Database Theory

Why learn about databases?

- ☞ A website without a database is just a static collection of text
- ☞ With a database backing our website, we have a place to put lots of important data, such as:
  - ☞ user information
  - ☞ product details
  - ☞ blog posts
  - ☞ etc

What is a database?

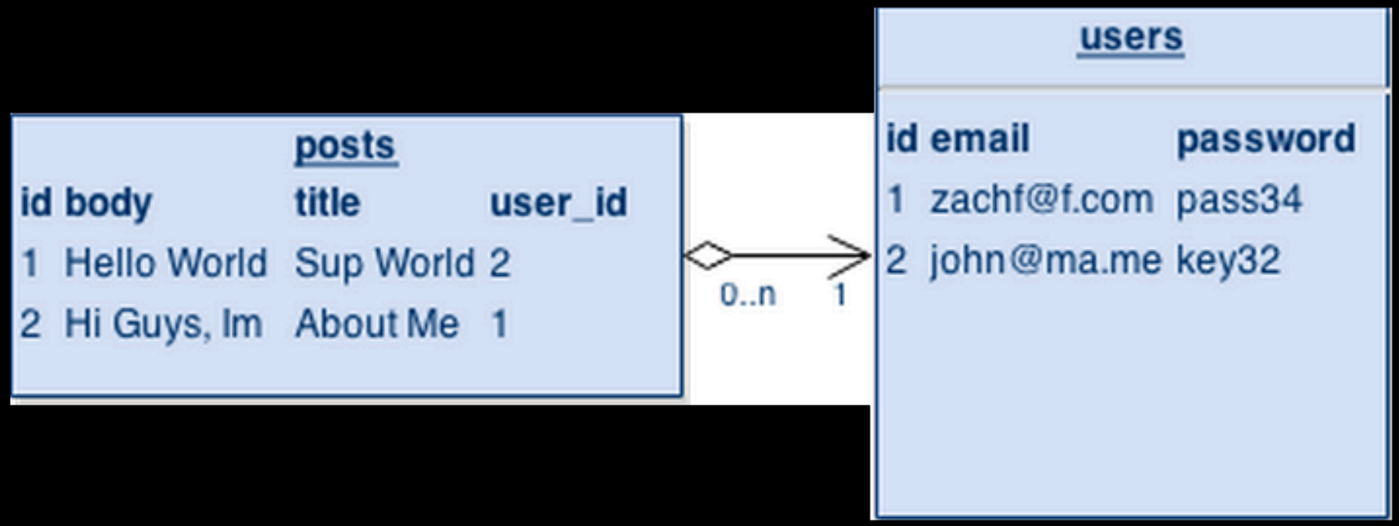
- ☞ A database is a collection of data organized into tables
- ☞ Each table in the database is typically used to model real-life objects, such as a user, a transaction, or an article
- ☞ Data can be inserted, read, updated, and destroyed from the database
- ☞ The database itself lives in a file on your computer

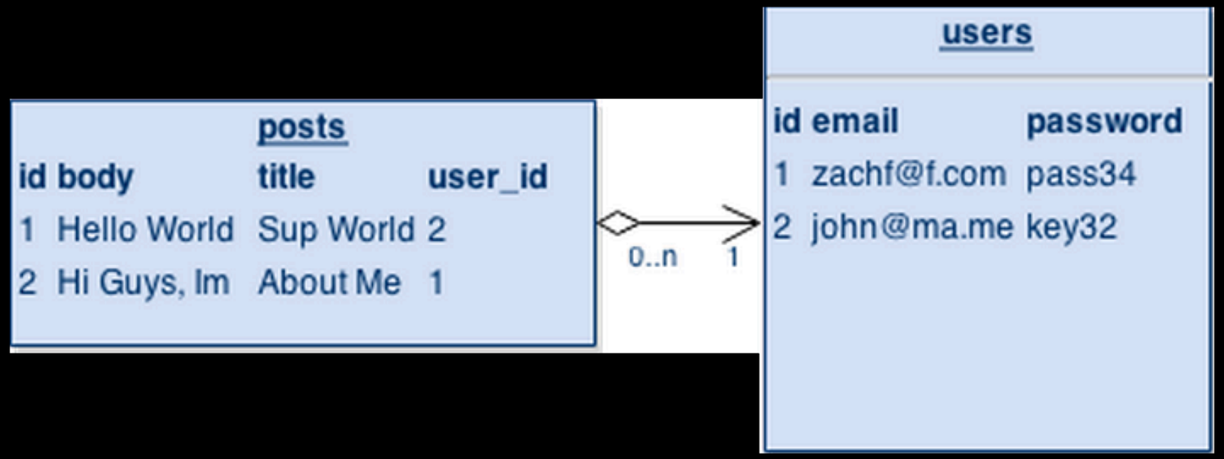
What is a database?

