client side rendering changes in response and how the data gets stored.

Full Stack:

- A developer that can work on both Front and Back End software

# Python Flask

Flask is a Python “Web framework”

- Library to make it easier to write a web application
- Examples: Flask, Django, react.js, vue.js, Angular, Ruby on Rails, Drupal…

How is this different from a “web server”?

API for defining backend services

- Handles form input/output, cookie management, session data, DB connections, overall page formatting, etc

Flask is a microframework

- Provides a minimal set of functionality (with extras as needed)

# Flask Hello World

Load the Flask library and setup your app

Define a **Route** and specify what is returned when we access it

Tell the web application to actually start running

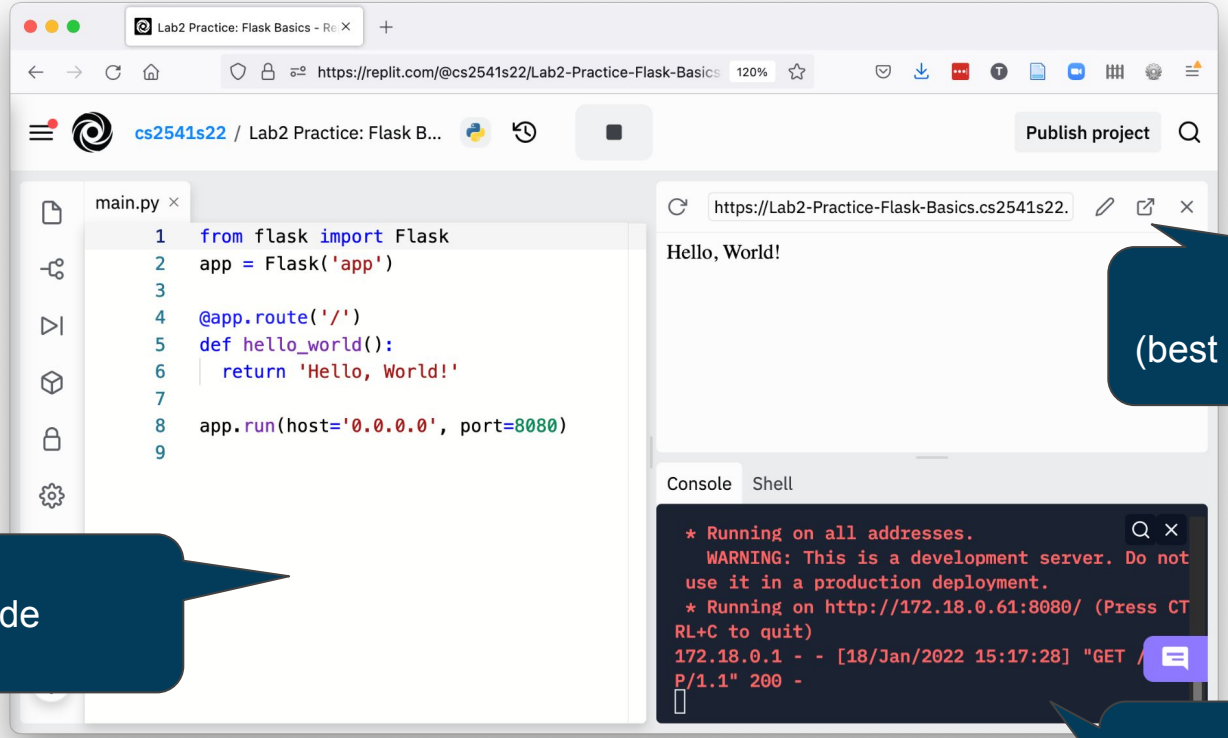
```
from flask import Flask
app = Flask('app')


@app.route('/')
def hello_world():
 return 'Hello, World!'


app.run(host='0.0.0.0', port=8080)
```

Be careful about copy/paste from slides!

# Flask on Replit.com




Write code

App preview
(best to open in new tab)

Log messages - print() calls will go here

# Routes + Templates = Flask

Routes

- A backend service endpoint URL
- Function to be called when route is accessed

Templates

- Defines front end appearance of website
- Interacts with back end to allow data to be filled in

Both are good examples of our goal of Abstraction!

