GENERAL ASSEMBLY

INTRO TO DATA SCIENCE

LECTURE 2: ETL AND DATA STORAGE

AGENDA

I. INTRO TO PYTHON

II. RELATIONAL DATABASES

III. NOSQL DATABASES

I. INTRO TO PYTHON

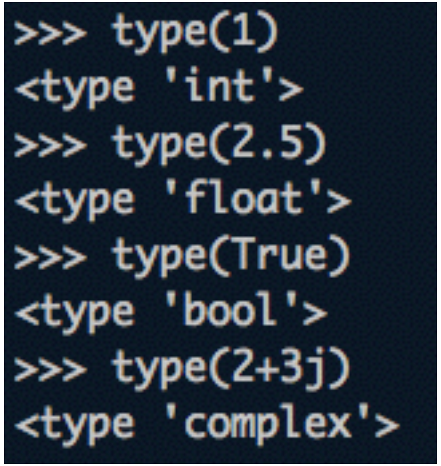
SETTING UP VARIABLES

- Python shell is just a complex calculator:
    - 10 * 15

    - x = 5
    - x #prints 5
    - x^2 #prints 25

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

There are four basic numeric types: **int, float, bool, complex, string**



```
>>> type(1)
<type 'int'>
>>> type(2.5)
<type 'float'>
>>> type(True)
<type 'bool'>
>>> type(2+3j)
<type 'complex'>
```

DATA TYPES

- Lists:
  - l = [1, 2, 3]
  - l = [‘happy’, ‘sad’, ‘indifferent’]

- Dictionaries (Maps):
  - Key-Value datastructure
  - d = { ‘first_name’ : ‘Arun’, ‘last_name’: ‘Ahuja’}

IF/ELSE STATEMENTS

- If/Else statements allow us to take different paths through depending on some condition:
- x = 5
- if x > 4:
    - print “This number was less than 4”

LOOPING

- Looping allows us to pass through some set of values and perform an operation on each

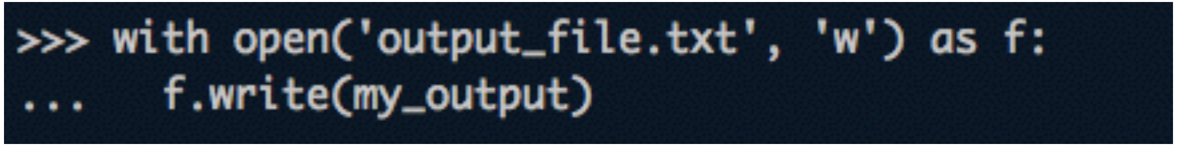
- l = [“happy”, “sad”, “don’t care”]
- for x in l:
    - print x
    - if x == ‘happy’:

FUNCTIONS

- Functions allow us to save some piece of functionality to reuse later

- def func(x):
    - if x > 4:
        - print “This number is less than 4

Our final example of a data type is the Python **file object**. *This represents an open connection to a file (eg) on your laptop.*

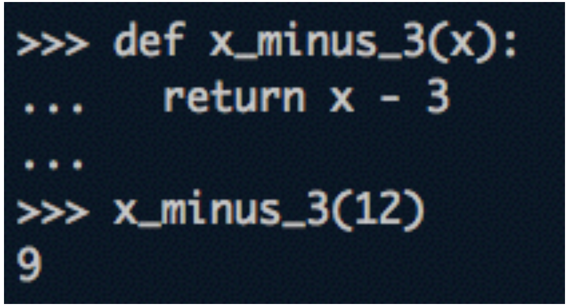


```
>>> with open('output_file.txt', 'w') as f:
...     f.write(my_output)
```

These are particularly easy to use in Python, especially using the with statement context manager, which automatically closes the file handle when it goes out of scope.

FUNCTIONS

Python allows you to define custom **functions** *as you would expect:*

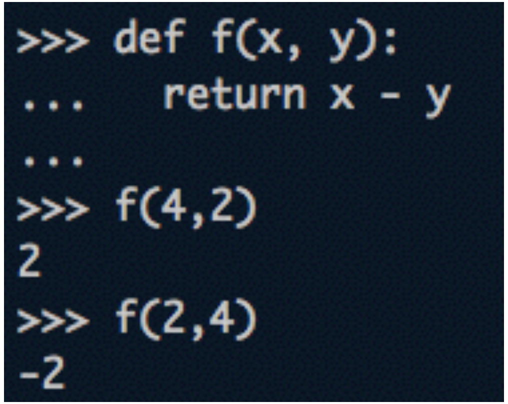


```
>>> def x_minus_3(x):
...     return x - 3
...
>>> x_minus_3(12)
9
```

Functions can optionally return a value with a **return statement** *(as this example does).*

Functions can take a number of **arguments** *as inputs, and these arguments can be specified in two ways:*

