Databases:
Intro to Relational Databases & Structured Query Language
Databases

A **database** is a collection of data that is organized and stored **according to some purpose**.

Defining my purpose precisely (audience, intentions, etc.) will help me make wise decisions later during the planning phases.

We need to store email addresses, and search by members' names!

So no need for postal codes, but make sure we can alphabetize!
The database is organized into tables (which resemble HTML tables or spreadsheets). Each table represents a real-world object or entity. Entities are usually nouns: a person or a university or a contract or even an event.
Databases

Each table is organized into rows and columns. Each row in the table is a record, which corresponds to one instance of that real-world entity. If our table stores info about people, for example, then one record would store John Smith’s data.
Databases

An entity usually has several characteristics or attributes. Each column corresponds to one attribute. Attributes of a person might include first name, last name, email address, and/or phone number.
Databases

The intersection of a row and a column (sometimes called a field or a cell) corresponds to one piece of information. John Smith’s email can be located by finding John Smith’s record and reading the matching cell in the email column.
Spreadsheets vs. Databases

A database is like a group of spreadsheets that can talk to one another.

Databases use a special language, Structured Query Language (SQL), to do that.
A database table corresponds to one spreadsheet. A database consists of multiple tables.

The top row of the table contains headers that describe the contents of each column. These headers aren’t data, exactly, but rather metadata — they’re data about the data. They label the attributes, but do nothing more.

Each record holds the data for one of the entity’s instances. Here, a record seems to be a person who owns pets. Each cell holds one piece of data.
I can ask questions of the table, and the database will give me an answer. If I ask, “Tell me the ID number of any person who has no pets,” the database will search the table’s columns and rows to find the answer.
I can ask questions of the table, and the database will give me an answer. If I ask, “Tell me the ID number of any person who has no pets,” the database will search the table’s columns and rows to find the answer.

Similarly, I could ask, “Tell me the name of any person who owns more than zero pets,” the database will search columns and rows to find that answer too. In this case, the right answer is Bob and Sue.
I can even ask questions that reach across multiple tables, and *that’s what makes a database far more powerful than a spreadsheet*. But I must organize my data very carefully to make use of that power. If my tables are disorganized, I cannot guarantee correct answers.
I can ask, “What’s Sue’s favorite pet’s name?” and query across tables to find the answer. My data has to be properly organized for a query like this to work. We’re about to spend some time learning those principles.
A database is a collection of data that is organized and stored according to some purpose.

The database is organized into tables that capture data about a real-world entity. Entities are nouns: people, places, things, events, etc. An entity might be somewhat abstract. You can think of it as an abstract template that describes concrete examples. (*Car* or *animal* might be entities.)

Each table is organized into rows and columns.

Each row in the table is a record, which corresponds to one instance (one example) of an entity. Honda Civic and giraffe are instances of *car* and *animal*.

Each column describes an attribute or a characteristic of the entity. *Year* and *colour* might be attributes of *car*. *Species* and *common name* might be attributes of *animal*.

Each cell or field contains one piece of information. In the *animal* table, the cell in the *giraffe* record under the *species* column might say *Giraffa camelopardalis*. 
Let’s say I belong to a birdwatching club and I’m building a database for my club. I need to keep track of my club members and I need to keep track of the birds they’ve seen and the dates when they saw those birds. Every month, I’ll award a prize to the person who’s seen the most birds.

Some club members are more experienced birdwatchers than others, and they are called “guides” because they can lead birdwatching hikes. Guides are not eligible for monthly prizes. Eventually, it would be nice to have a webpage listing all the monthly winners.

A “Member” table? Need some contact info, probably.
Building a DB: An Example

Let’s say I belong to a birdwatching club and I’m building a database for my club. I need to keep track of my club members and I need to keep track of the birds they’ve seen and the dates when they saw those birds. Every month, I’ll award a prize to the person who’s seen the most birds.

Some club members are more experienced birdwatchers than others, and they are called “guides” because they can lead birdwatching hikes. Guides are not eligible for monthly prizes. Eventually, it would be nice to have a webpage listing all the monthly winners.

A “Member” table? Need some contact info, probably.

A “Bird” table? Need to store dates too, somehow.
Let’s say I belong to a birdwatching club and I’m building a database for my club. I need to keep track of my club members and I need to keep track of the birds they’ve seen and the dates when they saw those birds. Every month, I’ll award a prize to the person who’s seen the most birds.

Some club members are more experienced birdwatchers than others, and they are called “guides” because they can lead birdwatching hikes. Guides are not eligible for monthly prizes. Eventually, it would be nice to have a webpage listing all the monthly winners.

A “Member” table? Need some contact info, probably.

A “Bird” table? Need to store dates too, somehow.

How to store the “guide” info? Let me think about that.
Building a DB: An Example

Let’s say I belong to a birdwatching club and I’m building a database for my club. I need to keep track of my club members and I need to keep track of the birds they’ve seen and the dates when they saw those birds. Every month, I’ll award a prize to the person who’s seen the most birds.

