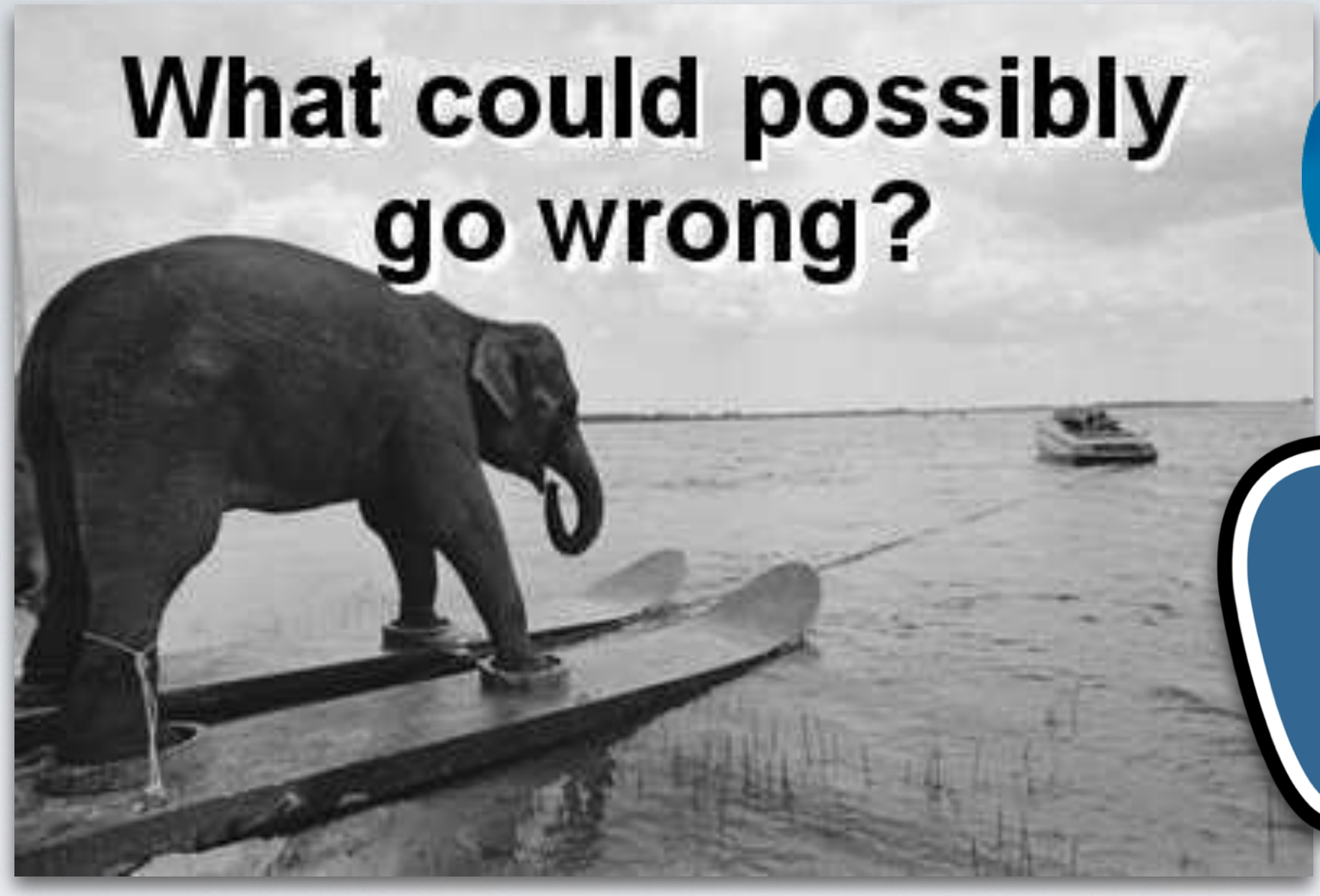
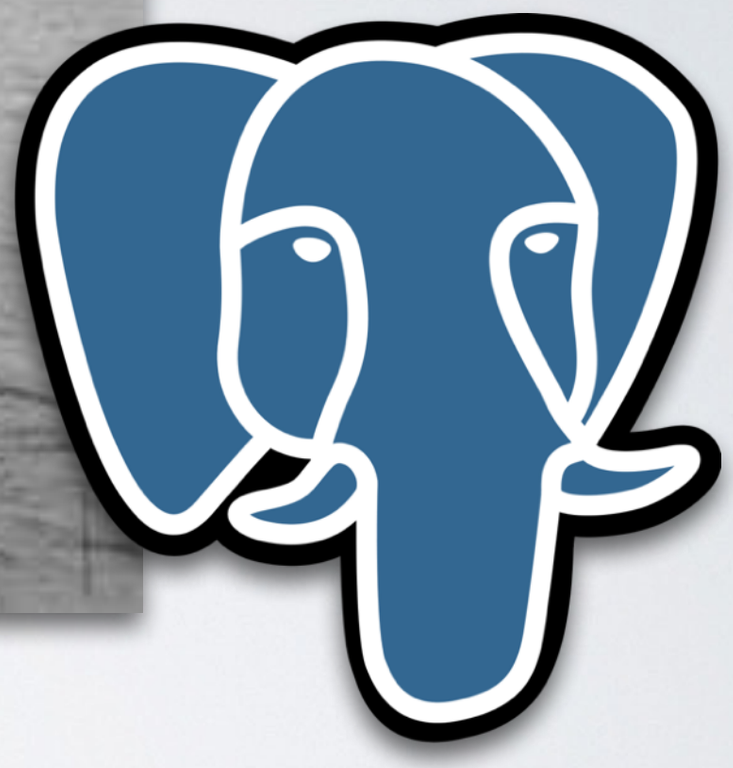