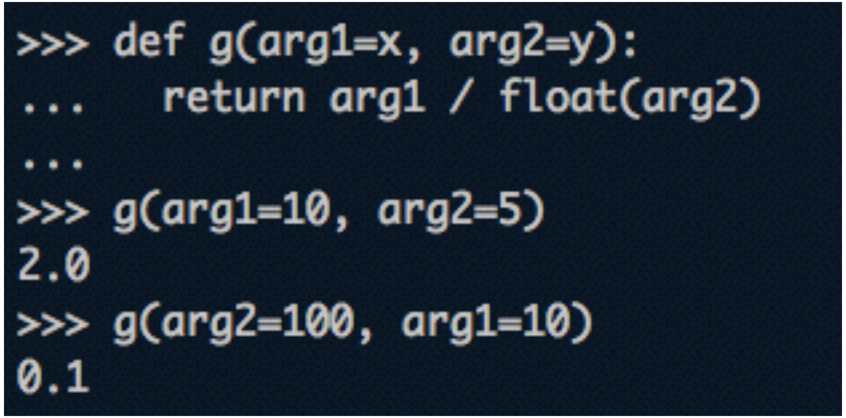
As **positional arguments***:*



```
>>> def f(x, y):
...     return x - y
...
>>> f(4,2)
2
>>> f(2,4)
-2
```

Functions can take a number of **arguments** *as inputs, and these arguments can be specified in two ways:*

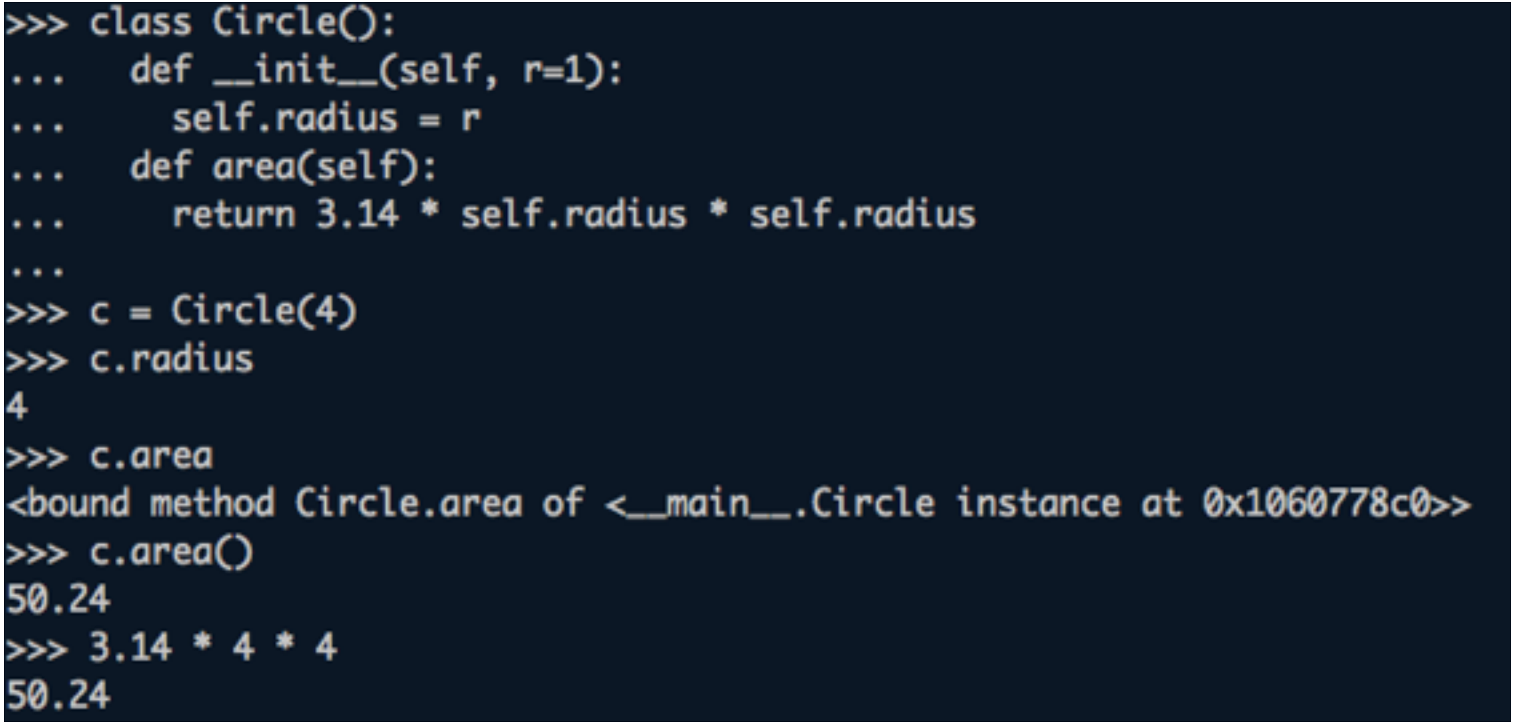
Or as **keyword arguments***:*



```
>>> def g(arg1=x, arg2=y):
...     return arg1 / float(arg2)
...
>>> g(arg1=10, arg2=5)
2.0
>>> g(arg2=100, arg1=10)
0.1
```

