

# Databases, SQL and MySQL

# Transient vs. Persistent data

- Program data in variable is transient
- When the program ends, data is lost
- If we rerun the program, the data will need to be collected or regenerated

# Databases

Organized collections of data  
(that reside on a computer)

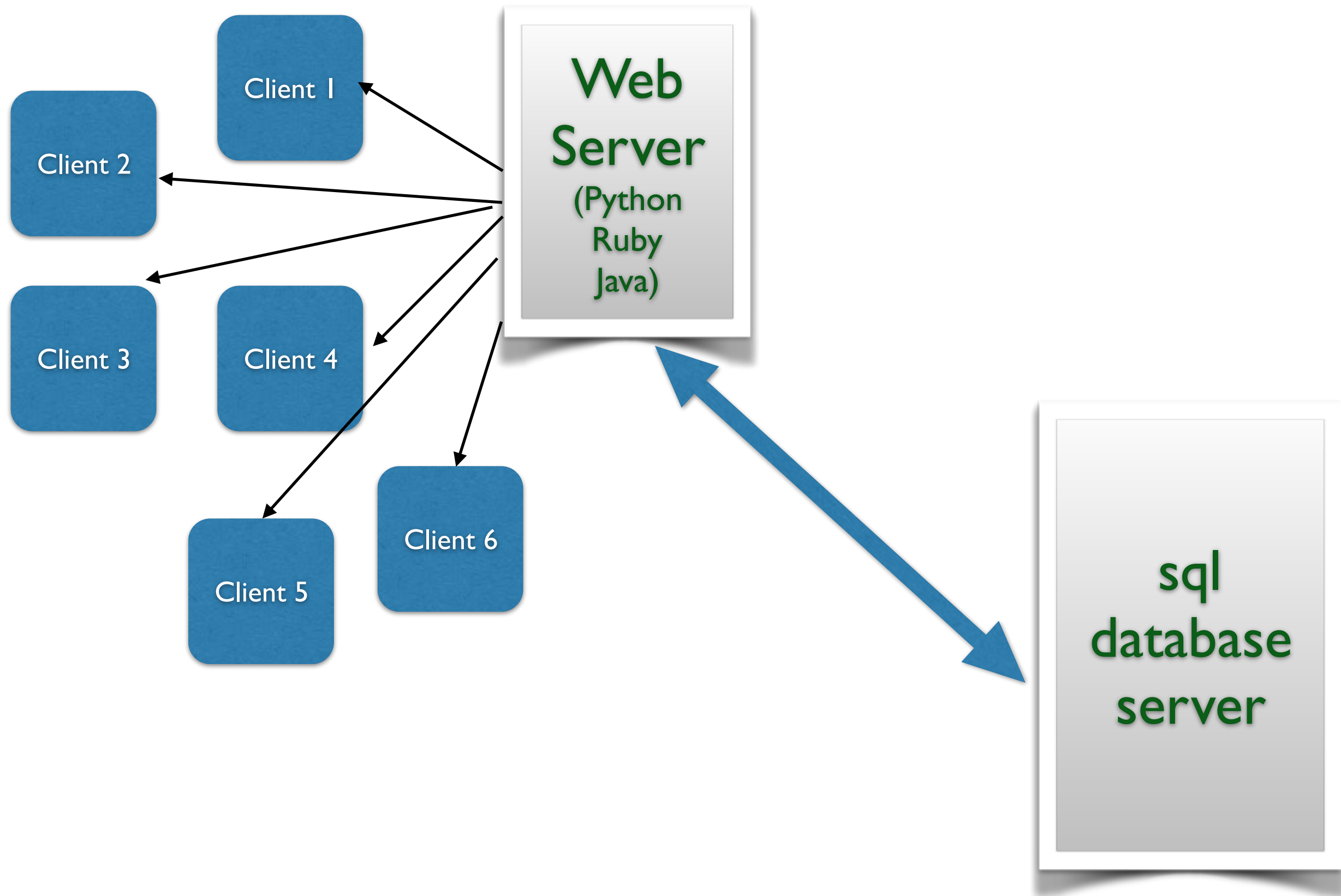
Digital organization methods:  
Relational databases  
NoSQL databases

# Who uses databases?

Almost every application today uses some  
sort of database!