POSTGRESQL AND PYTHON

… plus sqlalchemy, alembic, flask, py-test, and friends

WHAT IS POSTGRESQL?

- Open Source Database
- Uses unique MVCC Architecture (Multi-Version Concurrency Control)
- Great SQL compliance, but didn't start with SQL
- https://www.postgresql.org/about/

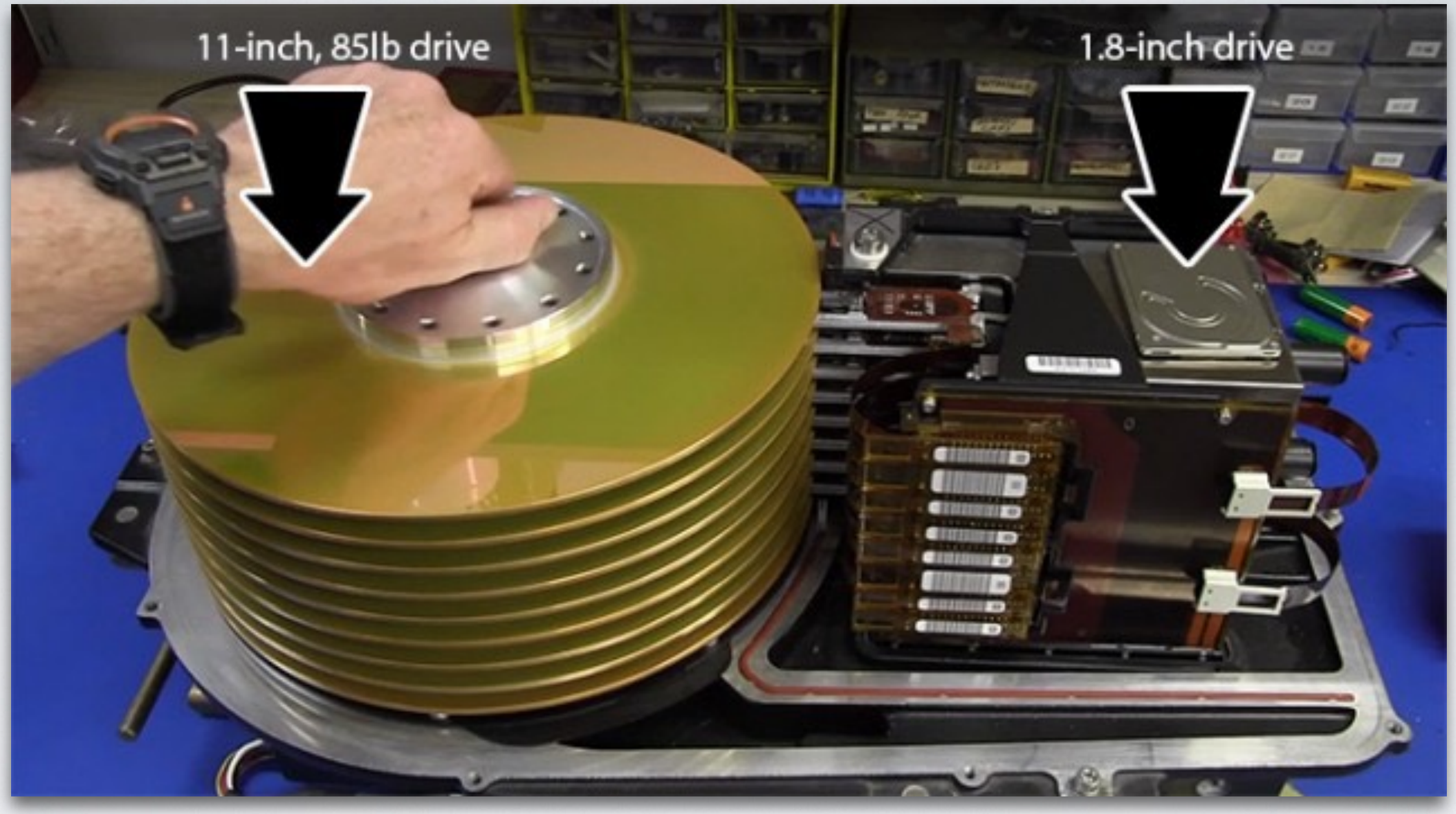
WHAT IS POSTGRESQL?