CLASSES

Python supports **classes** *with member attributes and functions:*



```
>>> class Circle():
...   def __init__(self, r=1):
...     self.radius = r
...   def area(self):
...     return 3.14 * self.radius * self.radius
...
>>> c = Circle(4)
>>> c.radius
4
>>> c.area
<bound method Circle.area of <__main__.Circle instance at 0x1060778c0>>
>>> c.area()
50.24
>>> 3.14 * 4 * 4
50.24
```

II. 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 will have to define some pre-defined organization strategy

retrieval : the ability to read data out

storage: the ability to write data and save it

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

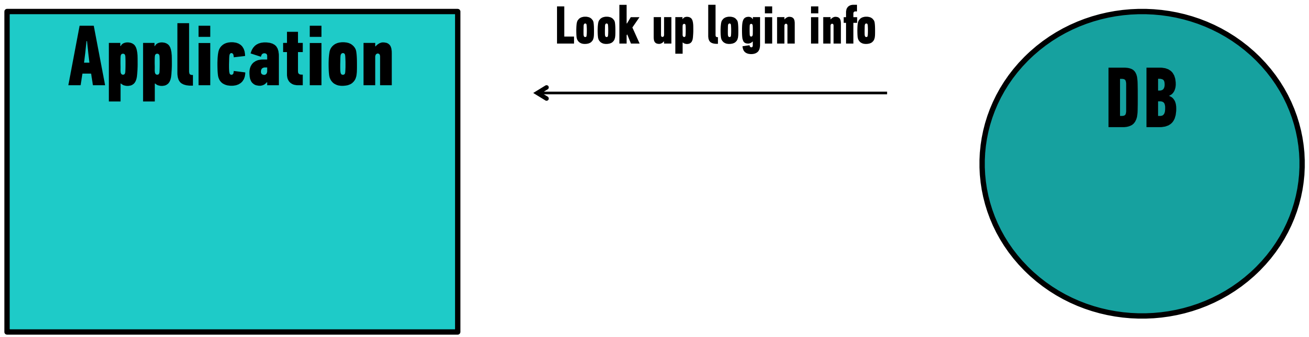
structured : we will have to define some pre-defined organization strategy

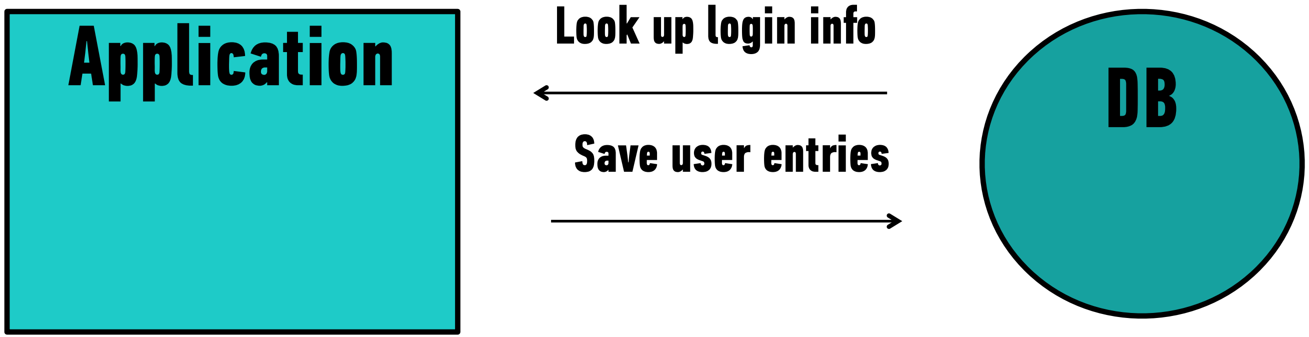
retrieval : the ability to read data our