If you go on the web, you're almost  
certainly interacting with a database

# Typical web app



# Relational databases

## Focus

- No redundancy
- Consistency across the database
- Efficient storage
- Simple design

## Design

- Data is stored in 2-dimensional tables
- Tables (relations) are logically connected sets of data
- Table rows (records/tuples) are information about one entity and have unique identifiers
- Table columns are attribute values
- SQL

# NoSQL Databases

## Focus

- Low latency
- Scalability
- Redundancy
  
- Typically stored on the cloud
- Does not (necessarily) use SQL (hence NoSQL)

## Examples

- Google BigTable (key-value-time triples)
- MongoDB (JSON like documents)
- Sparksee (Graphs – nodes and arcs)
- Dynamo and Amazon DynamoDB (key-value pairs)

# Relational databases

**Data Model:** How to model the structure of our data

**Relational model:** Converting our data model into a set of tables

**Normalization:** Remove redundancy and ensure data consistency



# Data model

## The Entity-Relationship Model

- provides an abstract and conceptual view of the data
- analyzes the entities and relationships in the data

# ER Model: Components

- **Entities**: Real world objects
  - ✦ student, course, professor, room
- **Relationships**: Association between entities
  - ✦ student **enrolled-in** course
  - ✦ professor **teaches** course
  - ✦ professor **advises** student
  - ✦ professor **has-office** room
- **Attributes**: Properties of entities or relationships
  - ✦ **student**: name, id\_number, major
  - ✦ **professor**: name, office, department
  - ✦ **professor teaches course**: rating