Some club members are more experienced birdwatchers than others, and they are called “guides” because they can lead birdwatching hikes. Guides are not eligible for monthly prizes. Eventually, it would be nice to have a webpage listing all the monthly winners.

A “Member” table? Need some contact info, probably.

A “Bird” table? Need to store dates too, somehow.

How to store the “guide” info? Let me think about that.

How to calculate and/or store these?
### Building the Member Table

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>email</th>
<th>phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bob Jones</td>
<td><a href="mailto:jonesy@blah.ca">jonesy@blah.ca</a></td>
<td>555-1234</td>
</tr>
</tbody>
</table>

My first attempt: a 4-column table. I decide the only contact info I need is **email** and **phone**. I decide to give everyone a unique **ID** number, which may help prevent confusion later (I have multiple Bobs in my club).

But what if I want to, for example, sort by last name? I can split the **name** column into **first** and **last**, which will help.
### Building the Member Table

<table>
<thead>
<tr>
<th>id</th>
<th>first</th>
<th>last</th>
<th>email</th>
<th>phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bob</td>
<td>Jones</td>
<td><a href="mailto:jonesy@blah.ca">jonesy@blah.ca</a></td>
<td>555-1234</td>
</tr>
</tbody>
</table>

My second attempt: a 5-column table. I’ve adhered to an important principle of database design:

**Design Principle #1**: Each cell in the table should contain only one piece of information.

You may decide that a name is only one piece of information. Here, though, I’ve decided that it’s two pieces of information.
My first attempt: a 2-column table. I decide to give every bird a unique **ID** number, which may help prevent confusion later. I also identify the **species**.

Is this specific enough? After all, birds have common names and they have scientific, Latin names. Let me try again.
My second attempt: a 3-column table. Each record represents a different species, but now I’m listing the official names, both common and scientific. I can get this data from my identification books, from encyclopedias, from other websites, etc. The ID number will allow me to verify that common names match scientific names.

I may choose to add more information later, but for now this seems sufficient. And I comply with Principle #1: each cell = 1 piece of data.
Next, I need to figure out how to store my log of bird sightings. Is the main organizational principle the person who saw the bird or the bird that was seen? Since I want to award a prize to the person who’s seen the most birds, I decide that person is the main criterion.

Here’s one option: I could add some columns to my member table, like this:

<table>
<thead>
<tr>
<th>id</th>
<th>first</th>
<th>last</th>
<th>email</th>
<th>phone</th>
<th>bird</th>
<th>date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bob</td>
<td>Jones</td>
<td><a href="mailto:jonesy@blah.ca">jonesy@blah.ca</a></td>
<td>555-1234</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

But now I need to keep adding columns for every bird sighting. *That’s bad!* I don’t want to alter my table when I add new data.
What if I create a new table, and store the person and the birds:

<table>
<thead>
<tr>
<th>person</th>
<th>sightings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sue</td>
<td>magpie, Sept 12</td>
</tr>
<tr>
<td></td>
<td>Blackgird, 10/4</td>
</tr>
<tr>
<td></td>
<td>crow; raven, September 14th</td>
</tr>
</tbody>
</table>

DISASTER! Lots of problems.
Which Sue? Was it a black-billed magpie? What’s a blackgird? Is it one of the three species of blackbird? If so, which one? Is 10/4 October 4 or April 10? Why are crow & raven put together? What does the ; signify here? How can I standardize dates?
**Violations of Good Table Design**

<table>
<thead>
<tr>
<th>person</th>
<th>sightings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sue</td>
<td>magpie, Sept 12</td>
</tr>
<tr>
<td></td>
<td>Blackgird, 10/4</td>
</tr>
<tr>
<td></td>
<td>crow; raven, September 14th</td>
</tr>
</tbody>
</table>

**Principle #1: Store only one thing per field.** I’m storing 7 things right now in the sightings column. The field is impossible to search effectively because it contains too much data and that data is not standardized (“normalized”).

**Principle #2: Store data in only one place.** I’m repeating Sue’s name from the member table; I’m repeating bird names (and spelling errors) from the bird table. Data stored in only one place is automatically normalized and is easy to update.
Building the Sightings Table

My third (?) attempt: use a separate table, but store just the *minimal* amount of information necessary. Adhere to my Design Principles: 1) one item per field; 2) store information in only one place.

Notice now that I’ll store just the ID numbers (and nothing more!) from the *member* table and from the *bird* table.

There are two crucially important concepts at work here: *keys* and *data normalization*.