- Maximum DB Size: Unlimited
- Maximum Table Size: 32 TB
- Maximum Row Size: 1.6 TB
- Maximum Field Size: 1 GB
- Maximum Rows Per Table: Unlimited
- Maximum Columns Per Table: 250-1600 depending on types
- Maximum Indexes Per Table: Unlimited

Database


ED80
042
ED42
42
ICL
EXCHANGEABLE DISC DRIVE
ED50
50
ED40
40
48

Database

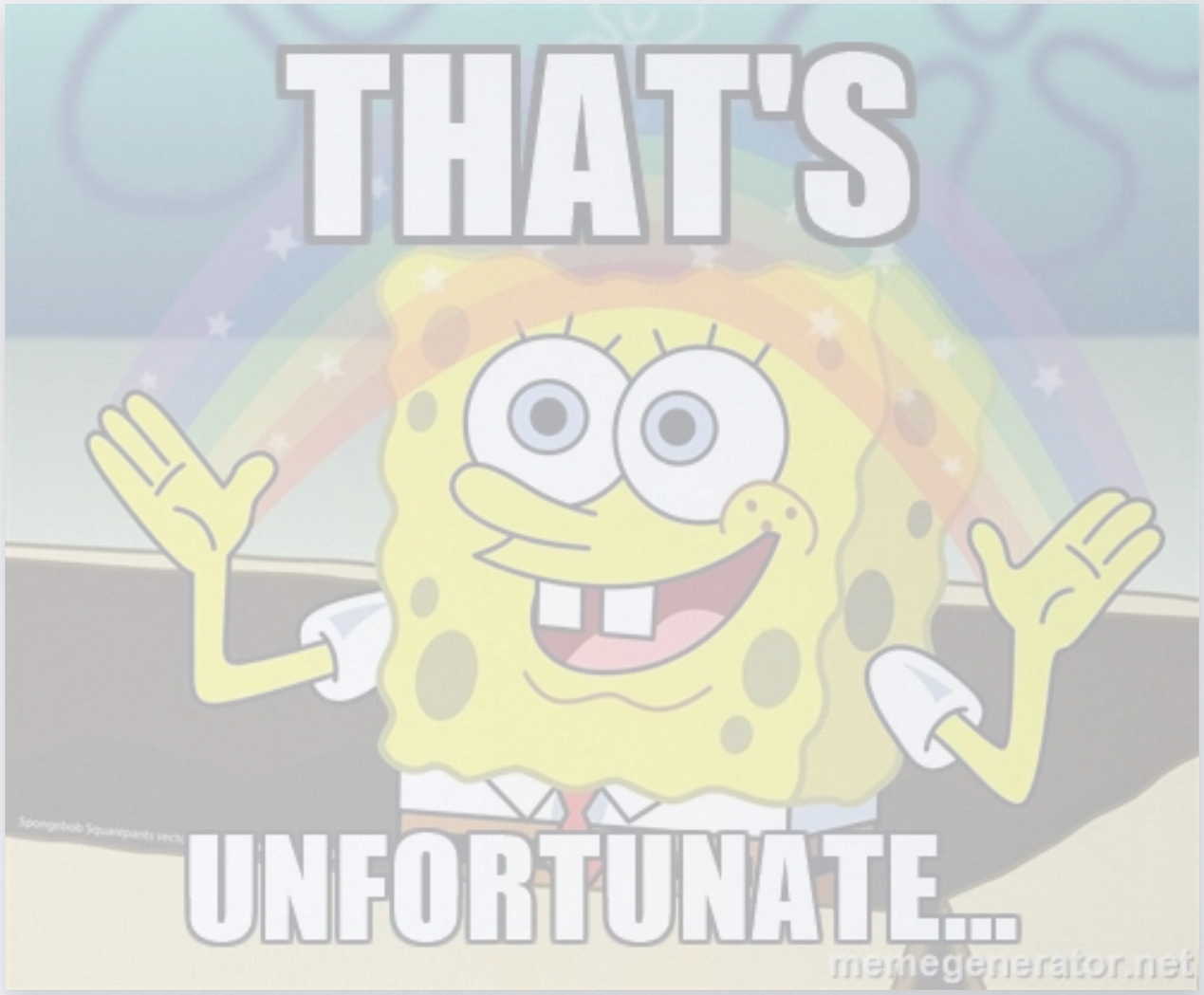


Database

It's in the ~~Computer~~ Database

DATABASE ACTIONS (CRUD)