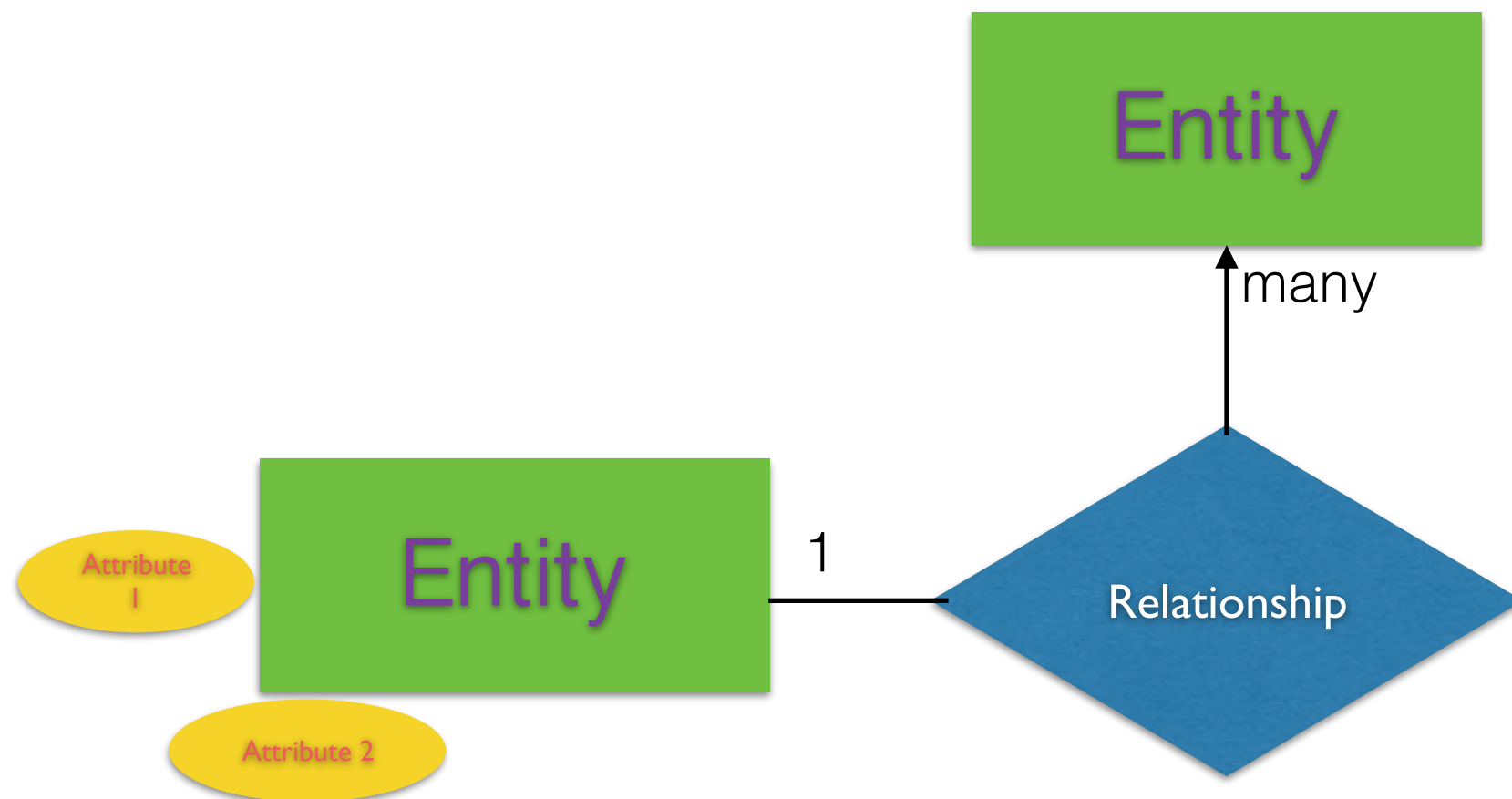
# ER Model: Relationships

## → Types of relationships:

- ♦ **one-one**: professor **has-office** room
- ♦ **one-many/many-one**: professor (1) **advises** student (many)
- ♦ **many-many**: student **enrolls-in** course
  - one student enrolls in many courses
  - one course has many students enrolled in it

