GENERAL ASSEMBLY

INTRO TO DATA SCIENCE

LECTURE 2: DATABASES, SQL, PYTHON, PANDAS

I. WHAT IS DATA SCIENCE?
II. THE DATA MINING WORKFLOW
III. WORKING AT THE UNIX COMMAND LINE

I. INTRO TO DATABASES

II. SQL (EXERCISES)

III. PYTHON (EXERCISES)

IF STILL TIME LEFT

IV. PANDAS (EXERCISES)

I. INTRO TO DATABASES

What is ETL?

- **E**xtract data
- **T**ransform data
- **L**oad data

Databases are a **structured** data source optimized for efficient **retrieval and storage**

Databases are a **structured** data source optimized for efficient **retrieval and storage**

structured: we'll have to define some pre-defined organization

Databases are a **structured** data source optimized for efficient **retrieval and storage**

structured: we'll have to define some pre-defined organization

e.g., a table with columns for first name, last name, DOB, address, etc.

Databases are a **structured** data source optimized for efficient **retrieval and storage**

structured: we'll have to define some pre-defined organization

retrieval: the ability to read data our

storage: the ability to write data and save it

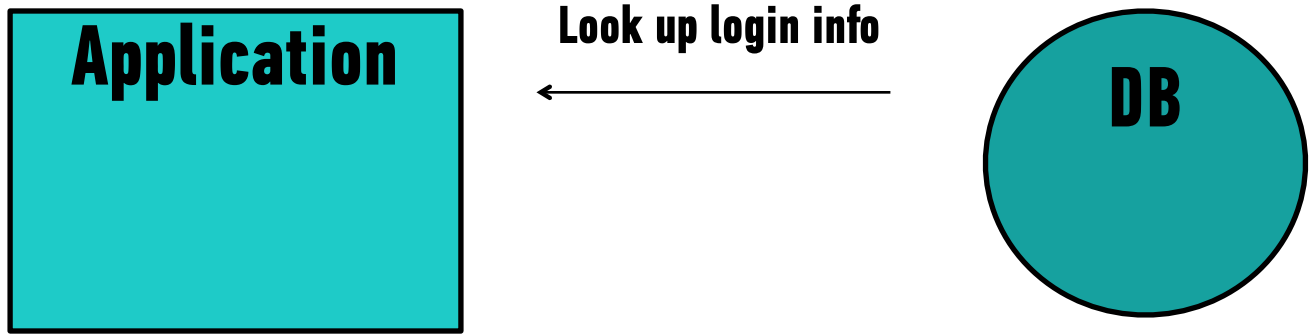
Databases are a **structured** data source optimized for efficient **retrieval and *persistent* storage**

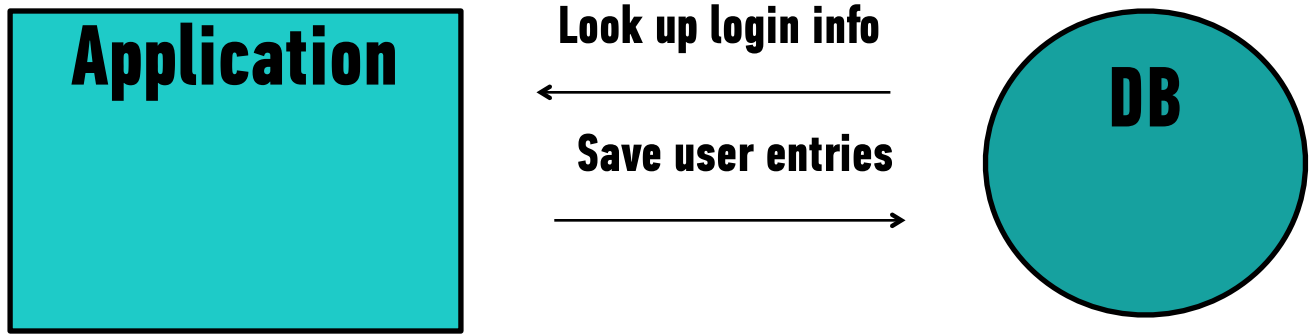
structured: we'll have to define some pre-defined organization

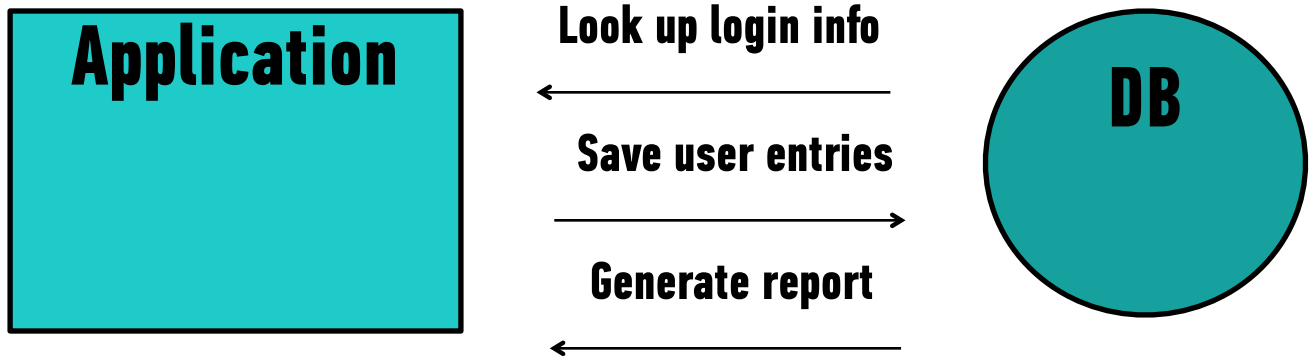
retrieval: the ability to read data our

storage: the ability to write data and save it


Application
DB


Application
Look up login info
DB


Application
Look up login info
Save user entries
DB


Application
Look up login info
Save user entries
Generate report
DB

A **relational database** is organized in the following manner:

- A database has **tables** which represent individual entities or objects
- Tables have a predefined **schema** - rules that tell it what columns exist and what they look like

