Introduction To Database Design

Demitri Muna
OSU

25 June 2014

Why Use A Database?

- Astronomy today generates more data each year than every year before it – combined.
- OK, I completely made that up, but enormous amounts of data are now generated (may be true with LSST!).
- Searching for what you are looking for in hundreds or thousands of FITS files is becoming increasingly unwieldy, repetitive, and time consuming.
- You are effectively writing your own search engine – let the CS community do that for you. (They're better than you at it anyway.)

Benefits of a Database

- One place to store all of your data, without worrying about file formats.
- Add other people's data for cross referencing/correlation to the same pool with little effort
- Searches are fast, whether you have one hundred objects or millions of them. The time taken for a full search does not scale geometrically as it would with simple files.
- You don't have to write a new program when you change the criteria for your search.
- The database language (SQL) is very easy to learn, and even easier to read.

Designing a Database

- You can take several full courses in database design and optimization, but for our needs, the basics are actually pretty straightforward (although maybe not immediately intuitive).
- The main principle is *normalization,* which is the idea that no information in the database is duplicated. You will frequently have to reorganize your data to accomplish this.
- The design (or blueprint) of the database is called a *schema*.

Creating a Database

The typical work flow to create a database:

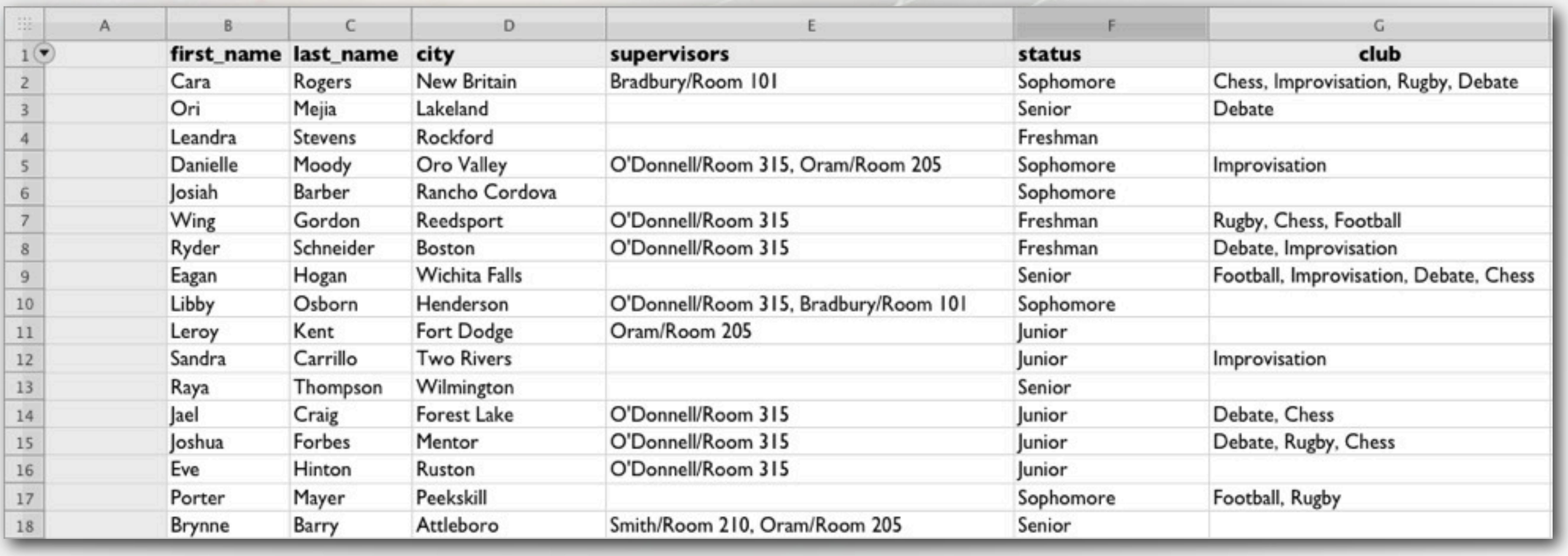
- Design a schema. Plan for future expansion/possibilities.
- Write a script to convert the data in it's given form (e.g. ASCII files, FITS files) into the new normalized form.
- Import the data into the database.