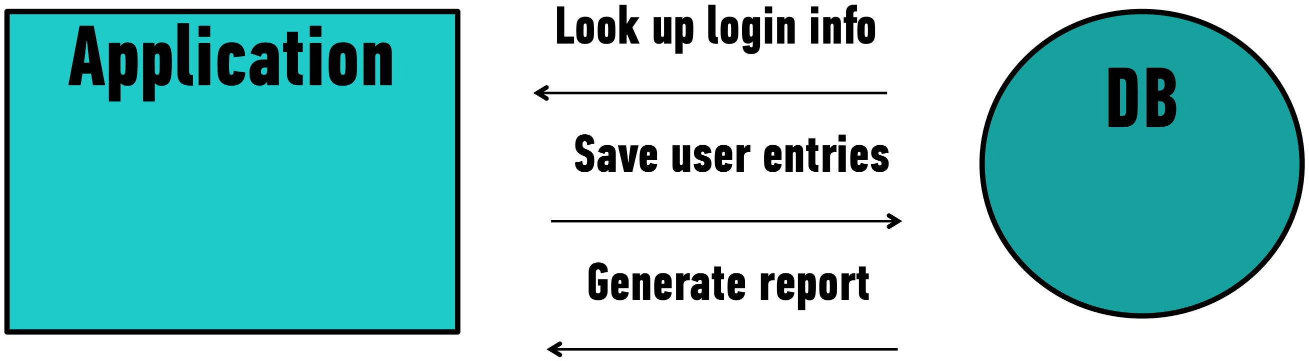
storage: the ability to write data and save it

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

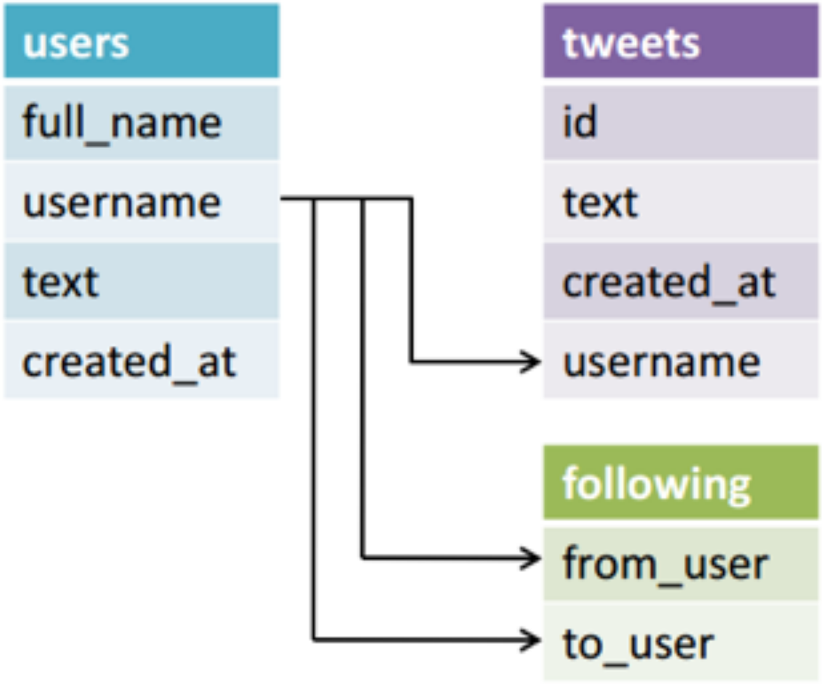
II. RELATIONAL DATABASES

Relational database are traditionally 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*

RELATIONAL DATABASES



users

full_name

username

text

created_at

tweets

id

text

created_at

username

following

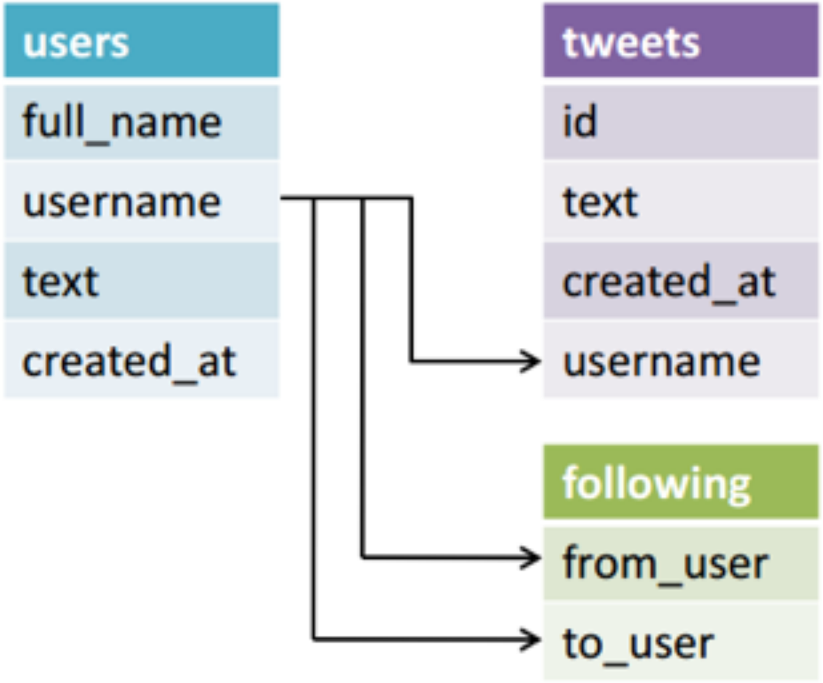
from_user

to_user

RELATIONAL DATABASES

Each table should have a **primary key** column- a unique identifier for that row