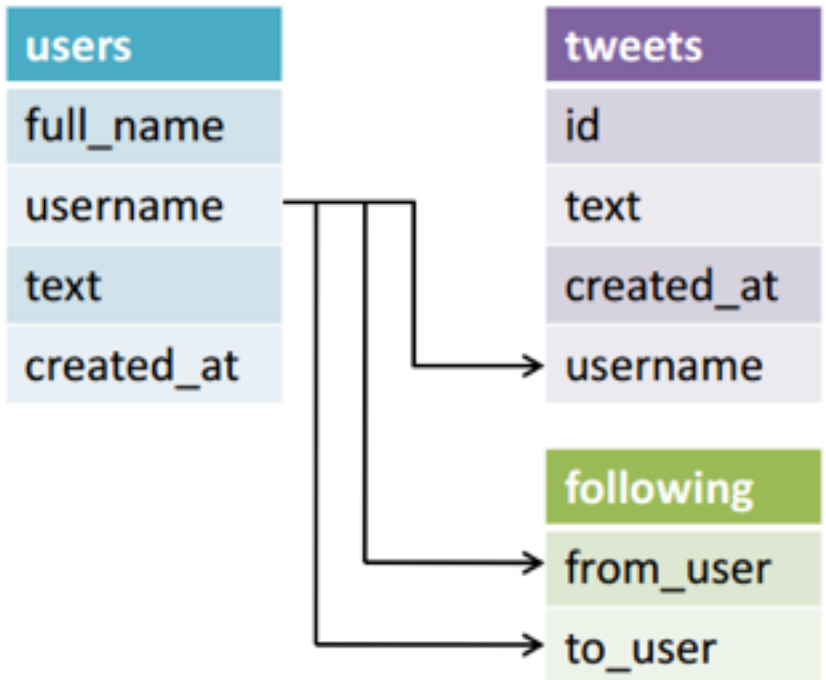
A **relational database** is organized in the following manner:

- table

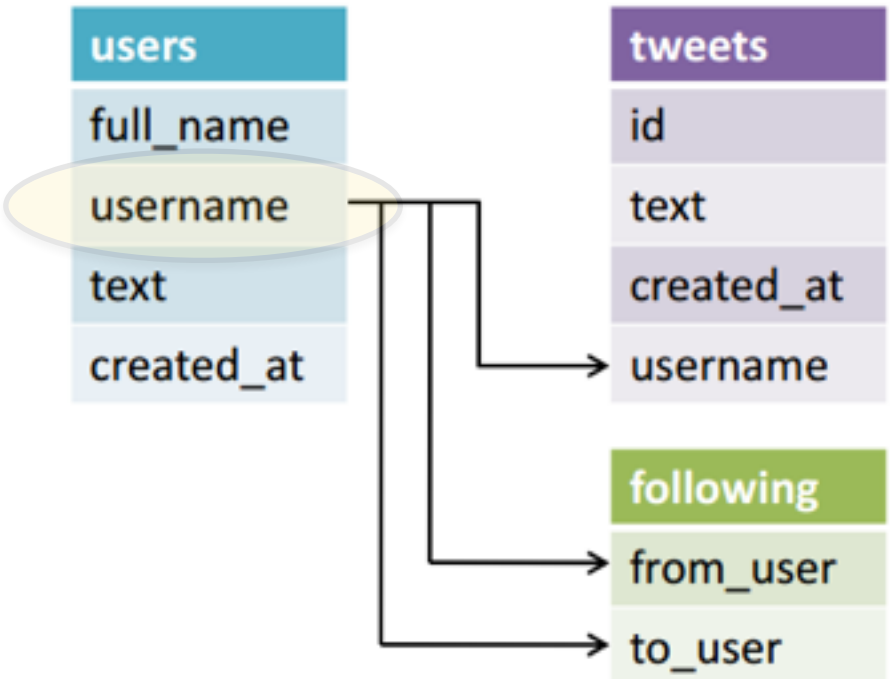
id	first_name	last_name	date_of_birth
312	Joe	Smith	1980-12-24
1532	Michelle	Anderson	1973-03-12

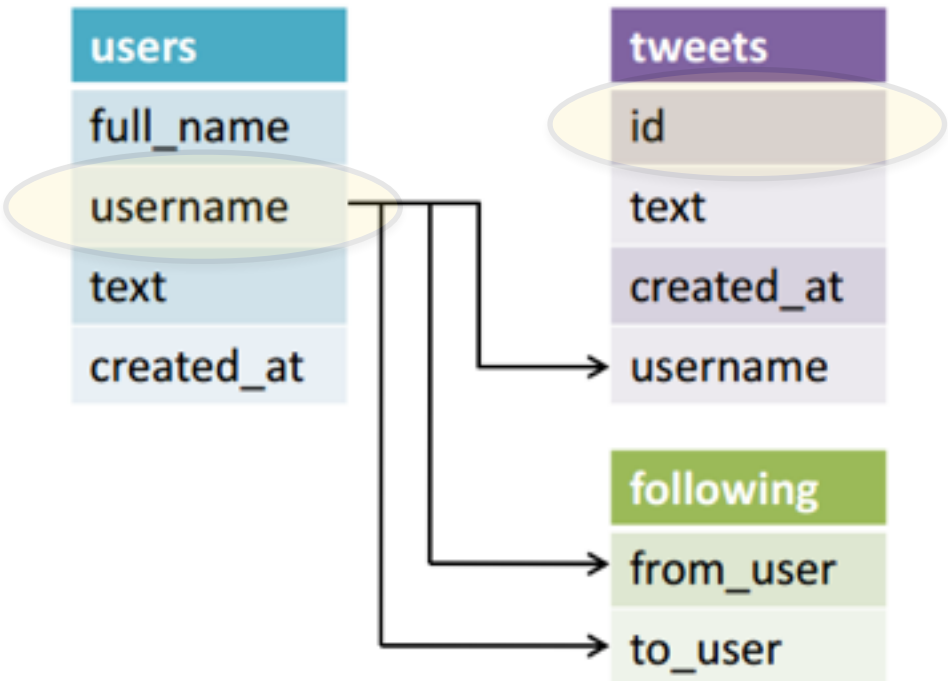
- schema

```
id              bigint
first_name      char(36)
last_name       char(36)
date_of_birth   timestamp
```


users
full_name
username
text
created_at
tweets
id
text
created_at
username
following
from_user
to_user