Designing A Schema

The best way to illustrate this is through a basic example.

What are the problems with data in this form?

see file “student_data.txt”



	A	B	C	D	E	F	G
1		**first_name**	**last_name**	**city**	**supervisors**	**status**	**club**
2		Cara	Rogers	New Britain	Bradbury/Room 101	Sophomore	Chess, Improvisation, Rugby, Debate
3		Ori	Mejia	Lakeland		Senior	Debate
4		Leandra	Stevens	Rockford		Freshman	
5		Danielle	Moody	Oro Valley	O'Donnell/Room 315, Oram/Room 205	Sophomore	Improvisation
6		Josiah	Barber	Rancho Cordova		Sophomore	
7		Wing	Gordon	Reedsport	O'Donnell/Room 315	Freshman	Rugby, Chess, Football
8		Ryder	Schneider	Boston	O'Donnell/Room 315	Freshman	Debate, Improvisation
9		Eagan	Hogan	Wichita Falls		Senior	Football, Improvisation, Debate, Chess
10		Libby	Osborn	Henderson	O'Donnell/Room 315, Bradbury/Room 101	Sophomore	
11		Leroy	Kent	Fort Dodge	Oram/Room 205	Junior	
12		Sandra	Carrillo	Two Rivers		Junior	Improvisation
13		Raya	Thompson	Wilmington		Senior	
14		Jael	Craig	Forest Lake	O'Donnell/Room 315	Junior	Debate, Chess
15		Joshua	Forbes	Mentor	O'Donnell/Room 315	Junior	Debate, Rugby, Chess
16		Eve	Hinton	Ruston	O'Donnell/Room 315	Junior	
17		Porter	Mayer	Peekskill		Sophomore	Football, Rugby
18		Brynne	Barry	Attleboro	Smith/Room 210, Oram/Room 205	Senior	

Designing A Schema

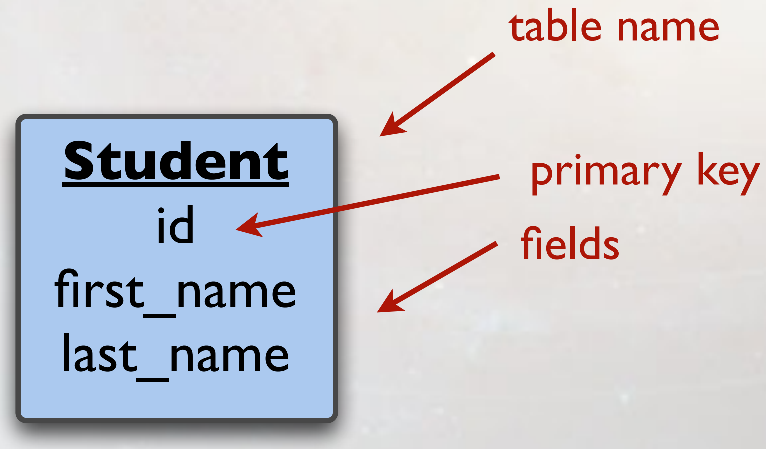
- Data is frequently repeated. A misspelling can lead to lost (or at least orphaned) data.
- Repeated data consumes more disk space unnecessarily.
- A spreadsheet doesn’t handle several pieces of data related to a single object.
- Only simple reports or queries are easy to make.
- You can’t put gigabytes of data into a spreadsheet.