RELATIONAL DATABASES



users

- full_name
- username
- text
- created_at

tweets

- id
- text
- created_at
- username

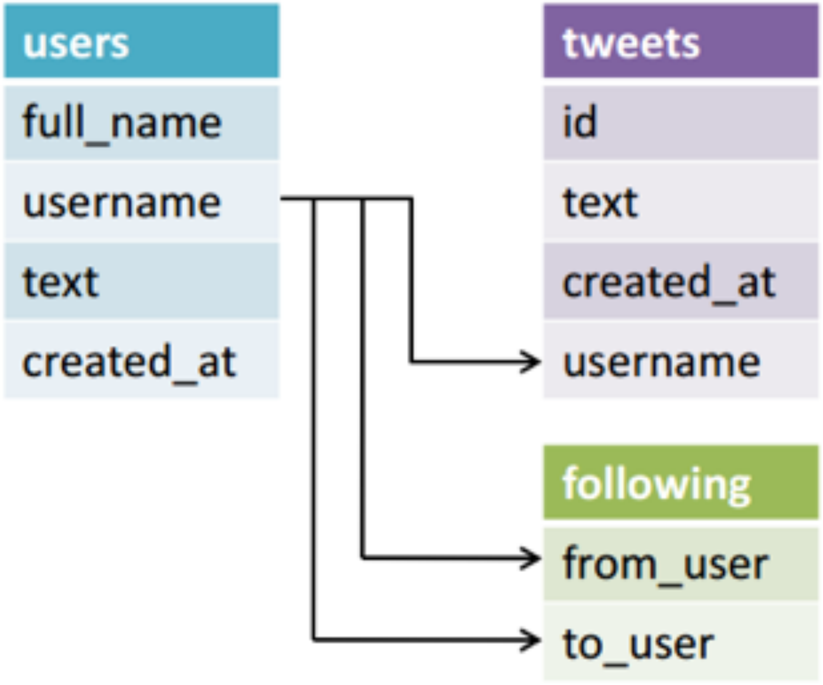
following

- from_user
- to_user

*Each table should have a **primary key** column- a unique identifier for that row*

*Additionally each table can have a **foreign key** column- an id that links this to table to another*

RELATIONAL DATABASES



users

- full_name
- username
- text
- created_at

tweets

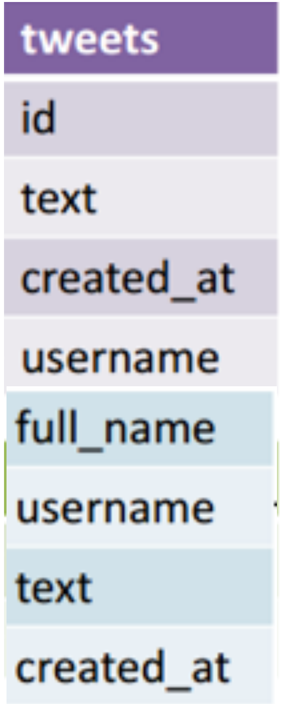
- id
- text
- created_at
- username

following

- from_user
- to_user

We could have had a table structure as follow:

Why is this different?



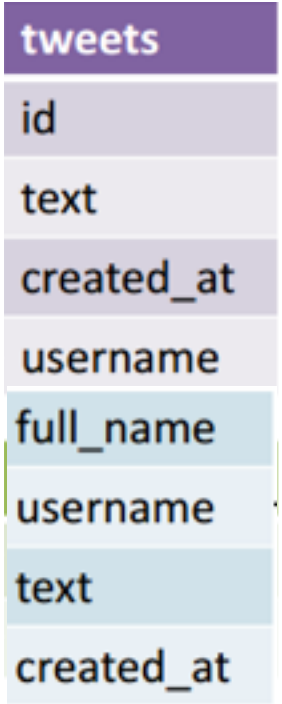
tweets
id
text
created_at
username
full_name
username
text
created_at