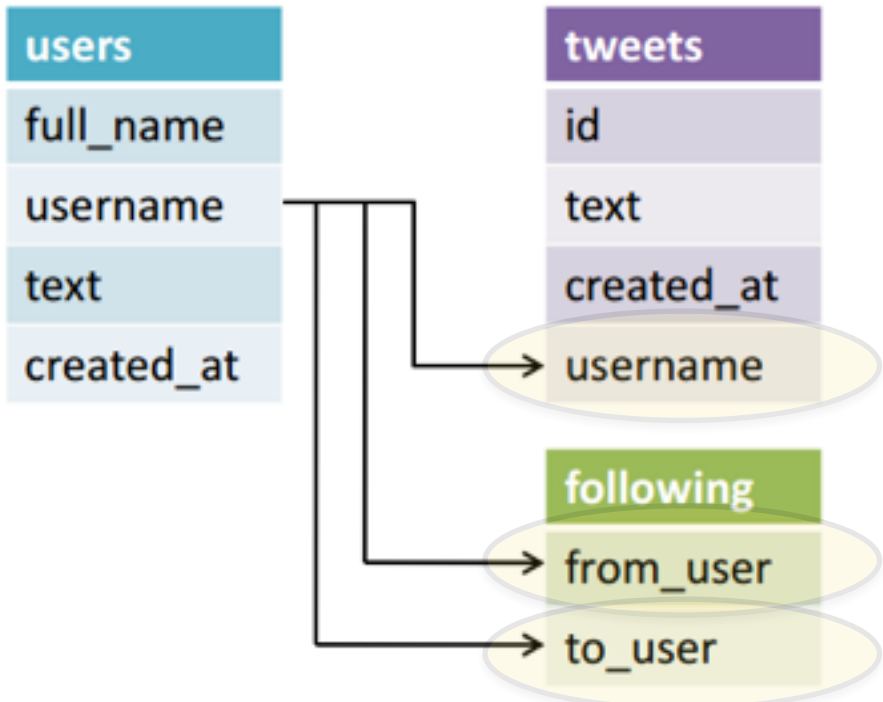
Each table should have a **primary key** column,
i.e., a unique identifier for that row


users
full_name
username
text
created_at
tweets
id
text
created_at
username
following
from_user
to_user


users
full_name
username
text
created_at
tweets
id
text
created_at
username
following
from_user
to_user

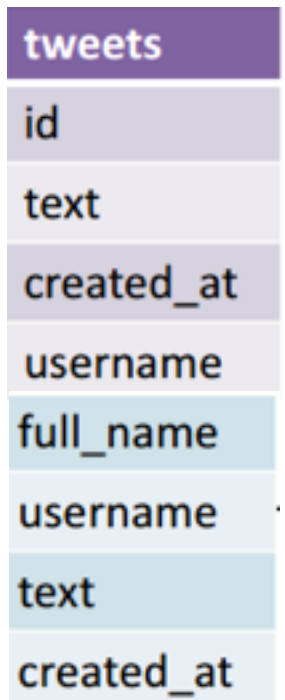
Each table should have a **primary key** column,
i.e., a unique identifier for that row

Additionally, each table can have a **foreign key** column,
i.e., an id that links this to table to another


users
full_name
username
text
created_at
tweets
id
text
created_at
username
following
from_user
to_user

We could have had a table structure as follow:

Why is this different?

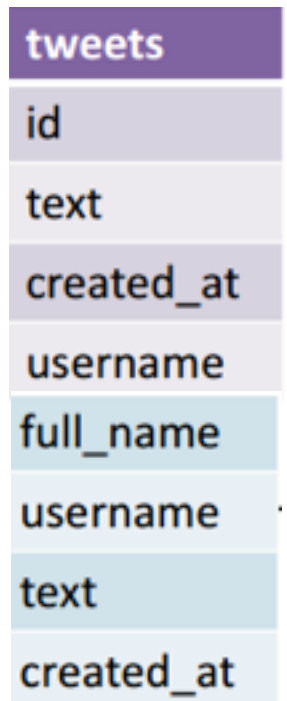


We could have had a table structure as follow:

Why is this different?

We would repeat the user information on each row.

This is called **denormalization**



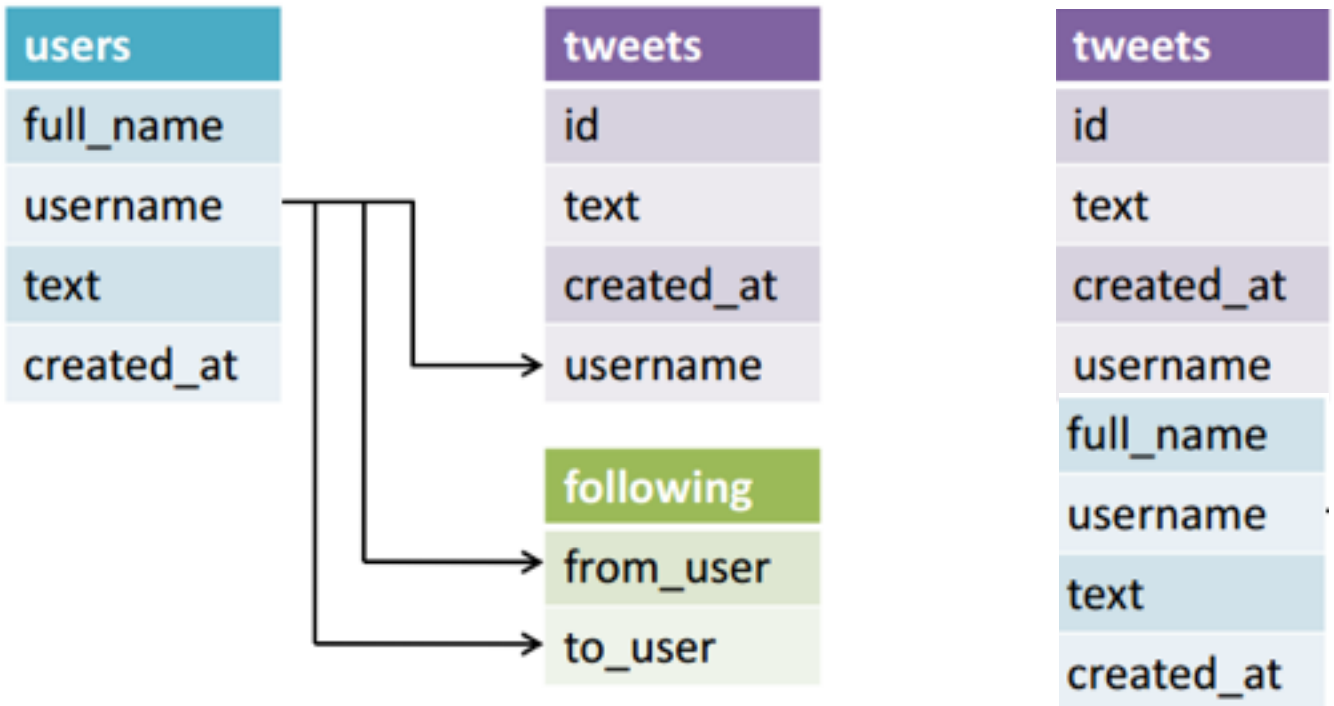
Normalized Data:

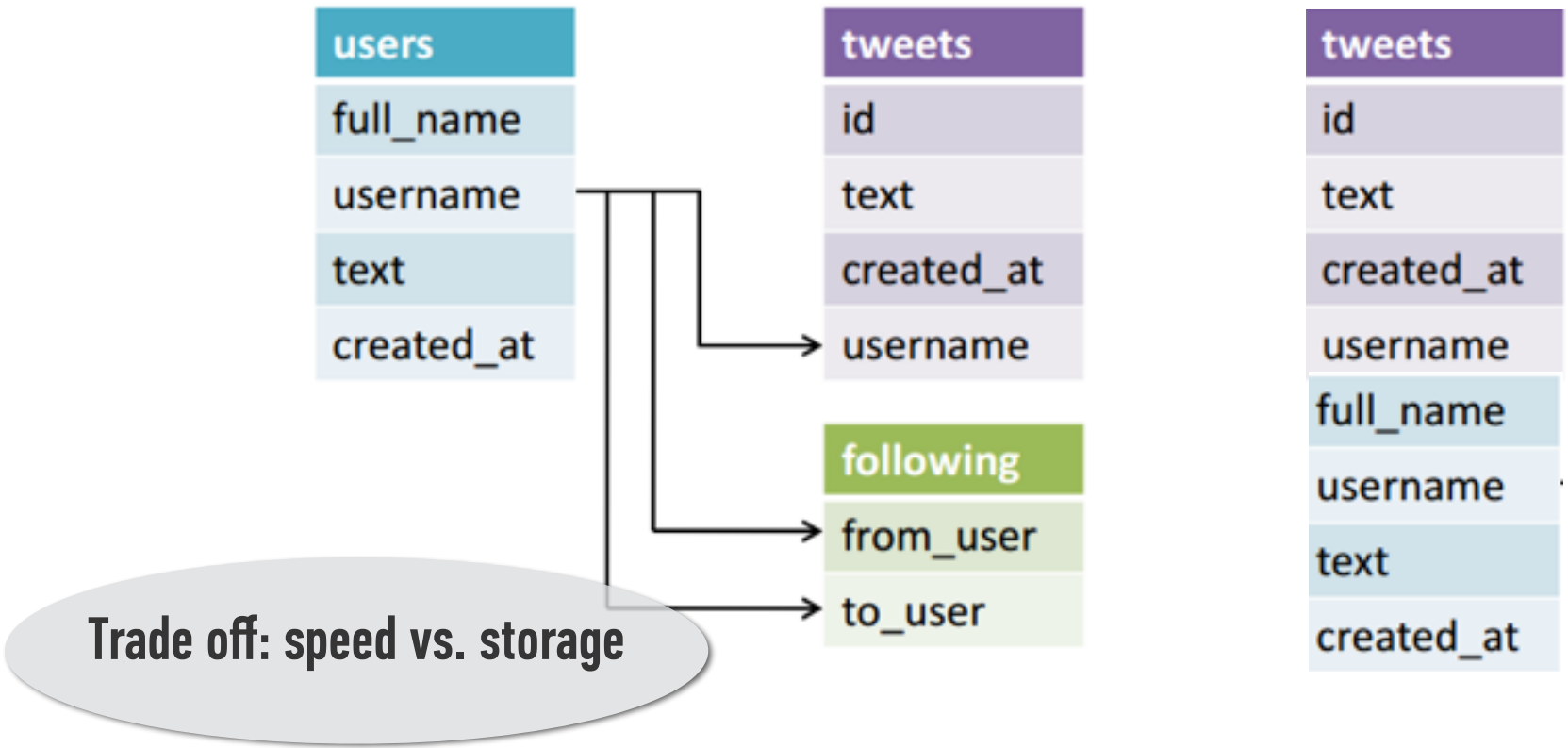
Many tables to reduce redundant or repeated data in a table

Denormalized Data:

Wide data, fields are often repeated

but removes the need to join together multiple tables


users
full_name
username
text
created_at
tweets
id
text
created_at
username
following
from_user
to_user
tweets
id
text
created_at
username
full_name
username
text
created_at


users
full_name
username
text
created_at
tweets
id
text
created_at
username
following
from_user
to_user
tweets
id
text
created_at
username
full_name
username
text
created_at
Trade off: speed vs. storage

Q: How do we commonly evaluate databases?

Q: How do we commonly evaluate databases?

- *read-speed vs. write speed*

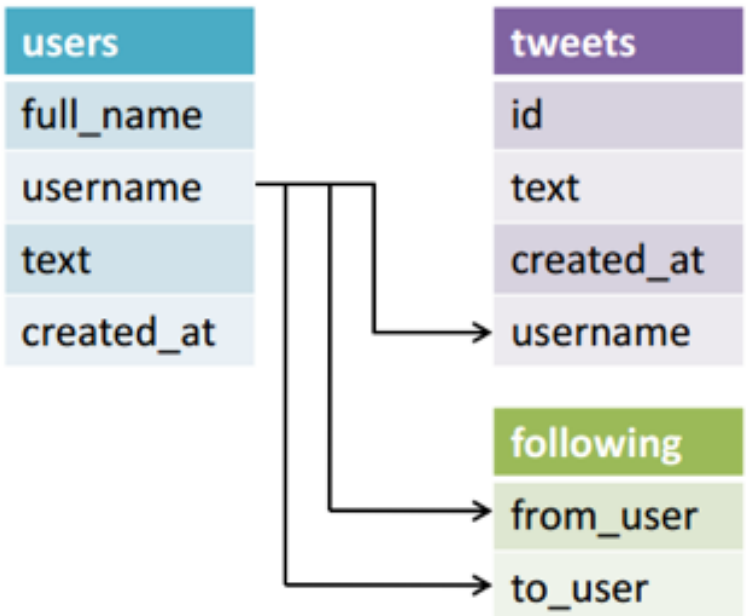
Q: How do we commonly evaluate databases?