Caveat

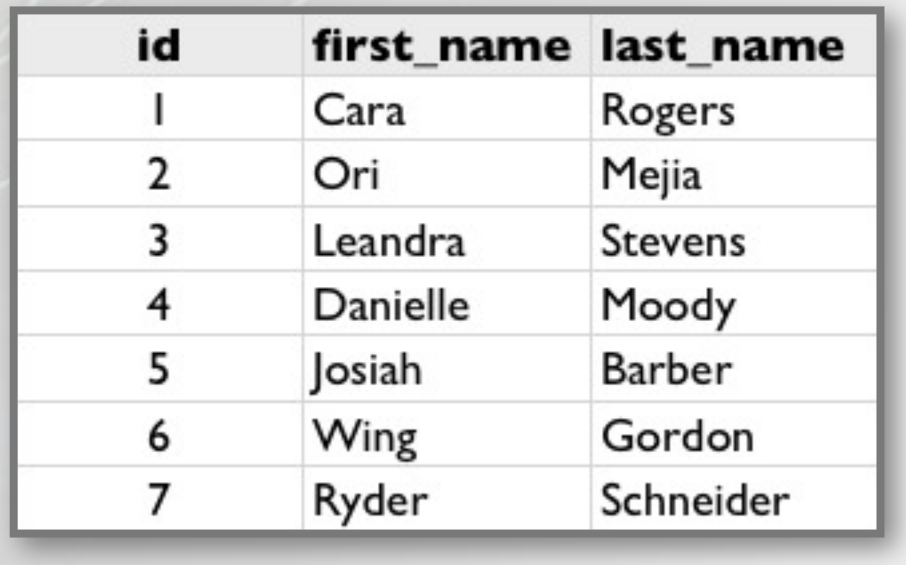
The example presented is a toy model. Inefficiencies here might seem trivial, but in a real dataset will quickly scale, e.g. wasting disk space, hinder efficient searches, etc.

Factoring the Data

Create a table for each "object" in your data model.
Think of a table as a single spreadsheet.



Student table
sample

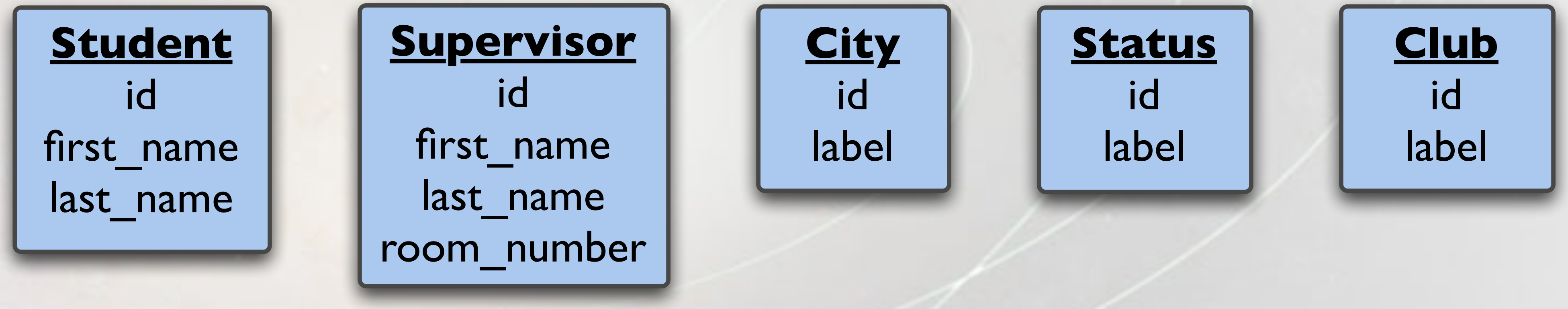


id	first_name	last_name
1	Cara	Rogers
2	Ori	Mejia
3	Leandra	Stevens
4	Danielle	Moody
5	Josiah	Barber
6	Wing	Gordon
7	Ryder	Schneider

Each table must have a *primary key*, which is a value that uniquely identifies a row. Typically this value is an integer.

It *should not* be meaningful or linked to any value in the table.
Repeat for every "noun" in the data model, e.g. 'city', 'status'.