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



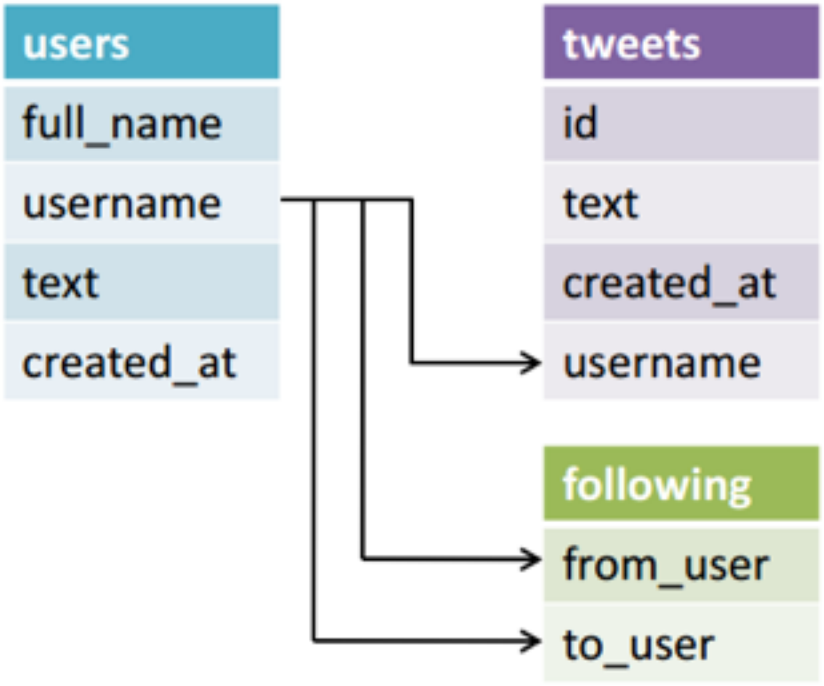
tweets
id
text
created_at
username
full_name
username
text
created_at

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

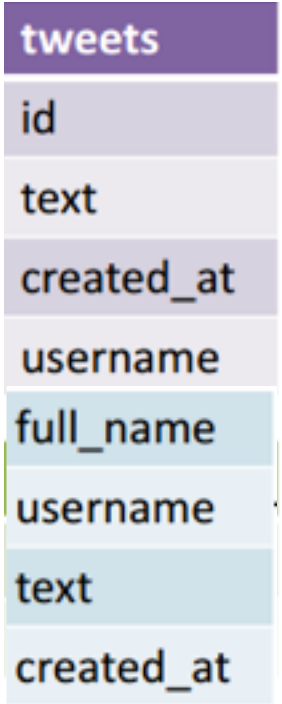
RELATIONAL DATABASES



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

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

Trade off of speed vs. storage

Q: How do we commonly evaluate databases?

read-speed vs. write speed

Q: How do we commonly evaluate databases?

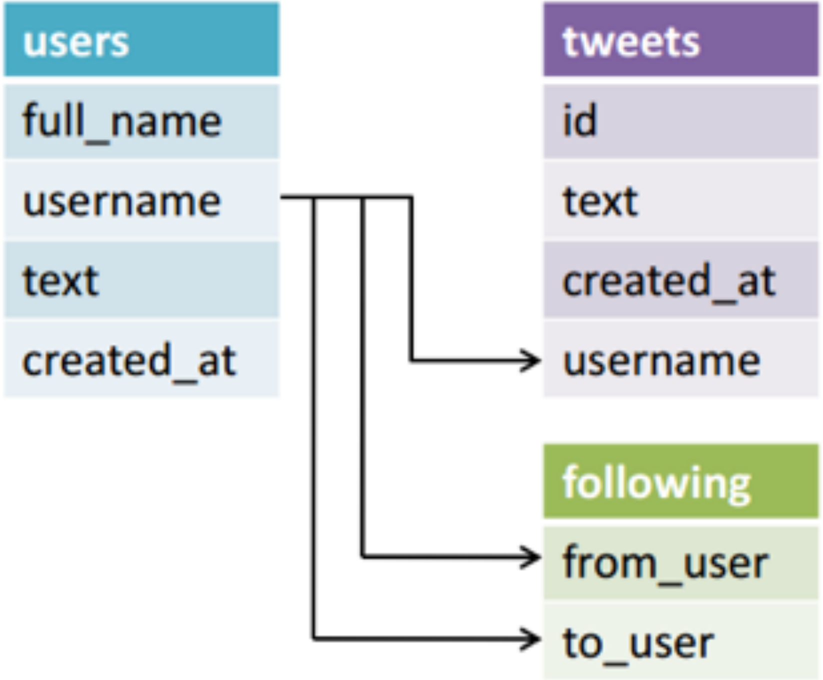
read-speed vs. write speed

space considerations

(...and many other criteria)

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

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



users
full_name
username
text
created_at

tweets
id
text
created_at
username

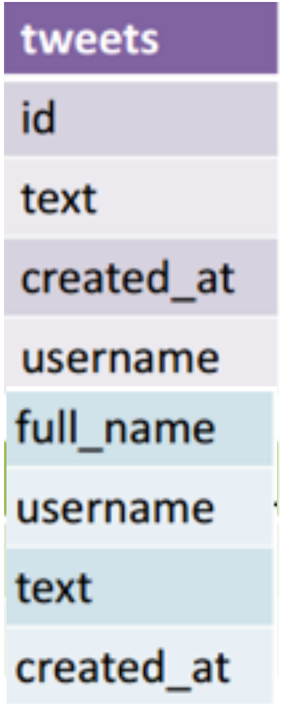
following
from_user
to_user

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

A: 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?