- ☞ Why not just use an Excel spreadsheet? Each table is **related to other tables in the system** when we use a database!
- ☞ This allows us to perform queries against multiple tables and retrieve data that is related to other data in the database

Database Example

Let's say that we have a simple database with a **users** table and **posts** table:



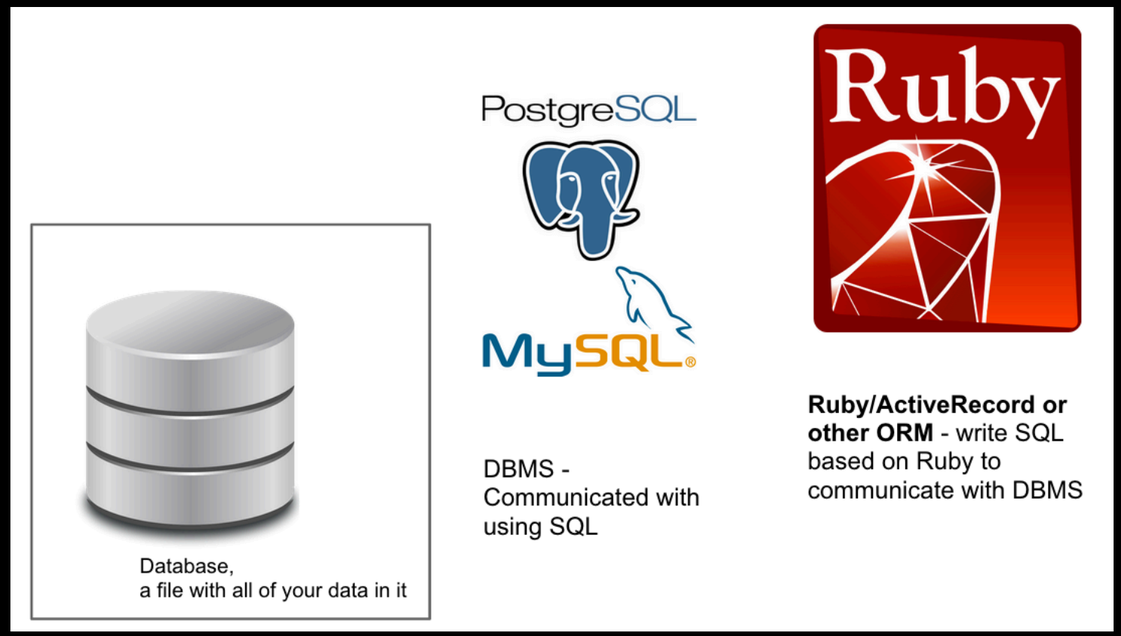


Each post belongs to a user and each user can have many posts (more on this later)

This simple concept, that tables can be related to each other and we can query “all of the posts that are associated with user 1” is the true power of the relational database.

The Bigger Picture

Each of these will become more clear as we address them.
For now though, here is the bigger picture:



Database Management Systems

- ☞ A **D**ata**B**ase **M**anagement **S**ystem, or **DBMS** is a software program that provides an interface to deal with the database itself
- ☞ It would be impossible for us to interact with the underlying data file that represents our data without using a DBMS of some kind

DBMS's - Examples

Oracle

- ☞ Sybase
- ☞ MySQL
- ☞ PostgreSQL
- ☞ SQLite

SQL = "Structured Query Language"

- ☞ Almost very DBMS comes with a version of **SQL**, or **S**tructured **Q**uery **L**anguage
- ☞ SQL allows you to declaratively perform all of the actions you'd need to on a database
  - ☞ create new records, read records, update records
  - ☞ destroy records, create new tables, create new columns on tables

SQL: Example

We won't be learning a TON of SQL in this class (you'll see why in a few slides), but here's a short example of generic SQL...

...assuming we'd already created a users table like the one described in our simple example earlier, we could insert a new record into it like so:

```
INSERT INTO users (email, password) VALUES ("zach@nycda.com", "pass1234");
```

SQL: Example

- ☞ Why is there no ID in our user record?
- ☞ SQL uses an AUTO-INCREMENT feature to assign a unique ID to every single row in a table called a **primary key**
- ☞ More on this later!