Factoring the Data



Student
id
first_name
last_name

Supervisor
id
first_name
last_name
room_number

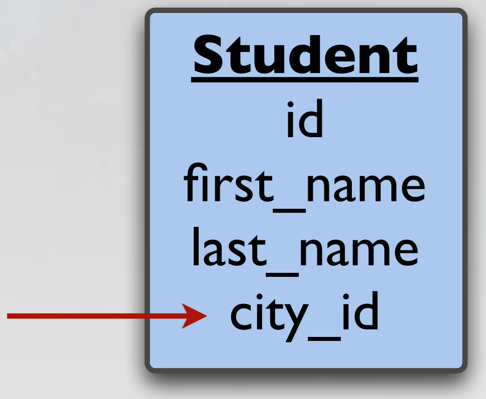
City
id
label

Status
id
label

Club
id
label

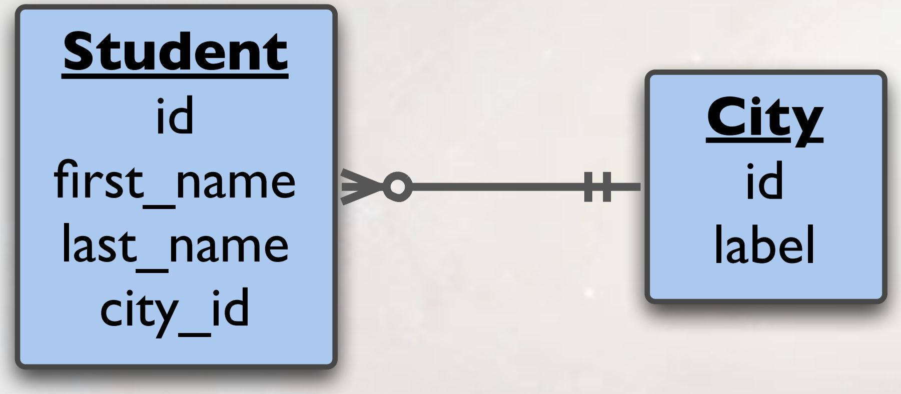
How do we know which city a student is from? Which clubs they belong to? Which supervisors they have?

Let’s start with the city. To identify a student with a city, we add a new field:



Student
id
first_name
last_name
city_id

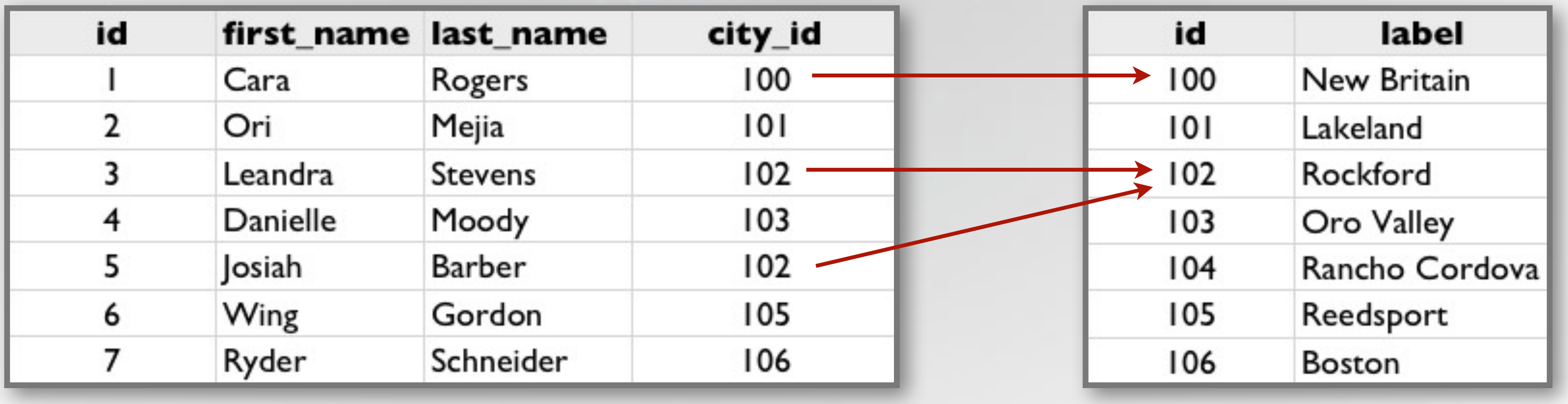
One-to-Many Relationship



- The field `city_id` in `student` maps to the primary key in the table `city` – this is called a *foreign key*.
- This defines *one-to-many relationship* – one student comes from one city, but one city can have many students.
- In this case, `city` is often called a *lookup table*, as the `city_id` field is used to look up the name of the city.
- An advantage is that the city name is itself is located in one and only one place in the database.