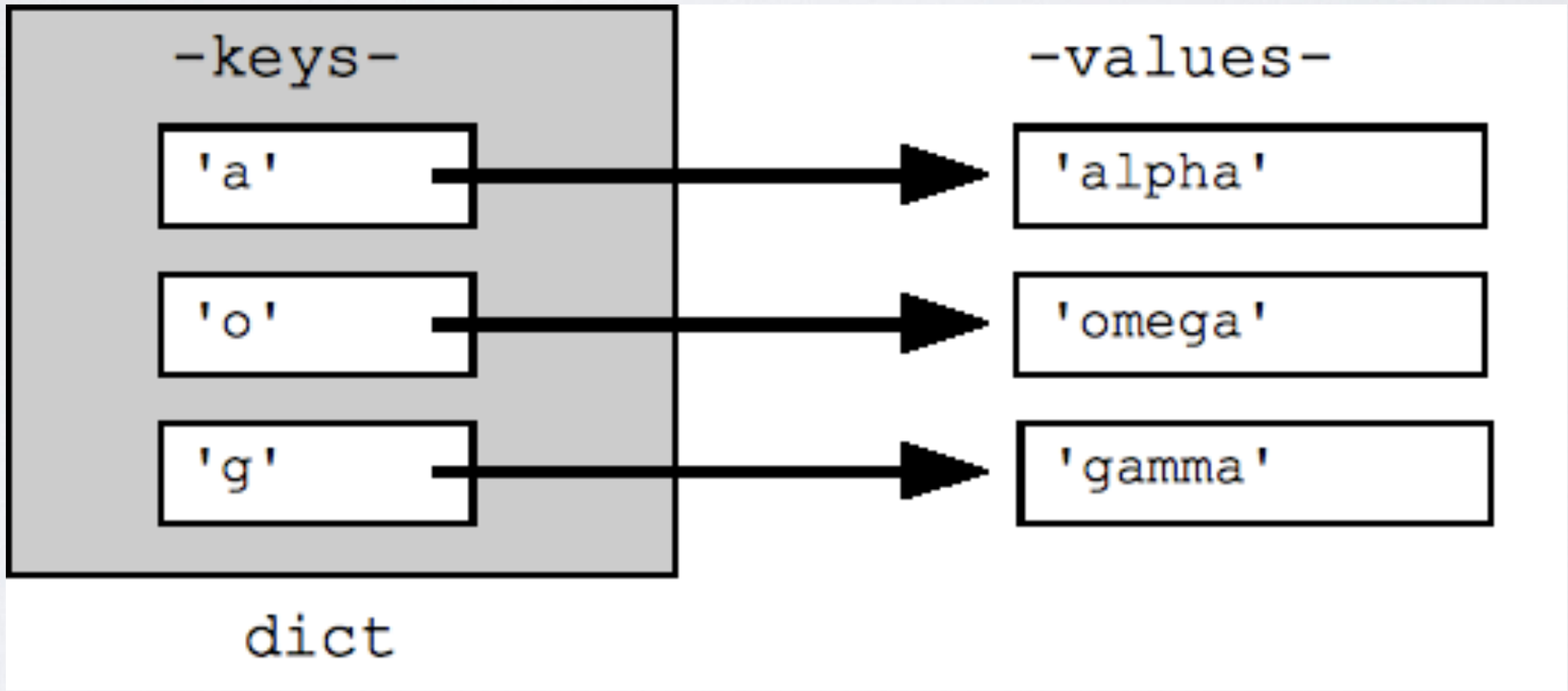
- CREATE
- READ
- UPDATE
- DELETE



TYPES OF DATABASES

- "SQL" a.k.a Relational Database
- NoSQL (Not only SQL - No SQL!)
  - key-value stores
  - document stores

KEY-VALUE STORE



KEY-VALUE STORE

54 =


```
{
  "id": 54,
  "company": "Arts Management Systems Ltd.",
  "date_entered": "2004-08-14T07:53:34-04:00",
  "date_updated": "2010-01-18T09:41:17-05:00",
  "deceased": false,
  "first_name": "David",
  "height": 193.00,
  "initial": ‘‘M J",
  "last_name": "McKeone",
  "nick_name": "Dave",
  "title": "Presenter Extraordinaire",
  "weight": 95.00,
  "gender": "Male",
  "salutation": "Mr."
}
```

KEY-VALUE STORE

Great for caching

https://dogpilecache.readthedocs.io/en/latest/

http://docs.sqlalchemy.org/en/latest/_modules/examples/dogpile_caching/caching_query.html