# The Entity Relationship Diagram

ER models are represented diagrammatically



university course database

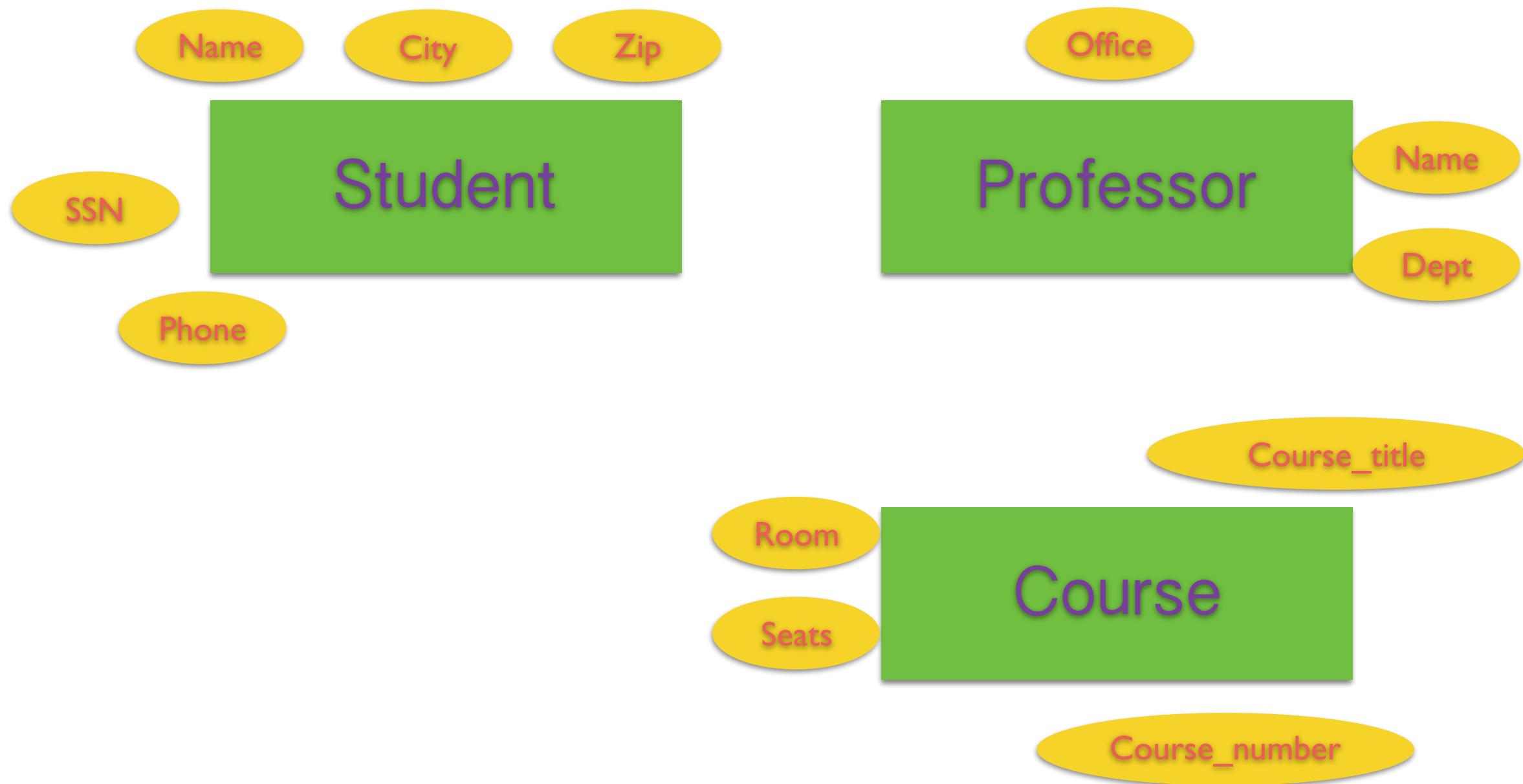
# Entities

Student

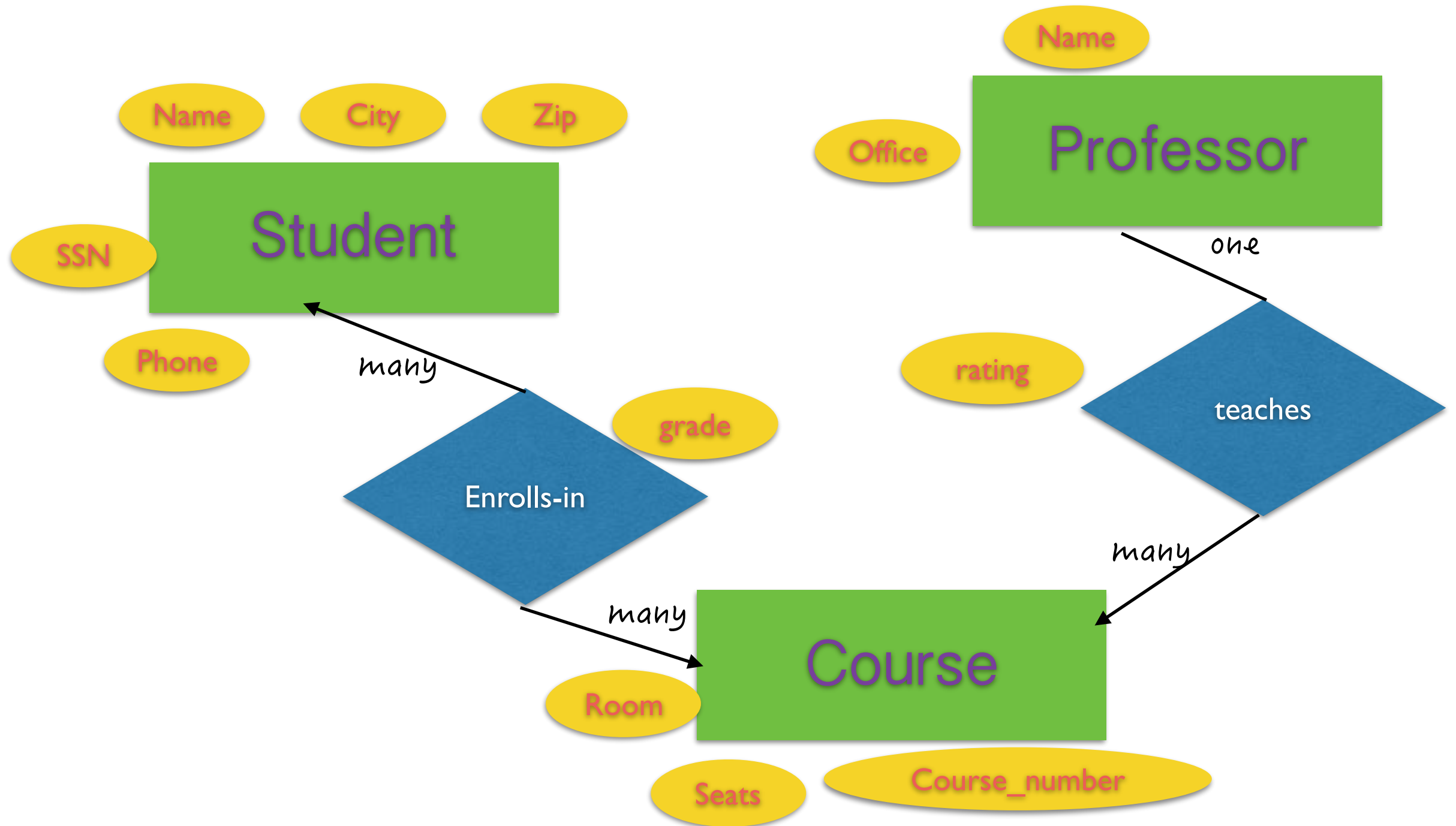
Professor

Course

# Attributes



# Relationships





# Keys

- **key**: an attribute, or a set of attributes, that uniquely identifies each object in an entity
- **Super key**: Any set of attributes that uniquely identifies an entity
  - \*student: name, ssno
  - \*student: name, city, phone
- **Candidate key**: A minimal super key
  - \*student: ssno
  - \*student: phone, city
- **Primary key**: The minimal candidate key
  - \*student: ssno

# Relational Database

- **ER Model**: A visual representation of the data model
  - the conceptual structure of the database
- **Relational model**: The database schema
  - the physical structure of the database
- **ER Model to Relational model**:
  - \* **Tables**: one table per entity and relationship
  - \* **Fields**: one field for each attribute
  - \* **Entity keys**: primary key
  - \* **Relationship keys**: composite. contains the key of each entity

# Entity tables

## Students

ssn	f name	l name	emails	phone	phone2	city	zip
111-22-3333	John	Childs	<u>john@gmail.com</u> , <u>jc123@columbia.edu</u>	646.123.1212		New York	10025
123-12-1234	Mary	Arias	<u>m.a@columbia.edu</u> , <u>ma222@hotmail.com</u>		347.766.7689	New York	10011