Ruby Web Applications + Databases

- ☞ Web applications written in Ruby typically use the **SQLite3** DBMS for testing locally because of its ease of use and quick setup
- ☞ **PostgreSQL** is used in production, meaning on the web server (computer) that will host the version of your app consumers interact with

The Bigger Picture

Each of these will become more clear as we address them.
For now though, here is the bigger picture:

ORMs - Object-relational mapping

- ☞ Ruby uses an ORM system to create a “virtual object database”
- ☞ Database tables are represented as Ruby objects
- ☞ These objects have special **methods** to perform tasks on our database by **writing SQL for us**

The Bigger Picture

Each of these will become more clear as we address them.
For now though, here is the bigger picture:

ORMs - Object-relational mapping

- ☞ For instance, consider a `users` table in your app's database
- ☞ Using ActiveRecord, a Ruby ORM system, you can **wrap** that table inside of a special `User` object
- ☞ Once you've defined that `User` object and linked it to the ActiveRecord ORM, you can call methods on it to interact with the `users` table

ORM - High-level Example

① Create a `users` table

② Create a Ruby **class** called `User`, extend it from the ActiveRecord library to access the ActiveRecord library's ORM functionality

③ Make queries on the underlying `users` table like so, in Ruby, that get converted into SQL by ActiveRecord.

```
user = User.find(5)
```

becomes

```
SELECT * FROM users WHERE user_id = 5;
```

ORM - High-level Example

1. ActiveRecord-written SQL gets run against the database and gets back a result, which it returns in a Ruby object:

```
<#User:x342344 name: "Zach Feldman", email: "zach@nycda.com">
```

1. Assuming the query was assigned to a Ruby variable called user, we can now query that variable, where the above object is stored:

```
user.email
> zach@nycda.com
```

ORMs: Web App Example

1. User requests information on a specific order from, say, Amazon.com's website or "web application"

2. Web application uses Ruby object representation of order to query the database

```
# Declare Order object to wrap
# the orders table
class Order < ActiveRecord::Base
end

# Query the table with ActiveRecord
Order.find(5)
```

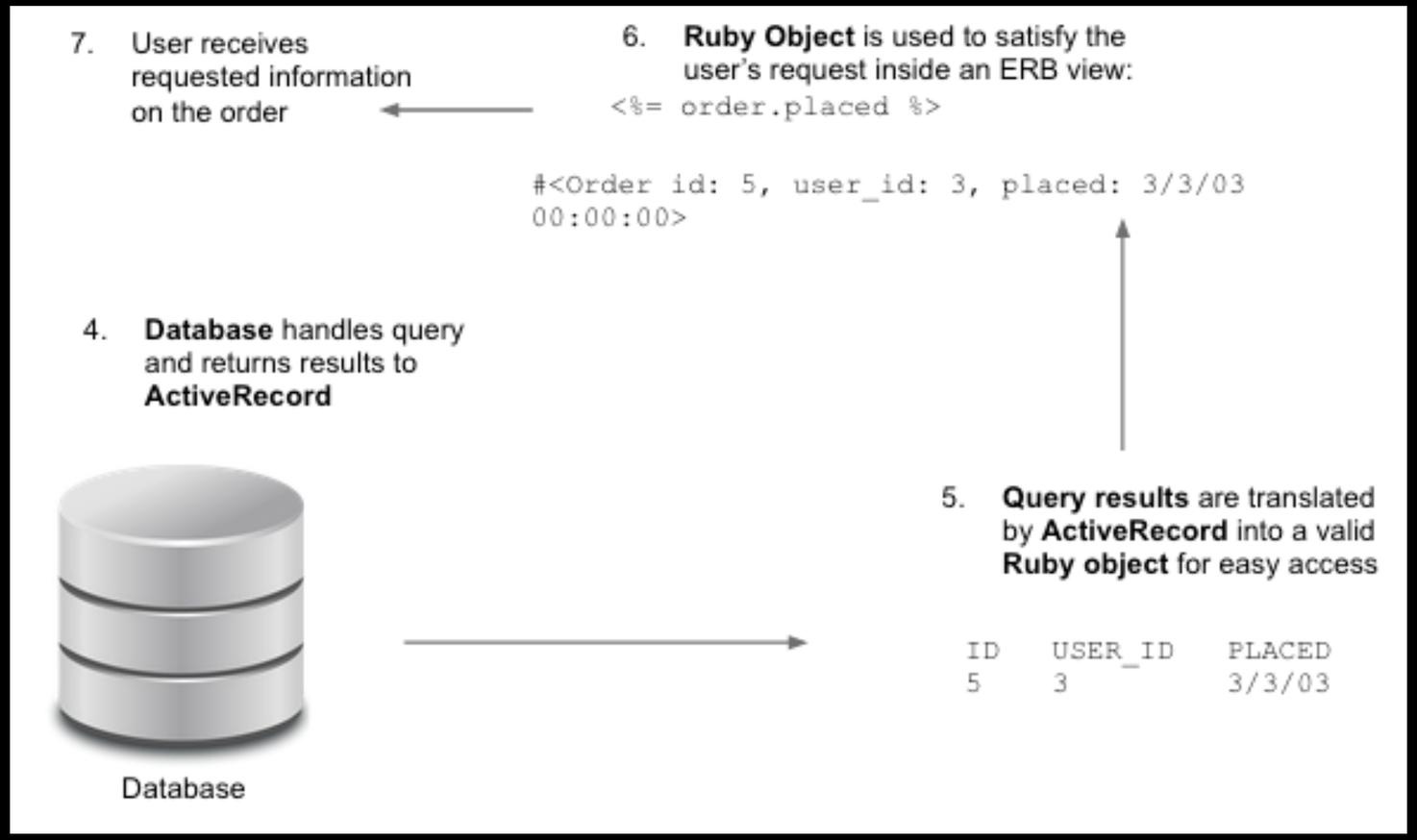
3. **Ruby** code is translated to **Structured Query Language (SQL)** code by **ActiveRecord,** database is sent SQL query

```
SELECT * FROM
orders WHERE id=5
```