- *read-speed vs. write speed*
- *space considerations*

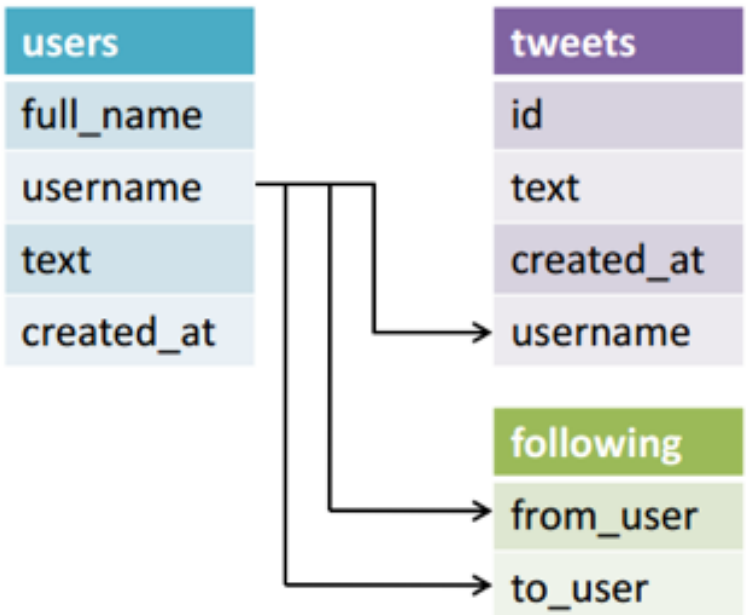
Q: How do we commonly evaluate databases?

- *read-speed vs. write speed*
- *space considerations*
- *(...and many other criteria)*

Q: Why are normalized tables (possibly) slower to read?

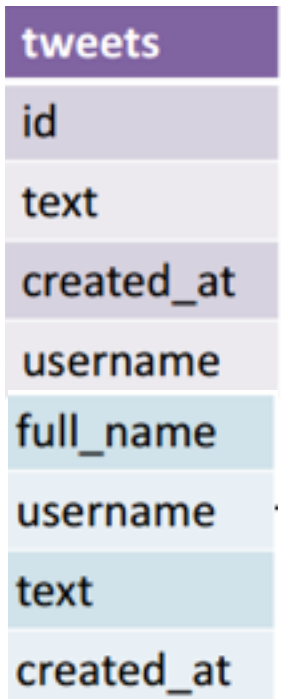


Q: Why are normalized tables (possibly) slower to read?

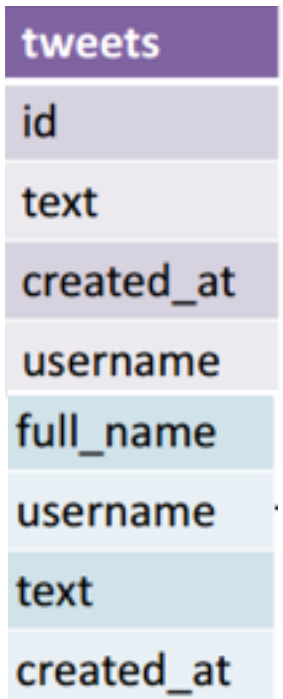


We'll have to get data from multiple tables to answer some questions

Q: Why are denormalized tables (possibly) slower to write?



Q: Why are denormalized tables (possibly) slower to write?



We'll have to write more data each time we store something

Databases are either relational or non-relational

- Relational: SQL (MySQL, PostgreSQL, ...)
- Non-relational: NoSQL (MongoDB, Cassandra, ...)

II. SQL (STRUCTURED QUERY LANGUAGE)

SQL (Structured Query Language) is a query language designed to extract, transform and load data in relational databases

SELECT: *Allows you to* ***retrieve*** *information from a table*

Syntax:

SELECT col1, col2 FROM table WHERE <some condition>

Example:
SELECT poll_title, poll_date FROM polls
WHERE romney_pct > obama_pct

GROUP BY: *Allows you to* ***aggregate*** *information from a table*

Syntax:

```
SELECT col1, AVG(col2) FROM table GROUP BY col1
```

Example:

```
SELECT poll_date, AVG(obama_pct) FROM polls
GROUP BY poll_date
```

GROUP BY: *Allows you to* ***aggregate*** *information from a table*

Syntax:

SELECT col1, AVG(col2) FROM table GROUP BY col1

Example:

SELECT poll_date, AVG(obama_pct) FROM polls
GROUP BY poll_date

GROUP BY: *Allows you to* ***aggregate*** *information from a table*

Syntax:

SELECT col1, AVG(col2) FROM table GROUP BY col1

There are usually a few common built-in operations:
SUM, AVG, MIN, MAX, COUNT

JOIN: *Allows you to* ***combine*** *multiple tables*

Syntax:

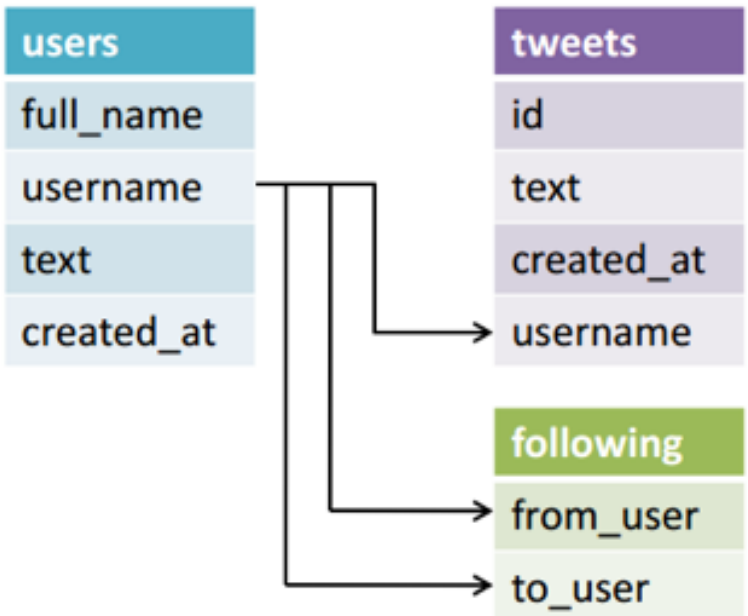
```
SELECT table1.col1, table1.col2, table2.col2
FROM  table1 JOIN table2 ON table1.col1 = table2.col2
```