Student table

City table

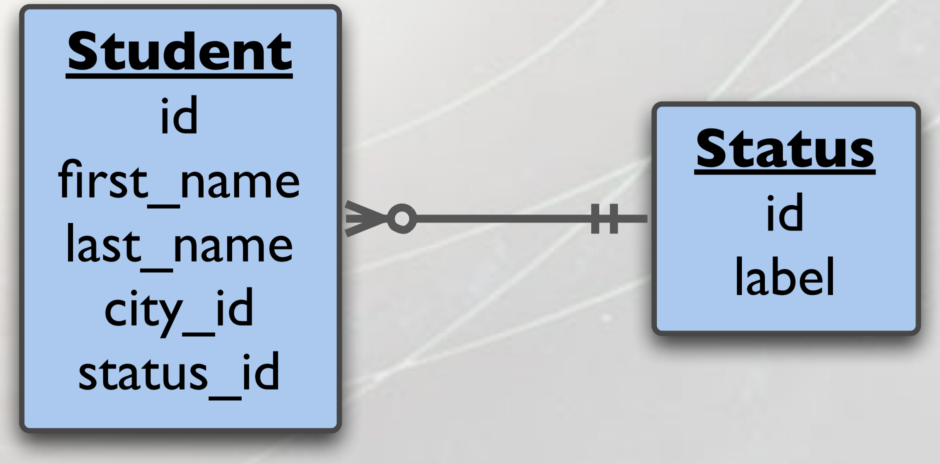


id	first_name	last_name	city_id
1	Cara	Rogers	100
2	Ori	Mejia	101
3	Leandra	Stevens	102
4	Danielle	Moody	103
5	Josiah	Barber	102
6	Wing	Gordon	105
7	Ryder	Schneider	106

id	label
100	New Britain
101	Lakeland
102	Rockford
103	Oro Valley
104	Rancho Cordova
105	Reedsport
106	Boston

Design A Schema

What about the relationship to status? The question to ask is, can a student (ever) have more than one status at a time?



This is another one-to-many relationship; `status` is another lookup table.

Student

id	first_name	last_name	city_id	status_id
1	Cara	Rogers	100	2
2	Ori	Mejia	101	4
3	Leandra	Stevens	102	1
4	Danielle	Moody	103	2
5	Josiah	Barber	102	2
6	Wing	Gordon	105	1
7	Ryder	Schneider	106	1

Status

id	label
1	Freshman
2	Sophomore
3	Junior
4	Senior

Many-to-Many Relationship

The relationship to supervisor is trickier as a student can have more than one supervisor, and a supervisor can certainly have more than one student. The same solution doesn't work:

Student
id
first_name
last_name
city_id
status_id
supervisor_id

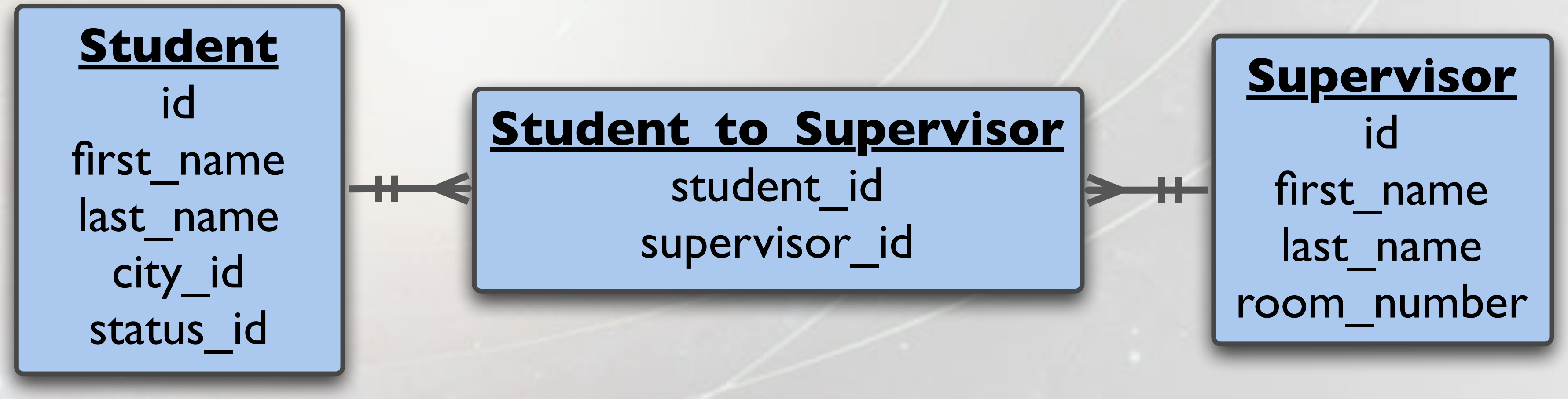
This allows only one supervisor at a time.