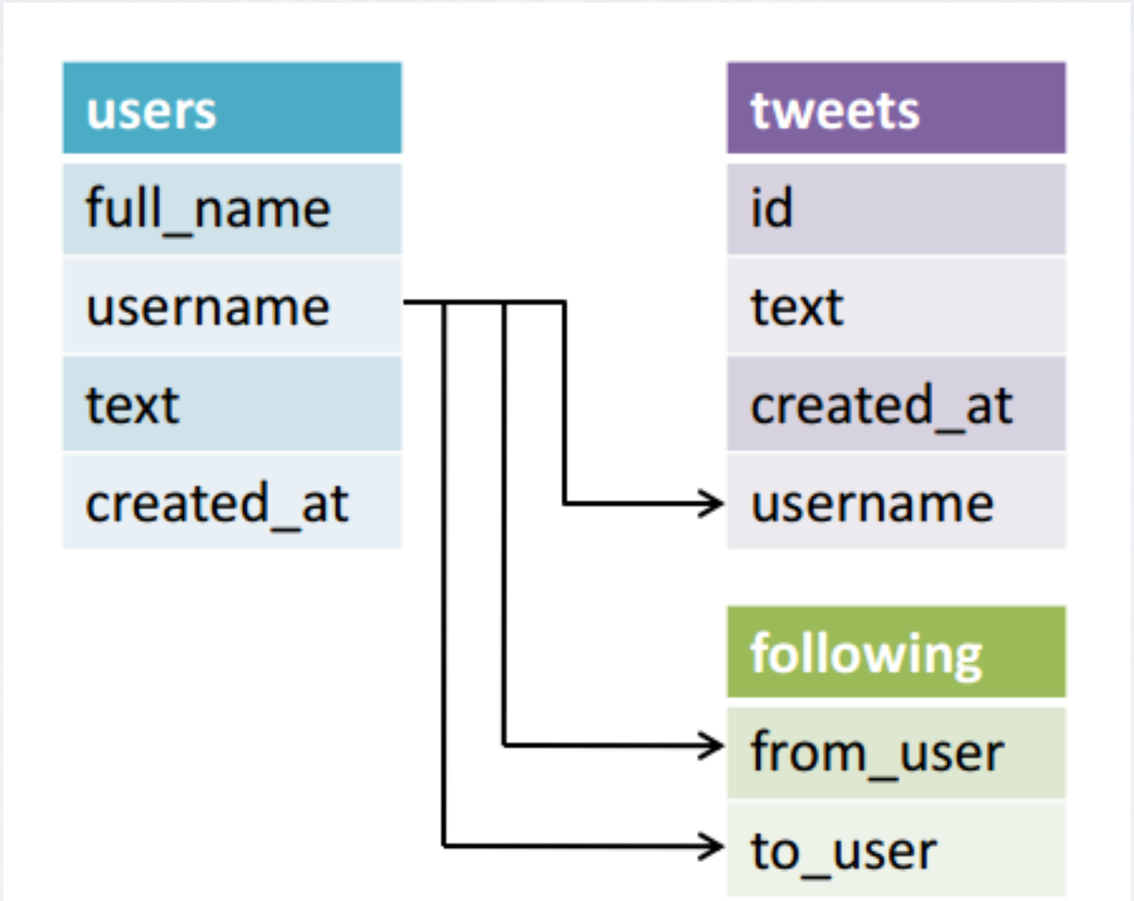
RELATIONAL DATABASES

Table

	A	B	C	D
1				
2	**id**	**first_name**	**last_name**	
3	1	David	McKeone	
4	2	Kurt	Neufeld	
5	3	Mike	Warren	
6				
7				