555-111-7777	Roberto	Perez	<u>perezr@gmail.com</u> , <u>rp1@columbia.edu</u>	917.333.5479	415.348.4789	San Francisco	94110
222-33-4455	Lila	Pennington	<u>lilap@gmail.com</u> , <u>lila@mail.com</u>	425.123.1212		Seattle	98105

## Courses

numb	name	room	seats
ieor1	Data	1127	60
ieor2	Python	303	100
ieor5	Simulation	331	55
ieor4	Optimizati	1127	60
ieor3	Machine	303	100
cs1	Intro to	303	100
cs2	Operating	1127	60

## Professors

name	office	bldg	dept
Prof. Micha	A702	mudd	ieor
Prof. Robyn	A301	cepsr	ieor
Prof. Lee	A673	mudd	cs
Prof. Wu	A244	uris	mech

# Relationship tables

## Enroll

ssn	f name	l name	class	grade
111-22-3333	John	Childs	ieor1	A
123-12-1234	Mary	Arias	ieor1	B
111-22-3333	John	Childs	ieor4	A
222-33-4455	Lila	Pennington	bus1	F

## Teaches

name	class	rating
Prof. Micha	ieor1	4.5
Prof. Robyn	ieor2	4.2
Prof. Lee	bus1	4.7
Prof. Wu	ieor4	5
Prof. Micha	ieor5	4.6

# Normalization

- The process of reorganizing a database to increase querying efficiency, consistency and to remove anomalies
  - insertion anomalies
  - update anomalies
  - deletion anomalies
  - repeating groups

# Insertion anomalies

- An insertion anomaly occurs when something needs to be added to the database but there is no place to add it
- **Example:**
  - The university decides to add a physics department
  - Initially, there are no faculty members
  - There is no way to add the department
- **Solution:**
  - Make **Department** into a separate entity and create a new **Department** → **Professor** relationship