- Flask helps us separate design of different backend services and cleanly separates front end from back end

# Routes Basics

Routes are functions:

- Specify the URL to access them
- Define the behavior to execute
- Return the content that should be displayed to the user

`@app.route` is a Python Decorator

- Special syntax to make a wrapper function. See [1] for details

```
from flask import Flask
app = Flask('app')

@app.route('/two')
def hello_world():
 Return '<html><body>This route has valid
HTML, but both will display!</body></html>'

@app.route('/')
def hello_world():
 return 'Hello, World from the root route!'

app.run(host='0.0.0.0', port=8080)
```

[1] https://realpython.com/primer-on-python-decorators/

# Templating Basics: Passing Variables

/templates/index.html

Templates MUST be in a folder /templates

```
<html>
    <head>
        <title> {{ title }} </title>
    </head>
    <body>
        <h1> Hello {{ username }}</h1>
    <body>
</html>
```

Import module for rendering templates

```
from flask import Flask
from flask import render_template
app = Flask('app')

@app.route('/index')
def index():
    name = 'Cat Meadows'
    return render_template('index.html',
        title = 'Welcome',username=name)

app.run(host='0.0.0.0', port=8080)
```

# Demo: Hello X and Hello Y

Reuse:

- Template can be used in multiple routes
- Each route can fill different data into the template

```
@app.route('/')
def hello_world():
 name = 'Cat Meadows'
 return render_template('hello.html', title =
'Welcome', username=name)

@app.route('/helloTim')
def hello_world():
 name = 'Tim Wood'
 return render_template('hello.html', title = 'Welcome
2', username=name)
```

Demo:
https://replit.com/team/cs2541s22/Lab2-Practice-Template-Data

# Routes with Parameters

We can extract data from the URL

- Parameters are available as python variables
- Flask lets you enforce types, have multiple parameters, etc

Modify the route definition and add the parameters as arguments to your function

What would we see if we visit /parameters/Ethan ?

```
from flask import Flask
app = Flask('app')

@app.route('/parameters/<name>')
def hello_name(name):
 return 'Hello, ' + name

app.run(host='0.0.0.0', port=8080)
```

# Activity 1: Hello/XYZ

5 minutes!

Our earlier code was dumb!

- Repetitive routes that are just different based on the incoming data

Group Task:

- Make a single route which can say "hello **XYZ**" based on the URL data
- Must use template

```
@app.route('/')
def hello_world():
 name = 'Cat Meadows'
 return render_template('hello.html', title =
'Welcome', username=name)

@app.route('/helloTim')
def hello_world():
 name = 'Tim Wood'
 return render_template('hello.html', title = 'Welcome
2', username=name)
```

/hello/Tim -> "Hello Tim!"
/hello/Ethan -> "Hello Ethan!"
etc...

**https://replit.com/team/cs2541s22/Lab2-Practice-Template-Data**

# Templating Basics: if

```
<html>
    <head>
        <title> {{ title }} </title>
    </head>
    <body>
        {% if username == "Cat Meadows": %}
        <h1> Hello, UTA!! </h1>
        {% else %}
        <h1> Hello, {{ username }} </h1>
        {% endif %}
    <body>
</html>
```

Syntax {% … %}

Always close conditionals with **{% endif %}**

# Templating Basics: for loop

```
<html>
    <head>
        <title> {{ title }} </title>
    </head>
    <body>
        <ul>
        {% for user in users: %}
        <li> {{ user }} </li>
        {% endfor %}
        </ul>
    <body>
</html>
```

Always close for loops with **{% endfor %}**

```
from flask import Flask
from flask import render_template
app = Flask('app')

@app.route('/index')
def index():
    users = ['Tim', 'Ethan', 'Cat']
    return render_template('index.html',
        title = 'Welcome',users=users)

app.run(host='0.0.0.0', port=8080)
```

# More Template Syntax

Demo: "Lab-2-Practice-Template-Syntax" replit

Learn more:

- https://realpython.com/primer-on-jinja-templating/
- https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-ii-templates

# Activity 2: Class Roster

1. Create a class roster **nested dictionary** in the format:

```
Tim = {
      "Name": "Tim Wood"
      "SID": "G12345678"
      "Engagement": 2
 }
 #... define more students here …
roster = {
      "Tim" : Tim,
      "Ethan" : Ethan,
      "Cat" : Cat
 }
```

2. Use templates and routes with parameters to display:
   - First route: list of students by name at the "/" index route
     - Each name should be a link to the second route
   - Second route: display name, SID, and engagement points for the student

Note: You should only use **two** templates and **two** routes

**Due TOMORROW 11:59PM**

https://replit.com/team/cs2541s22/Lab2-Advanced-Routes-and-Templates

# Formatting Examples

Emphasis box 1

Emphasis box 2

Emphasis box 3

Emphasis box 4

```
from flask import Flask
app = Flask('app')


@app.route('/')
def hello_world():
 return 'Hello, World!'

app.run(host='0.0.0.0', port=8080)
```