4. **Database** handles query and returns results to **ActiveRecord**

Database

ORMs: Web App Example



Database column data types: in general

☞ Similar to Ruby and JavaScript, database columns also have data types to be able to perform operations like math and concatenation (joining of strings) on similar types of data

☞ Data types vary by **DBMS** type, but **ActiveRecord** generally refers to them like so:

```
string #for text < 255 chars
text #for text > 255 chars
integer #for numbers like 32,1,50, 1000
float #for numbers with a decimal place
datetime #for date + time data
boolean #for true/false data
```

Database column data types: with a specific DBMS (PostgreSQL)

PostgreSQL, a different DBMS, generally refers to database data types like so:

```
VARCHAR(length) #for text
integer #for numbers like 32,1,50, 1000
real #for numbers with a decimal place
timestamp #for date + time data
boolean #for true/false data
```

Constructing a database table: schema

- ☞ In order for us to store data in a database table, we'll need to define its **schema** first
- ☞ A **schema** is kind of like the first row of an Excel spreadsheet with the titles at the top
- ☞ The schema defines what kind of data we'd like to store in our table and what we'd like to refer to it as, kind of like a map of our DB
- ☞ To start a schema, we'll start with a specific table in the database, some columns, and their data types!

Example Users table schema

Users

```
id: integer
fname: string
lname: string
email: string
created_at: datetime
admin: boolean
```

The primary key

- ☞ The ID column, typically an integer, is the **primary key** for most tables in the database
- ☞ This means that it is the primary means of referring to a record in a **unique** way, by table
- ☞ Primary keys are typically NOT allowed to be duplicated in another record so each record has a unique identifier

Exercise

- ☞ Design the schema for a transactions table to represent the data for a monetary transaction
- ☞ Note which column is the **primary key**
- ☞ We'll go over it together in a few minutes!

Database Relationships

- ☞ Tables in the database are typically related to each other in certain ways
- ☞ This increases the efficiency of the database
- ☞ For example, instead of having many extra columns on each `transaction` record to store the related user's name, email, phone number, etc, we could simply store that user's ID and look up that information in the `Users` table

Database Relationships: 1-to-1

- ☞ In a one-to-one relationship, a record in one table is related to only one record in another table.

- ☞ Examples:
  One `User` has one `Profile` record
  One `Customer` has one `CustomerDetail` record
  One `Citizen` has one `SocialSecurityNumber` record

Database Relationships: 1-to-1

- ☞ To implement this sort of relationship, one of the tables being linked together, A, needs to contain a **foreign key** that references the **primary key** of the other table B
- ☞ For instance, a `Profile` record needs to contain a `user_id:integer` column so we know which user it should be associated with

Database Relationships: 1-to-many

- ☞ In a one-to-many relationship, a record in one table is related to many records in another table

- ☞ Examples:
  One User has many BlogPosts
  One User has many Orders
  One User has many Tweets

Database Relationships: 1-to-many

- ☞ Similar to a one-to-one relationship
- ☞ To implement this sort of relationship, one of the tables being linked together, A, needs to contain a **foreign key** that references the **primary key** of the other table B
- ☞ For instance, a `Profile` record needs to contain a `user_id:integer` column so we know which user it should be associated with