# Insertion anomalies

## Professor

name	office	dept
Prof. Micha	A702	ieor
Prof. Robyn	A301	ieor
Prof. Lee	A673	cs
Prof. Wu	A244	mech

## Department

name
ieor
cs
mech
physics

## Belongs

name	dept
Prof. Micha	ieor
Prof. Robyn	ieor
Prof. Lee	cs
Prof. Wu	mech

# Update anomalies

- An update anomaly occurs when there is a change to the value of an attribute of an entity (or relationship) but that change needs to be made in multiple places
- A database with the potential for update anomalies can have redundant data and can therefore be inconsistent
- **Example:**
  - By removing the platform in front of classroom 1127, the number of seats is expanded to 70
  - The change needs to be made for every course that is being taught in room 1127
- **Solution:**
  - Make **Room** into a separate entity and create a new **Taught-in** → **Course** relationship



# Update anomalies

## Course

numb	name	room	seats
ieor1	Data	1127	60
ieor2	Python	303	100
ieor5	Simulation	331	55
ieor4	Optimizati	1127	60
ieor3	Machine	303	100
cs1	Intro to	303	100
cs2	Operating	1127	60

## Room

numb	seats
1127	60
303	100
331	55

## Taught-in

number	room
ieor1	1127
ieor2	303
ieor5	331
ieor4	1127
ieor3	303
cs1	303
cs2	1127

# deletion anomalies

- A deletion anomaly occurs when deleting something from the database results in some other, possibly important, fact being deleted as well
- A database with the potential for deletion anomalies can lose data
- **Example:**
  - Prof. Micha leaves for a position at MIT
  - The record in Professor is deleted
  - There is no longer a room A702 in our database and we also lose the fact that it is in Mudd
- **Solution:**
  - Make **Office** into a separate entity and create a new **has-office** → **Professor** relationship

# deletion anomalies

## Office

room	bldg
A702	mudd
A301	cepsr
A673	pupin
A244	mudd

## Professor

name
Prof. Micha
Prof. Robyn
Prof. Lee
Prof. Wu

## has-office

name	office
Prof. Micha	A702
Prof. Robyn	A301
Prof. Lee	A673
Prof. Wu	A244

# repeating groups

- repeating groups occur when an attribute has multiple values and all these values are included in the same table row
- A database with repeating groups is less efficient in data retrieval
- **Example:**
  - A student may have one, two, many phone numbers and emails
  - Necessitates null entries in some columns
  - What is the maximum number of phone numbers?
- **Solution:**
  - Make **Phone** into a separate entity and create a new **has-number** → **Student** relationship

# Normalization

- **First Normal Form:** no repeating groups; attribute values should be atomic
- **Second Normal Form:** every non-key attribute should be dependent on a full key (all attributes)
- **Third Normal Form:** No non-key attribute should depend on a non-key attribute

Databases that are in 3NF are unlikely to exhibit deletion, update, and insertion anomalies