JOIN: *Allows you to* ***combine*** *multiple tables*

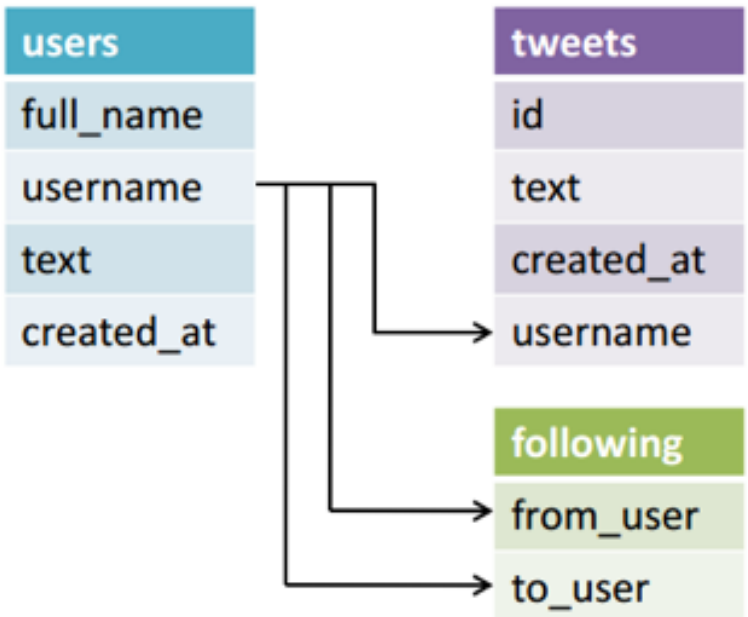
Syntax:

```
SELECT table1.col1, table1.col2, table2.col2
FROM  (JOIN table1, table2 ON table1.col1 = table2.col2)
```

Create a table with all the users and their tweets

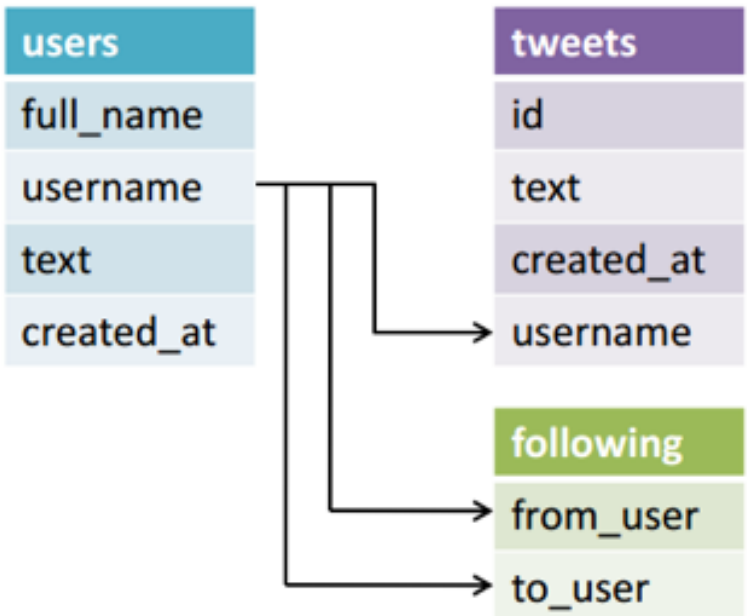


Create a table with all the users and their tweets



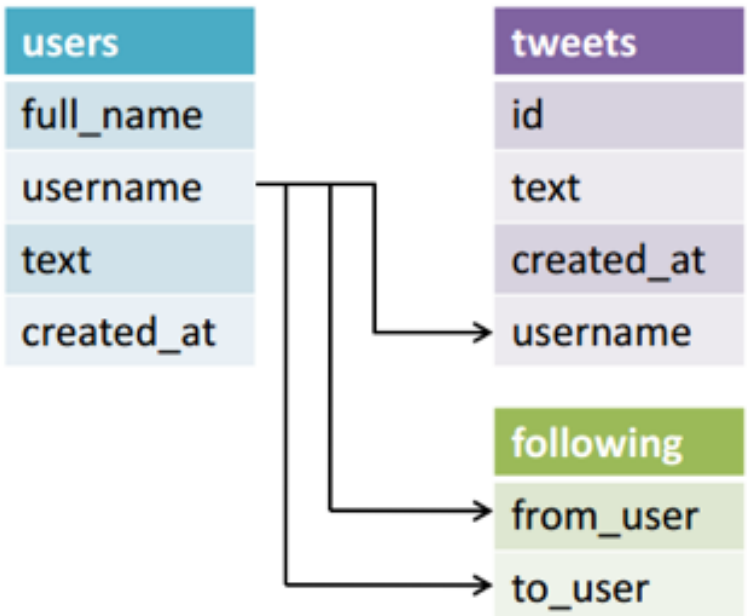
username	tweet
Joe	Hello, world!
Joe	Just tweetin'
Michelle	I am eating pizza tonight