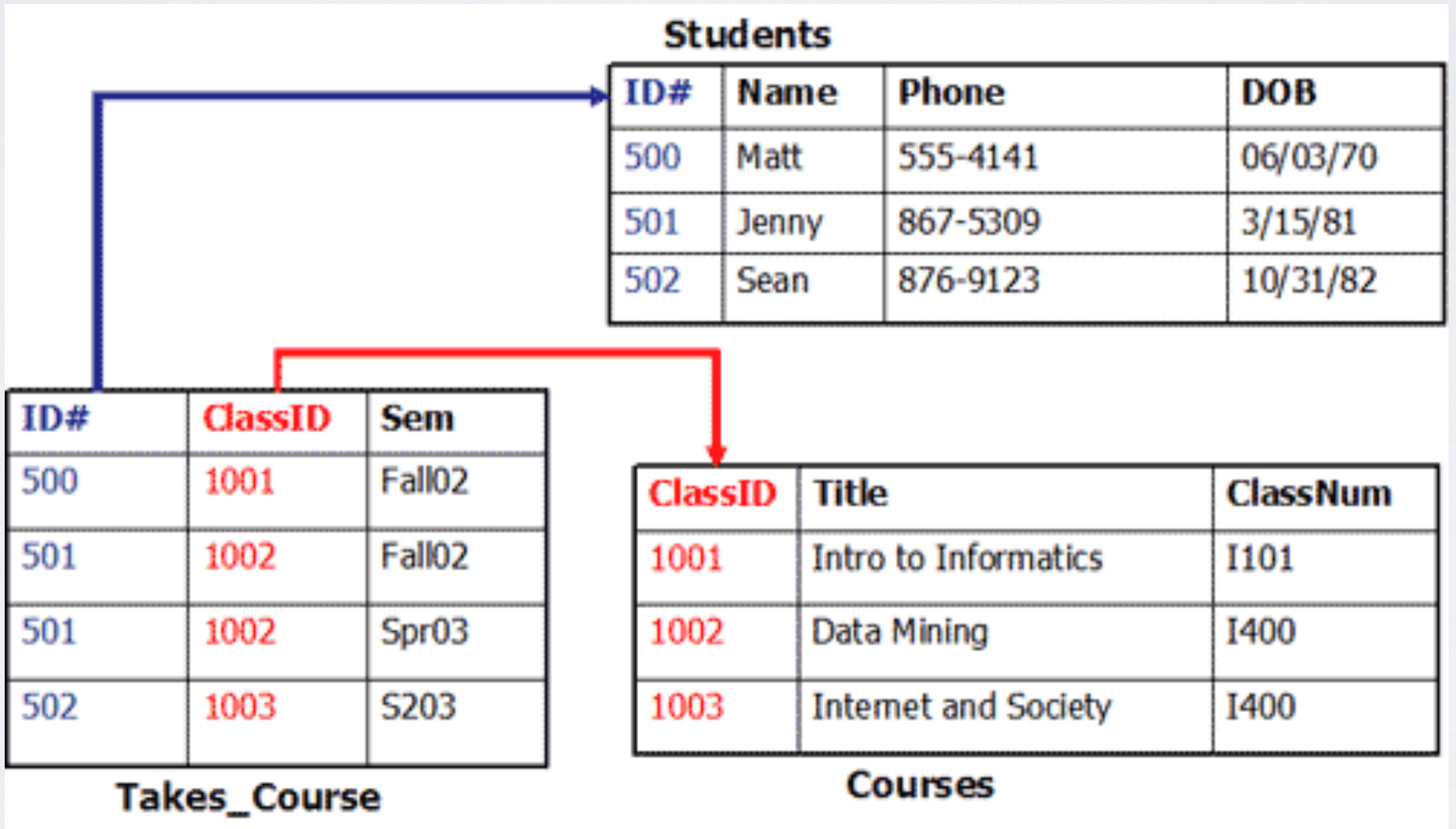
RELATIONAL DATABASES

Relationships



RELATIONAL DATABASES

Relationships - Primary/Foreign Keys



Students

ID#	Name	Phone	DOB
500	Matt	555-4141	06/03/70
501	Jenny	867-5309	3/15/81
502	Sean	876-9123	10/31/82

Takes_Course

ID#	ClassID	Sem
500	1001	Fall02
501	1002	Fall02
501	1002	Spr03
502	1003	S203

Courses

ClassID	Title	ClassNum
1001	Intro to Informatics	I101
1002	Data Mining	I400
1003	Internet and Society	I400

DATABASE ACTIONS (CRUD)

- CREATE > INSERT
- READ > SELECT
- UPDATE > UPDATE
- DELETE > DELETE

SQL READ

```
SELECT *
FROM users
WHERE first_name = 'Kurt'
```

CREATE

```
INSERT INTO users (first_name, last_name)
VALUES (‘David’, ‘McKeone’)
```

UPDATE

UPDATE users
SET data = {‘awesome’: true}
WHERE first_name = ‘Mike’

DELETE

```
DELETE FROM users
WHERE first_name = 'Mike'
```

DOCUMENT DATABASE

- Table ~= Document
- Documents aren't rows/columns (they are trees)
- Relationship enforcement is up to you (good luck!)

HOW DO YOU TRACK RELATIONSHIPS IN NOSQL?

All the answers for how to store many-to-many associations in the "NoSQL way" reduce to the same thing: storing data redundantly.

In NoSQL, you don't design your database based on the relationships between data entities. You design your database based on the queries you will run against it. Use the same criteria you would use to denormalize a relational database: if it's more important for data to have cohesion (think of values in a comma-separated list instead of a normalized table), then do it that way.

But this inevitably optimizes for one type of query (e.g. comments by any user for a given article) at the expense of other types of queries (comments for any article by a given user). If your application has the need for both types of queries to be equally optimized, you should not denormalize. And likewise, you should not use a NoSQL solution if you need to use the data in a relational way.

There is a risk with denormalization and redundancy that redundant sets of data will get out of sync with one another. This is called an anomaly. When you use a normalized relational database, the RDBMS can prevent anomalies. In a denormalized database or in NoSQL, it becomes your responsibility to write application code to prevent anomalies.