# First Normal Form

Define Primary Key + Remove repeating groups and non-atomic values

## Students

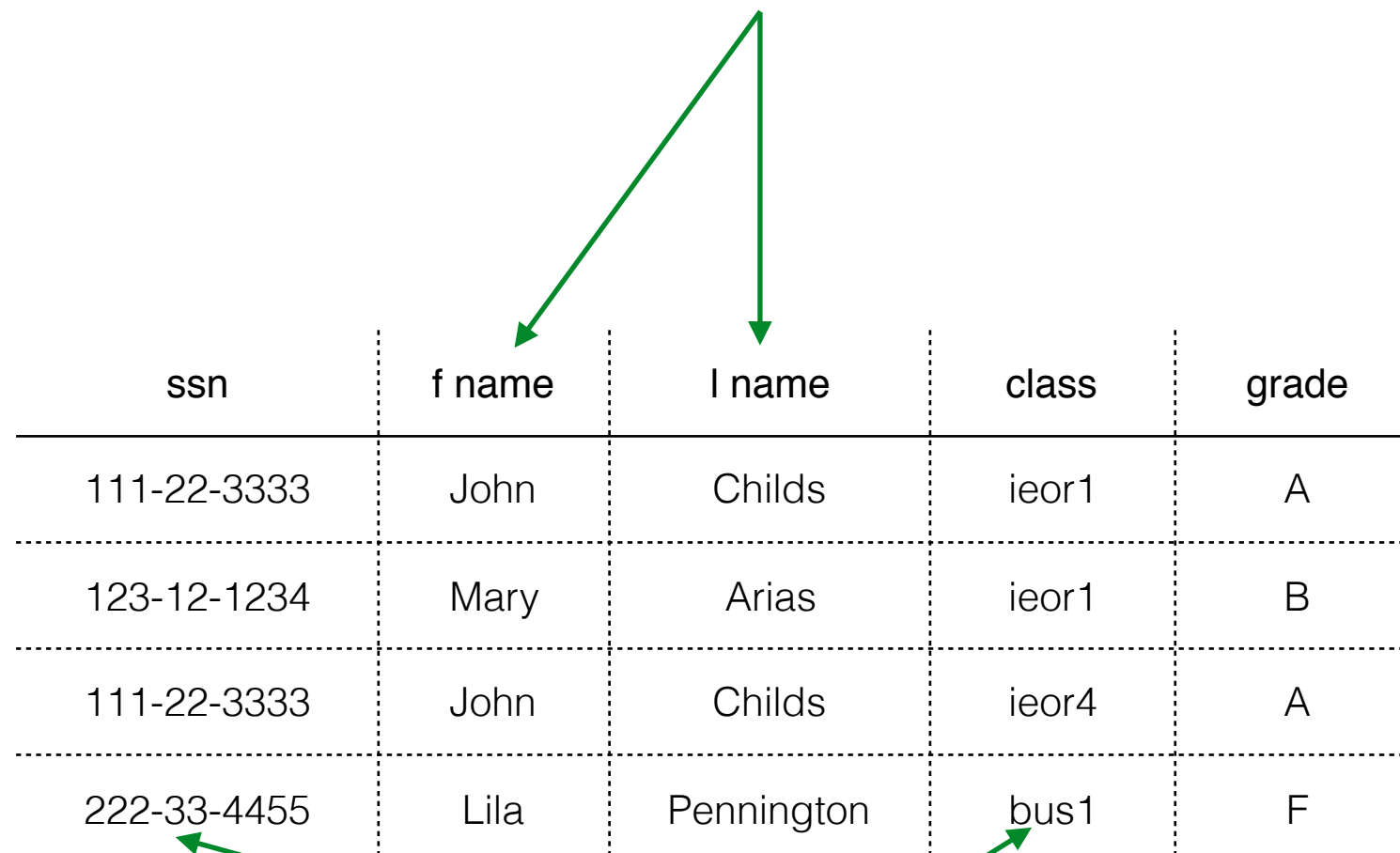


ssn	f name	l name	emails	phone	phone2	city	zip
111-22-3333	John	Childs	<u>john@gmail.com</u> , <u>jc123@columbia.edu</u>	646.123.1212		New York	10025
123-12-1234	Mary	Arias	<u>m.a@columbia.edu</u> , <u>ma222@hotmail.com</u>		347.766.7689	New York	10011
555-111-7777	Roberto	Perez	<u>perezr@gmail.com</u> , <u>rp1@columbia.edu</u>	917.333.5479	415.348.4789	San Francisco	94110
222-33-4455	Lila	Pennington	<u>lilap@gmail.com</u> , <u>lila@mail.com</u>	425.123.1212		Seattle	98105

# Second Normal Form

Remove non-key attributes that don't depend on a full key.