Database Relationships: many-to-many

- In a many-to-many relationship, a record in one table is related to many records in another table and vice versa.
- Examples:
  - Many Users have many Addresses
    (an address can belong to multiple users, multiple users could have the same address)
  - Many Products have many Orders
    (a product could be ordered many different times, each order could have many different products in it)

Database Relationships: many-to-many

- To implement this sort of relationship, we need what is called a **join table**, a third table to associate records together from two separate tables
- For instance:
  - a `Product` record has an id column and an Order record also contains an id column
  - If we create a new table called `ProductOrders` with a `product_id` and `order_id` column, we can associate a product with as many orders as we want and vice versa

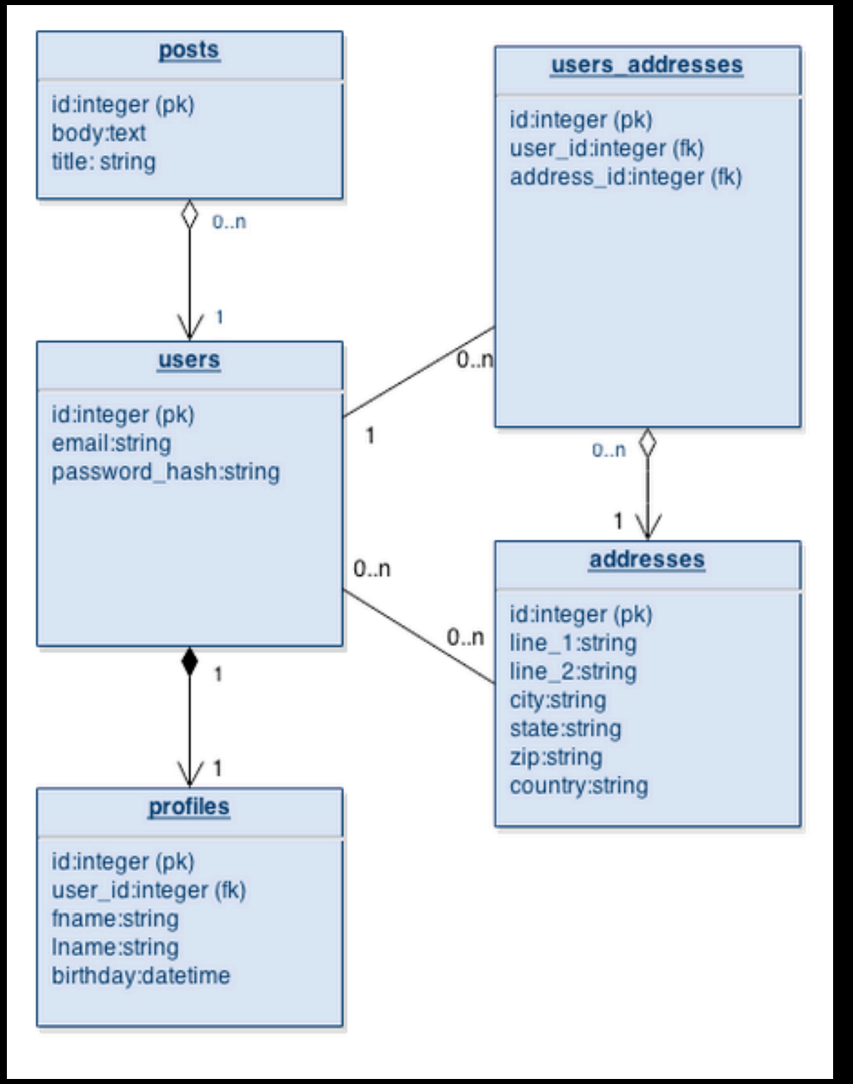
A full database diagram

- ☞ A “full” db diagram has all of your tables with their columns and data types
- ☞ The relationships between tables are illustrated with arrows

1 → 0..n = 1 to many

1 → 1 = 1 to one

0..n → 0..n = many to many



Exercise

☞ Let’s build a database diagram for an existing major website together!

Exercise

- ☞ Develop a database diagram for one of your favorite websites.
- ☞ Start with the users table and go from there!
- ☞ Be sure to indicate the relationships between different tables in the database
- ☞ You could use www.draw.io or a pen and paper

A final word/summary

A **database** is composed of **tables**. Each **table** has **columns** and can be **related** to other tables in the database. Each column has a **data type**. Each table also has **rows** of data which correspond to the **column names** and data types described in the **schema**.

And they all lived happily ever after!