Create a table with all the users and their tweets



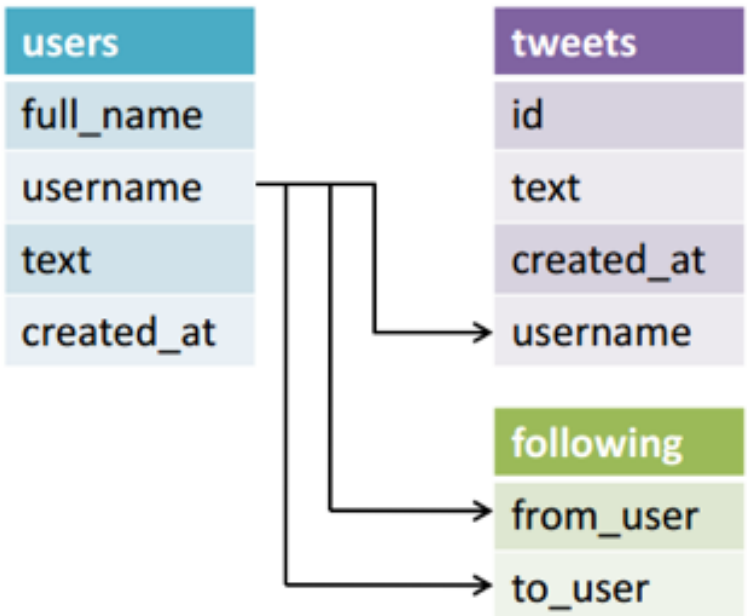

```
SELECT
  users.username,
  text
```

Create a table with all the users and their tweets



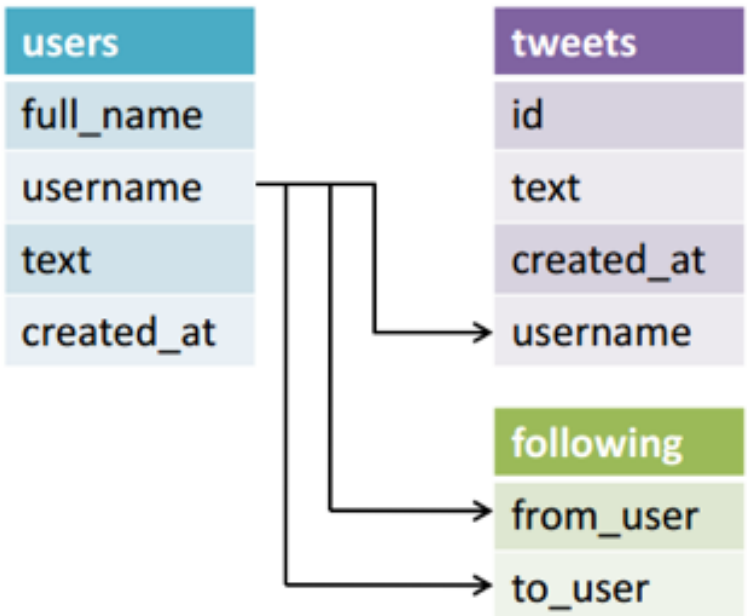

```
SELECT
  users.username,
  text
FROM users
```

Create a table with all the users and their tweets




```
SELECT
  users.username,
  text
FROM users
JOIN tweets
```

Create a table with all the users and their tweets




```
SELECT
  users.username,
  text
FROM users
JOIN tweets
ON users.username = tweets.username
```

INSERT: *Allows you to* ***add*** *data to tables*

Syntax:

```
INSERT INTO <table> (col1, col2)
VALUES( ...)
```

Example:

```
INSERT INTO classroom (first_name, last_name)
VALUES('John', 'Doe');
```

- Go to github.com/ga-students/DAT-23-NYC
- Scroll down to lesson #2 and click on SQL Exercises

III. PYTHON

Q: What is Python?

Q: What is Python?

A: An open source, high-level, dynamic scripting language.

Q: What is Python?

A: An open source, high-level, dynamic scripting language.

open source*: free! (both binaries and source files)*

Q: What is Python?

A: An open source, high-level, dynamic scripting language.

open source*: free! (both binaries and source files)*

high-level*: interpreted (not compiled)*

Q: What is Python?

A: An open source, high-level, dynamic scripting language.

open source*: free! (both binaries and source files)*

high-level*: interpreted (not compiled)*

dynamic*: things that would typically happen at compile time happen at runtime instead (e.g., dynamic typing)*

- Created by Guido van Rossum in 1991
- Benevolent Dictator for Life

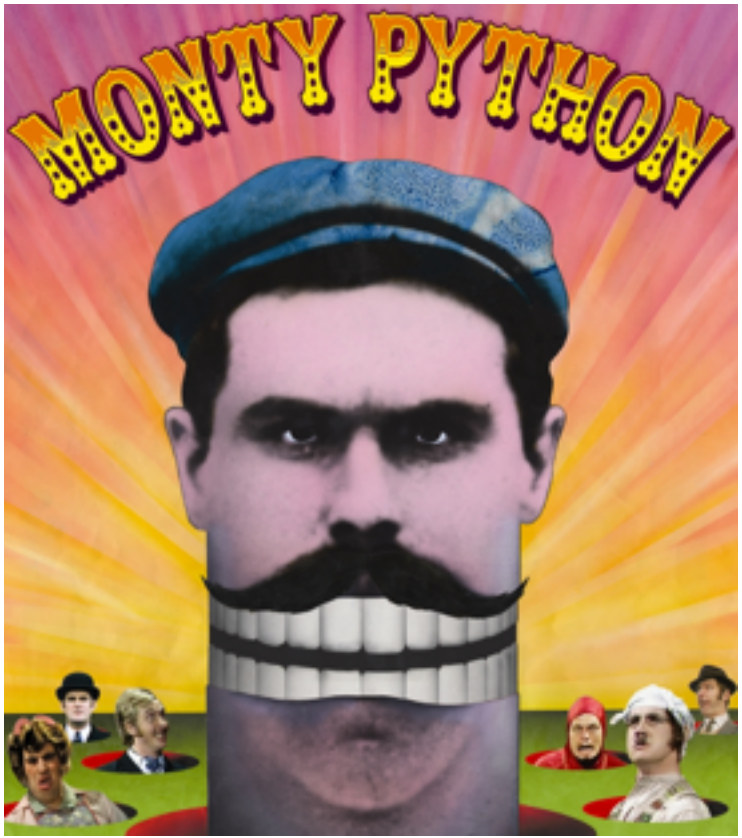
- Created by Guido van Rossum in 1991
- Benevolent Dictator for Life

- Currently on version 3 ...
  - but most still use 2.7+

- Created by Guido van Rossum in 1991
- Benevolent Dictator for Life

- Currently on version 3 ...
  - but most still use 2.7+

- Named after Monty Python
  - Still many references to TV show



WHY PYTHON?