Enroll



ssn	f name	l name	class	grade
111-22-3333	John	Childs	ieor1	A
123-12-1234	Mary	Arias	ieor1	B
111-22-3333	John	Childs	ieor4	A
222-33-4455	Lila	Pennington	bus1	F

Composite Key

# Third Normal Form

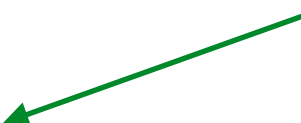
Remove non-key attributes that depend on a non-key attribute

Key

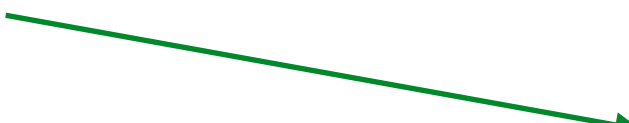


class	room	seats
ieor1_s1	Mudd 111	50
ieor2_s1	Mudd 321	75
ieor5_s1	Mudd 401	60
ieor4_s1	Mudd 401	60
ieor_ph2	Seminar	15

Taught-in



class	room
ieor1_s1	Mudd 111
ieor2_s1	Mudd 321
ieor5_s1	Mudd 401
ieor4_s1	Mudd 401
ieor_ph2	Seminar



room	seats
Mudd 111	50
Mudd 321	75
Mudd 401	60
Mudd 401	100
Seminar	15



# SQL

## Structured Query Language

- The language for relational databases
- SQL is a *declarative language* (“*what*” not “*how*”)
- SQL queries act on tables
- SQL queries return tables

# MySQL

- An open source relational dbms
- MySQL server is where the database resides
- MySQL clients are the programs that talk to the server
- One server can have many clients

# PostGresSQL

- Object relational database
- Designed for interaction with software
- Commonly used for web servers

# SQL Basics

## → List databases

- mysql: show databases;

- postgresql: \l

## → Create new database

- mysql/postgres: create database schooldb;

## → Choose a working database

- mysql: use schooldb;

- postgres: \c schooldb;

## → Delete a database

- mysql/postgres: drop database schooldb;

## → Identify current database

- mysql: select database();

- postgres: select current\_database();

# SQL Basics

## → List tables

→mysql: `show tables;`

→postgresql: `\d`

## → Create new table (relation)

→mysql:

→ `create table Students (SSN varchar(10), FirstName varchar(20), LastName varchar(40)) primary key (SSN);`

→postgres:

→ `create table Students (SSN varchar(10) primary key, FirstName varchar(20), LastName varchar(40));`

## → Add a new column

→mysql/postgres: `alter table students add Major int;`

## → Delete a table

→mysql/postgres: `drop table Students;`

# SQL Basics

- Show the structure of a table
  - mysql: `describe students;`
  - postgresql: `\d+ students;`
- Add data row into a table
  - mysql:
    - `insert into Students (SSN, FirstName, LastName) values ("123121234", "Kevin", 'Jensen');`
  - postgres:
    - `insert into Students (SSN, FirstName, LastName) values ('123121234', 'Kevin', 'Jensen');`
- List all data in a table
  - mysql/postgres: `select * from students;`
- List selected columns from a table
  - mysql/postgres:
    - `select firstname, lastname from students;`

# SQL Basics

## →Conditionally list from a table

### →mysql/postgres:

→ select SSN from Students where LastName='Jensen';

## →Sort the rows

### →mysql/postgres:

→select \* from students order by SSN;

## →Get unique rows

### →mysql/postgres:

→select distinct ssn from students;

## →Match a pattern

### →mysql/postgres:

→ select ssn from students where lastname like 'Jens%';

## →Choose from a set of values

### →mysql/postgres:

→ select \* from students where zip in ('10027','10025');

# SQL Basics

## → Modify a value

### → mysql/postgres:

→ `update students set zip = 10025 where ssn = '123121234';`

## → Delete a row

### → mysql/postgres:

→ `delete from students where zip = '10027';`

## → Group data

### → mysql/postgres:

→ `select count(zip) from students group by zip;`

→ `select zip, avg(balance) from students group by zip;`

## → List selected columns from a table

### → mysql/postgres:

→ `select zip, avg(balance) from students  
group by zip;`



# SQL Basics

## → Modify a value

### → mysql/postgres:

→ `update students set zip = 10025 where ssn = '123121234';`

## → Delete a row

### → mysql/postgres:

→ `delete from students where zip = '10027';`

## → Group data

### → mysql/postgres:

→ `select count(zip) from students group by zip;`

→ `select zip, avg(balance) from students group by zip;`

## → Work with multiple tables

### → mysql/postgres:

→ `select firstname, lastname, c.ssn, zip, balance,  
c.course from students, courses c where  
students.ssn = c.ssn;`