Student
id
first_name
last_name
city_id
status_id
supervisor1_id
supervisor2_id

- This doesn't allow more than two supervisors (could happen).
- Space for two supervisor foreign keys is taken up for *every* student (row), whether they have two or not. This can lead to a big waste of disk space.
- When you search for the supervisor, which field do you look up?

Many-to-Many Relationship

We solve this by introducing a new table to link `student` and `supervisor`:



Student
id
first_name
last_name
city_id
status_id

Student_to_Supervisor
student_id
supervisor_id

Supervisor
id
first_name
last_name
room_number

This is called a *join table*. Note the lack of a dedicated primary key. Here, the pairing of the `student_id` and the `supervisor_id` form a unique identifier. This is called a *joint primary key* or a *composite primary key*.

Some students have no supervisors; no space in the database is used in this case.

Note how Moody (student_id=4) has two supervisors.

Student

id	first_name	last_name
1	Cara	Rogers
2	Ori	Mejia
3	Leandra	Stevens
4	Danielle	Moody
5	Josiah	Barber
6	Wing	Gordon
7	Ryder	Schneider

Student_to_Supervisor

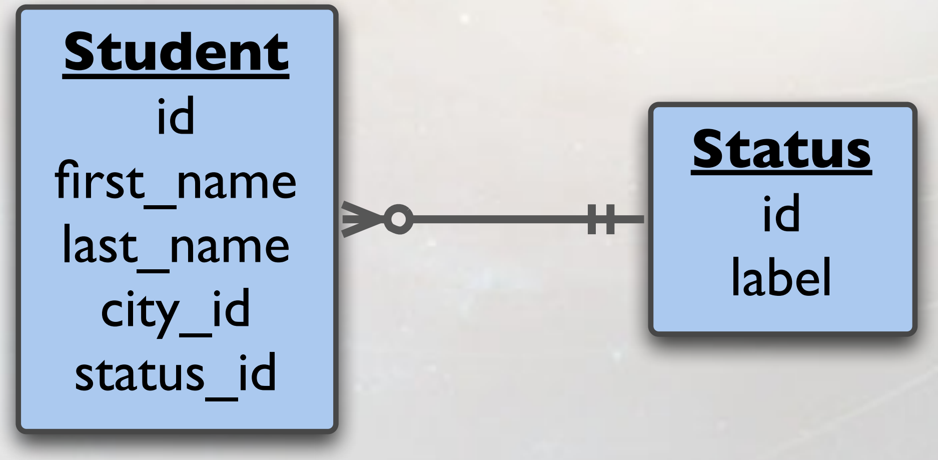
student_id	supervisor_id
1	10
4	4
4	9
6	4
7	4

Supervisor

id	first_name	last_name	room_number
10	David	Tennant	101
4	Tom	Baker	315
9	Christopher	Eccleston	205
11	Matt	Smith	210

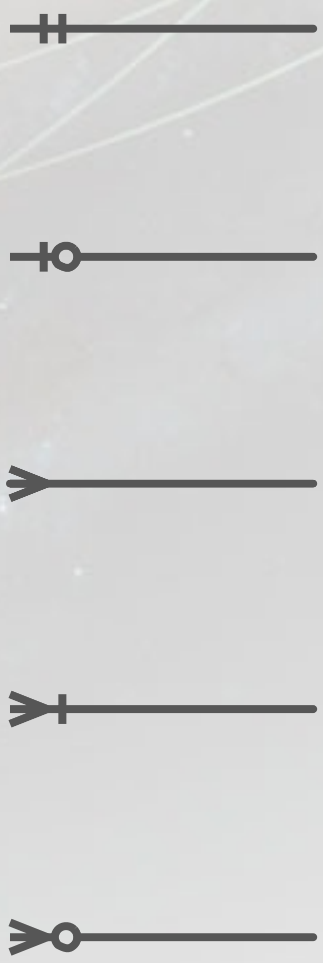
Schema Notation

Arrows in the schema note the relationship between tables. Sometimes the information is a note to the designer and is not enforced (only indicated) in the schema design, but can be in code.