tweets
id
text
created_at
username
full_name
username
text
created_at

NORMALIZED VS DENORMALIZED

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

A: We’ll have to write more information on each write

SQL is a query language to load, retrieve and update data in relational databases

THE SELECT COMMAND

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)
```

THE INSERT COMMAND

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

Syntax and Example:

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

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

II. NO-SQL DATABASES

NO-SQL databases are a new trend in databases

NO-SQL databases are a new trend in databases

*The title **NOSQL** refers to the lack of a relational structure between stored objects*

NO-SQL databases are a new trend in databases

The title ***NOSQL*** *refers to the lack of a relational structure between stored objects*

Most importantly, they often attempt to minimize the need for ***JOIN*** *operations*

POPULAR NOSQL DATABASES

Memcached

Apache HBase

Cassandra

MongoDB

POPULAR NOSQL DATABASES

Memcached	::	*LiveJournal*
Apache HBase	::	*Google BigTable*
Cassandra	::	*Amazon Dynamo*
MongoDB		

Memcached was:

- *developed by LiveJournal*
- *distributed key-value store (HashMap or Python Dict)*
- *Support two operations:*

  get** and **set

Memcached was:

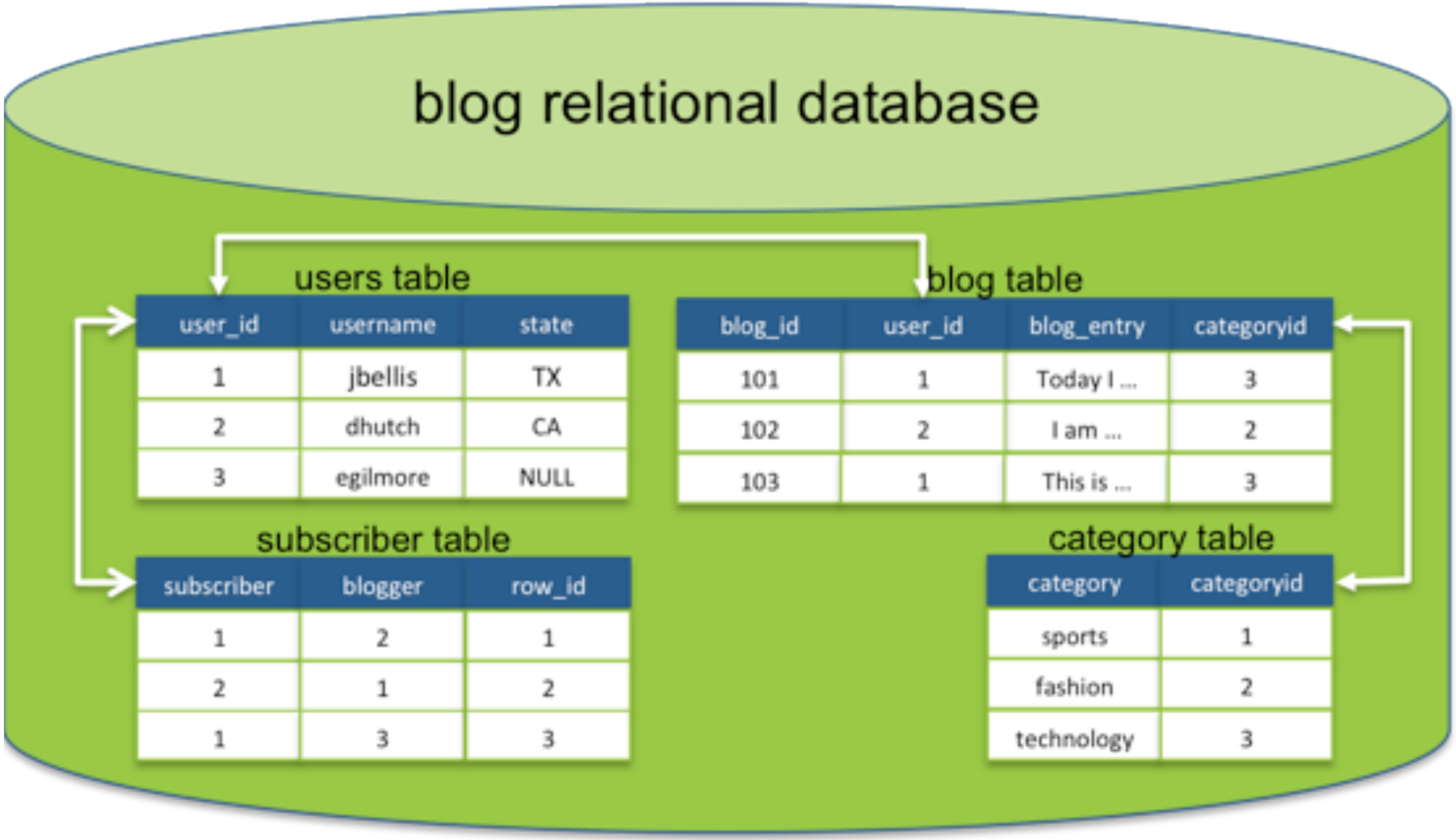
- *developed by LiveJournal*
- *distributed key-value store (HashMap or Python Dict)*
- *Support two **very fast** operations:*

    get** and **set

Cassandra was

- *developed by Facebook*
- *Messages application and Inbox Search*
- *Key-Value* ***(-ish)***
    - *supports query by key or key range*
- *Very fast writing speeds*
- *Useful for record keeping, logging*

APACHE CASSANDRA



users table

user_id	username	state
1	jbellis	TX
2	dhutch	CA
3	egilmore	NULL

blog table

blog_id	user_id	blog_entry	categoryid
101	1	Today I ...	3
102	2	I am ...	2
103	1	This is ...	3

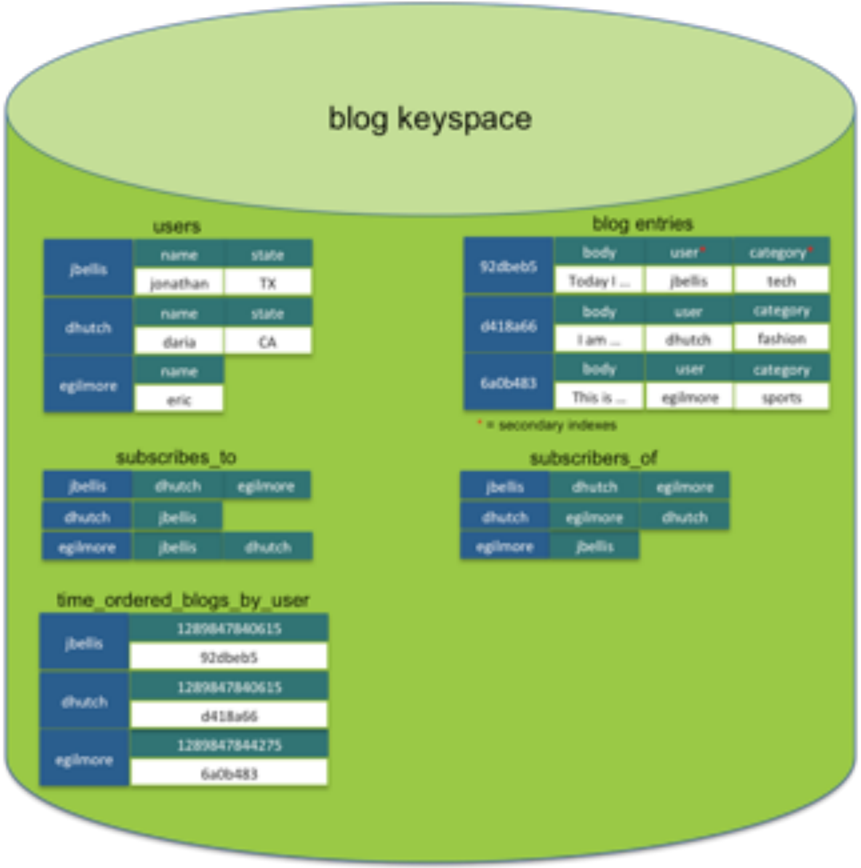
subscriber table

subscriber	blogger	row_id
1	2	1
2	1	2
1	3	3

category table

category	categoryid
sports	1
fashion	2
technology	3

APACHE CASSANDRA



blog keyspace

users

jbellis	name	state
	jonathan	TX
dhutch	name	state
	daria	CA
egilmore	name	
	eric	

blog entries

92dbeb5	body	user*	category*
	Today I ...	jbellis	tech
d418a66	body	user	category
	I am ...	dhutch	fashion
6a0b483	body	user	category
	This is ...	egilmore	sports

* = secondary indexes

subscribes_to

jbellis	dhutch	egilmore
dhutch	jbellis	
egilmore	jbellis	dhutch

subscribers_of

jbellis	dhutch	egilmore
dhutch	egilmore	dhutch
egilmore	jbellis	

time_ordered_blogs_by_user

jbellis	1289847840615
	92dbeb5
dhutch	1289847840615
	d418a66
egilmore	1289847844275
	6a0b483