<table>
<thead>
<tr>
<th>person</th>
<th>bird</th>
<th>date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2005-09-28</td>
</tr>
</tbody>
</table>
The ID numbers in each of these tables is a **primary key**. The numbers in that column are **never duplicated again in any other record**. Each record in the table has a unique primary key. I use the primary key as my reference to that individual record. Primary keys are not unique across the entire database, just unique within individual tables.
The **person** and **bird** columns in my **sightings** table are **foreign keys**. They are not unique within this table; they simply refer to unique records in tables different from this one.
Foreign keys can be duplicated many times. I can’t tell from here which birds or people these are, but I can check the other tables to find out. *Notice that I have purposely not duplicated information in multiple tables.* I have repeated keys, but not other important information.
Design Principle #2: Store each piece of data in only one place.

Use a primary key to identify an element’s record. Use foreign keys to “relate” or to “connect” data from one table to another.

These principles are called data normalization. The goal is to remove data redundancy in order to produce a well-organized database design that is a high-quality representation of the real world. Data normalization increases the clarity and accuracy of the data model.

The standard practice is to name tables and columns in the singular. So instead of members, I choose member. Instead of birds, I choose bird.
Let’s say I belong to a birdwatching club and I’m building a database for my club. I need to keep track of my club members and I need to keep track of the birds they’ve seen and the dates when they saw those birds. Every month, I’ll award a prize to the person who’s seen the most birds.

Some club members are more experienced birdwatchers than others, and they are called “guides” because they can lead birdwatching hikes. Guides are not eligible for monthly prizes. Eventually, it would be nice to have a webpage listing all the monthly winners.

How to store “guides”? Maybe on the member table...?
Guides: One Solution

<table>
<thead>
<tr>
<th>id</th>
<th>first</th>
<th>last</th>
<th>email</th>
<th>phone</th>
<th>guide</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bob</td>
<td>Jones</td>
<td><a href="mailto:jonesy@blah.ca">jonesy@blah.ca</a></td>
<td>555-1234</td>
<td>yes</td>
</tr>
</tbody>
</table>

I can add a column to my member table, and insert the value yes if this person is a guide. If not, I can set it to no or else leave it NULL.
### Guides: One Solution

<table>
<thead>
<tr>
<th>id</th>
<th>first</th>
<th>last</th>
<th>email</th>
<th>phone</th>
<th>guide</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bob</td>
<td>Jones</td>
<td><a href="mailto:jonesy@blah.ca">jonesy@blah.ca</a></td>
<td>555-1234</td>
<td>yes</td>
</tr>
<tr>
<td>2</td>
<td>Sue</td>
<td>Rogers</td>
<td><a href="mailto:sr@blah.ca">sr@blah.ca</a></td>
<td>555-9876</td>
<td>no</td>
</tr>
<tr>
<td>3</td>
<td>Bob</td>
<td>Smith</td>
<td><a href="mailto:bs@bs.ca">bs@bs.ca</a></td>
<td>555-4321</td>
<td>no</td>
</tr>
</tbody>
</table>

If I don’t have many guides, most of the values will be no. Although this is a valid solution (and probably how I’d do it in a real production system), here I want to opt for a different solution.
I’ll create another table called `guide` that stores only the ID’s of the people who serve as guides. Now I do not have to waste time searching through a bigger table to find them. If I get a new guide, I’ll add a new record. If someone doesn’t want to be a guide, I’ll remove that person’s record from this table. They’re still a member, just not a guide.
Let’s say I belong to a birdwatching club and I’m building a database for my club. I need to keep track of my club members and I need to keep track of the birds they’ve seen and the dates when they saw those birds. Every month, I’ll award a prize to the person who’s seen the most birds. Some club members are more experienced birdwatchers than others, and they are called “guides” because they can lead birdwatching hikes. Guides are not eligible for monthly prizes. Eventually, it would be nice to have a webpage listing all the monthly winners.

This sounds like a calculation. I’ll have the database count up the number of sightings for me and give me that person’s name. I won’t store this anywhere, but instead will calculate the answer whenever I need it.
**Group Time: Design a Database**

1. **Nouns = Tables.** Look at your data (or write a prose description of your project) and determine the important nouns. They are good candidates to become tables.

2. **Verbs = Relationships.** These are often signified by “is a” or “has a” relationships, but other verbs can signify relationships too. These connections between tables are often signified graphically by drawing a line from one table to another. We’ll model that as a primary key / foreign key relationship.
   - Shelly has a Mercedes.       (relationship b/t person and car)
   - Ann is the author of *How to Hike*.   (relationship b/t person and book)
   - Blue Buck is brewed by Phillips.       (b/t beer and brewery)
   - Facebook trades on NASDAQ.   (b/t company and stock exchange)

3. **Adjectives = Columns.** Important properties of nouns often becomes attributes in your tables. Remember, not every adjective needs to be captured! And some nouns (e.g., email address) should be attributes!
Major nouns become entities and get their own tables. Not all nouns are worthy, however!

An entity’s salient properties become its column attributes. Sometimes these are adjectives; sometimes these are nouns.

Store only one thing per field. Even lists of data must be stored one piece at a time.

Store each piece of data in only one place. Data stored in only one place is automatically normalized and is less prone to errors.

Don’t store what you can calculate. Store a person’s birthdate, not their age.