A student must have one and only one status (e.g. a student cannot be a freshman and a senior, and cannot be unclassified.

Many students can have a particular status (e.g. there are many sophomores). There may be no students of a single status (e.g. "senior" is a status, but there may be no seniors).



one and only one

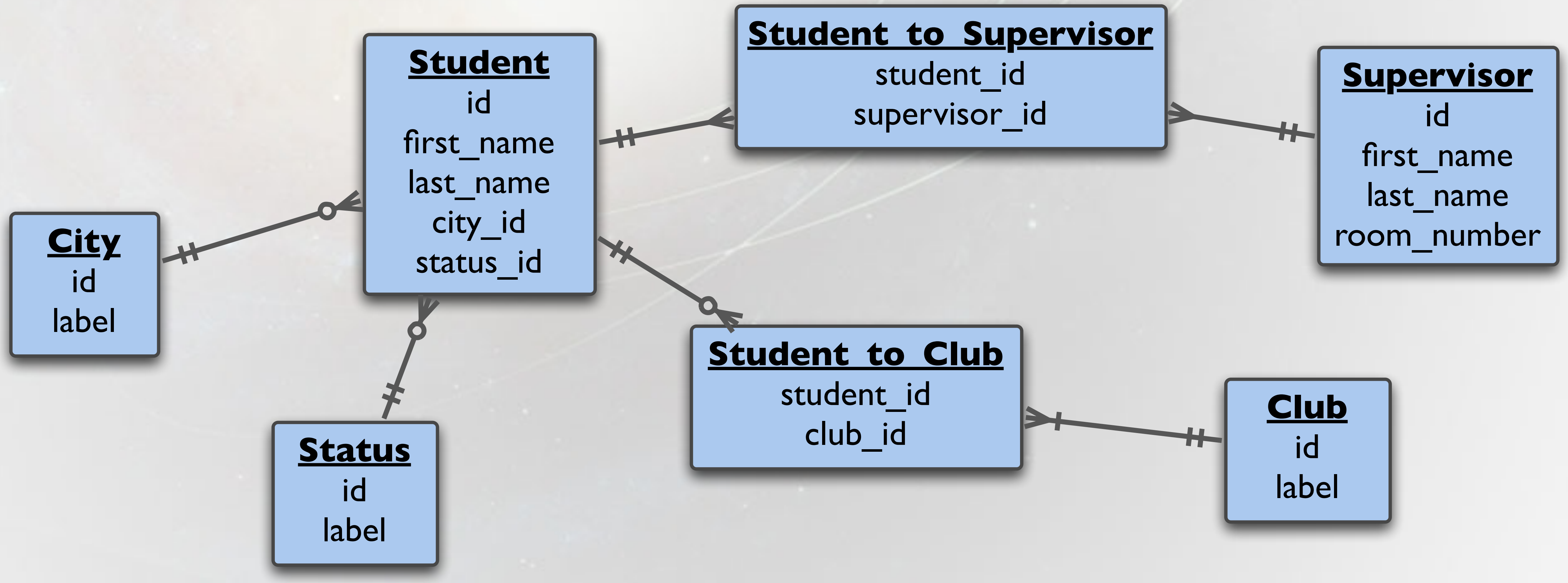
zero or one

to many

to one or many

to zero or many

The Full Schema



Student
id
first_name
last_name
city_id
status_id

Student_to_Supervisor
student_id
supervisor_id

Supervisor
id
first_name
last_name
room_number

City
id
label

Status
id
label

Student_to_Club
student_id
club_id

Club
id
label