WHAT ARE THE ADVANTAGES TO PYTHON?

- Easy to learn, easy to use
- Batteries Included: large collection of built-in libraries
- Simple and clean syntax

WHAT ARE THE ADVANTAGES TO PYTHON?

- Easy to learn, easy to use
- Batteries Included: large collection of built-in libraries
- Simple and clean syntax – very strict indent rules

WHAT ARE THE ADVANTAGES TO PYTHON?

- Easy to install new package: pip, easy_install

  Try:

  - > pip install oauth2
  - > pip install twitter

WHAT ARE THE ADVANTAGES TO PYTHON?

Java

```
public static void main( String args []) {
    System.out.println("Hello world");
}
```

WHAT ARE THE ADVANTAGES TO PYTHON?

Python

```
print "Hello World"
```

WHAT SETS PYTHON APART?

- Type system:
  - Dynamic typing!

WHAT IS TYPING?

- Need to tell the program WHAT something is:
  - Is it text: a string?
  - Is it numeric: an integer?
  - C, Java: double pi = 3.14...

```
double pi = 3.14;
```

- Can lead to hard to read to code

The most basic data structure is the **None** *type.*
This is the equivalent of NULL in other languages.

There are three basic numeric types:
int: (whole numbers)
float: (decimal numbers)
bool: (true or false)

```
>>> type(1)
<type 'int'>
>>> type(2.5)
<type 'float'>
>>> type(True)
<type 'bool'>
```

Python supports dynamic typing

```
>>> x = 1
>>> x
1
>>> x = 'horseshoe'
>>> x
'horseshoe'
>>> _
```

WHAT SETS PYTHON APART?

- Type system:
  - Dynamic typing!

- Interpreted language
  - No compilation

WHAT SETS PYTHON APART?

- Type system:
  - Dynamic typing!
- Interpreted language
  - No compilation

Python sounds amazing! What is it bad at?

Python sounds amazing! What is it bad at?

- Python is slower than a lower-level language
  (but keep in mind that this is a conscious tradeoff)

Python sounds amazing! What is it bad at?

- Python is slower than a lower-level language
  (but keep in mind that this is a conscious tradeoff)

- No compilation means discovery of errors at runtime

Python sounds amazing! What is it bad at?

- Python is slower than a lower-level language
  (but keep in mind that this is a conscious tradeoff)

- No compilation means discovery of errors at runtime

```
if year == “2016”:
    print This is an election year!”
```

Missing quotation mark will only be noticed when the print command will be executed

Python sounds amazing! What is it bad at?

- Python is slower than a lower-level language
  (but keep in mind that this is a conscious tradeoff)
- No compilation means discovery of errors at runtime
- Dynamic typing allows for bad practice

PYTHON SYNTAX

DATA TYPES

```
x = 36  # this is an integer
x = 3.14  # a decimal number
x = True  # either True or False
x = "This is a string"
```

DATA TYPES

```
x = [1, 2, 3, 4]  # a list

# lists can contain elements of any type
x = [36, 3.14, True, "This is a string"]
x = [36, 3.14, True, "This is a string", [1, 2, 3, 4]]

# elements are numbered, starting with 0 (!)
print x[0]  # will print first element
```

DATA TYPES

```
# dictionaries (maps)
x = {'name': 'Joe', 'age': 75}  # this is a dictionary
x = dict(name='Joe', age=75)  # same as above (old syntax)

print x['name']  # will print 'Joe'
```

IF/ELSE STATEMENTS

- Allow us to take different paths through depending on some condition

```
x = 5
if x > 4:
    print "This number is greater than 4"
```

IF/ELSE STATEMENTS

- Allow us to take different paths through depending on some condition

```
x = 5
if x > 4:
    print "This number is greater than 4"
else:
    print "This number is not greater than 4"
```

IF/ELSE STATEMENTS

- Allow us to take different paths through depending on some condition

```
x = 5
if x > 4:
    print "This number is greater than 4"
elif x == 4:
    print "This number is equal to 4"
else:
    print "This number is smaller than 4"
```

LOOPING – FOR

- Allows us to perform the same operation on each element, one by one

```
emotions = ["happy", "sad", "¯\_(ツ)_/¯"]
for state in emotions:
    print "I feel", state
    if state == "happy":
        print "Happy is good, hooray!"
```

LOOPING – FOR

- Allows us to perform the same operation on each element, one by one

```
emotions = ["happy", "sad", "¯\_(ツ)_/¯"]
for state in emotions:
    print "I feel", state
    if state == "happy":
        print "Happy is good, hooray!"
```

```
I feel happy
Happy is good, hooray!
I feel sad
I feel ¯\_(ツ)_/¯
```

LOOPING – WHILE

- Allows us to perform the same operation on each element, one by one

```
emotions = ["happy", "sad", "¯\_(ツ)_/¯"]
while len(emotions) > 0:
    state = emotions.pop()
    if state == "happy":
        print "Happy is good, hooray!"
```

LOOPING – WHILE

- Allows us to perform the same operation on each element, one by one

```
emotions = ["happy", "sad", "¯\_(ツ)_/¯"]
while len(emotions) > 0:
    state = emotions.pop()
    if state == "happy":
        print "Happy is good, hooray!"
```

```
I feel ¯\_(ツ)_/¯
I feel sad
I feel happy
Happy is good, hooray!
```

FUNCTIONS

- Allow us to save some piece of code to reuse later

```
def greater_than_four(x):
    if x > 4:
        print "This number is greater than 4"

greater_than_four(6)
```

OPERATIONS

- Python shell is just a complex calculator:

```
>>> 3 + 4
7
>>> 1 / 2
0
>>> 1 / 2.
0.5
>>> 3 ** 2
9
```

OPERATIONS

- Python shell is just a complex calculator:

```
>>> ['A', 'B'] + ['A', 'C']
['A', 'B', 'A', 'C']
>>> ['A'] * 5
['A', 'A', 'A', 'A', 'A']
>>> 'A' * 5
'AAAAA'
>>> list('ABCDEF')
['A', 'B', 'C', 'D', 'E', 'F']
```

- Go to github.com/ga-students/DAT-23-NYC
- Scroll down to lesson #2 and click on Python Exercises