One might think that it'd be great for a NoSQL database to do the hard work of preventing anomalies for you. There is a paradigm that can do this -- the relational paradigm.

http://stackoverflow.com/a/4210561/589362

POSTGRESQL - RELATIONAL

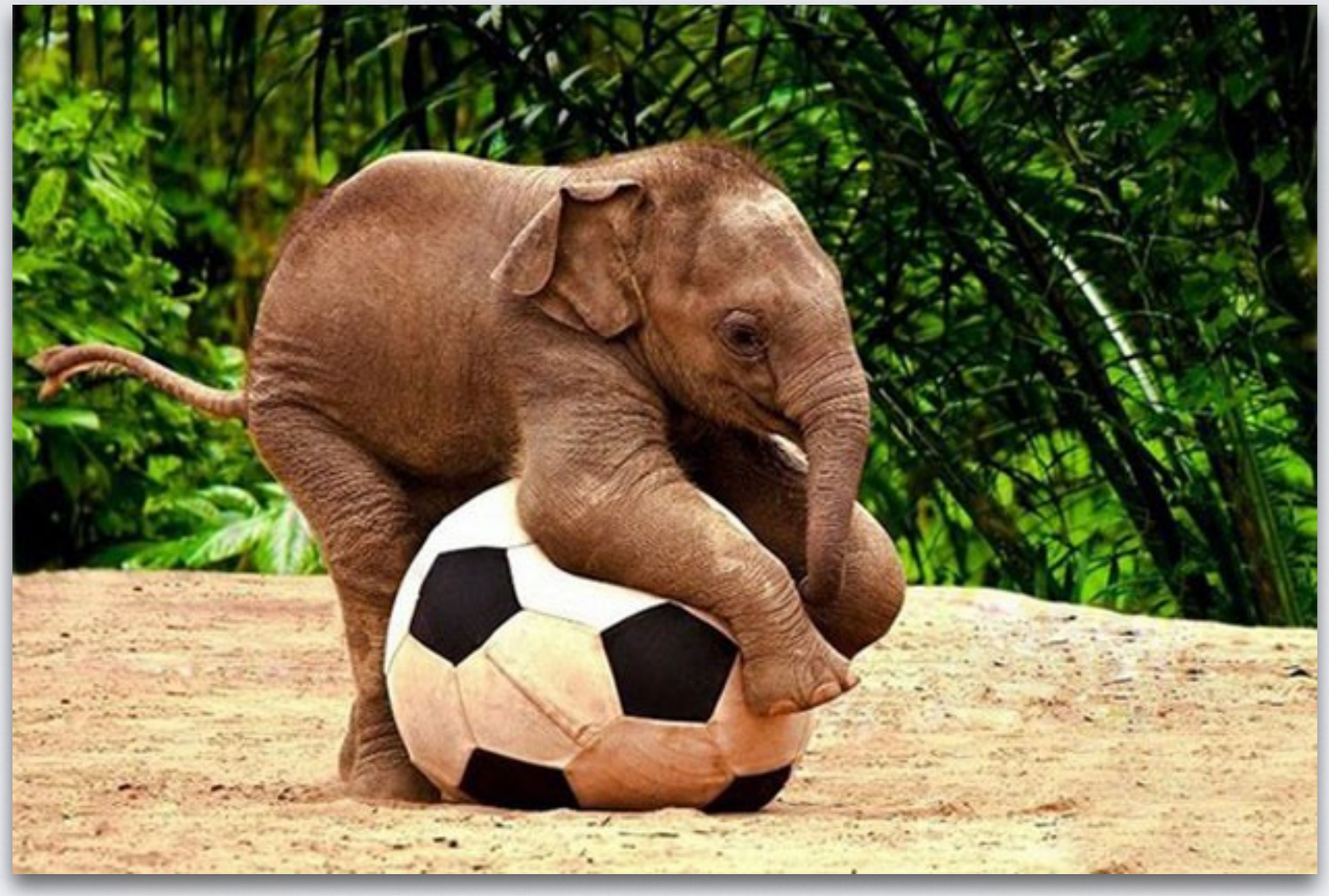
- Good for the vast majority of your data
- Will help you with data correctness when you don't know you need help (but ya, that can be frustrating)
- Fast enough that for most applications a single server (maybe with replication) will be good for a long, long time.

Distributed: https://www.citusdata.com/

SO, WHAT COULD POSSIBLY GO WRONG?

NOTHING! TURNS OUT, ELEPHANTS CAN MOVE FAST!!

It’ll be fine…



… just play with the elephant.