postgreSQL

postgreSQL in Flask

~~in 15 or so slides~~
37 slides

but first …

Principle of Least Privilege

A user (or process) should have the lowest level of privilege required in order to perform his/her assigned task.

Principle of Least Privilege

- not root
  - why? because as root we can do anything

MySQL easy:

```
CREATE DATABASE IF NOT EXISTS aliendb;
GRANT ALL PRIVILEGES ON aliendb.* to
'alienUsername'@'localhost' identified by 'alienPassword';
```

postgreSQL method

1. login as postgres and restrict access to important databases.
   `REVOKE connect ON DATABASE world FROM PUBLIC;`
2. create db associated w/ new user. `\i music.sql`
3. create user:
   ```
   create user music with password
   ‘2Vk0H39RLW6fA14GVd';
   ```
4. `grant select, insert on albums to music;`
   `(need to be in the music database – \c music`

   `grant all on sequence albums_id_seq to music;`

a test

```
mitsugo@cs350:~/workspace $ psql -U music -h localhost
Password for user music:
psql (9.3.5)
Type "help" for help.

music=> \c world
FATAL:  permission denied for database "world"
DETAIL:  User does not have CONNECT privilege.
Previous connection kept
```

a test

```
music=> \c music
You are now connected to database "music" as user
"music".
music=> select * from albums;
 id |     artist     |         name         | rank
----+----------------+----------------------+------
  1 | Jason Aldean   | Old Boots, New Dirt  |    1
  2 | Taylor Swift   | 1989                 |   37
  3 | Hozier         | Hozier               |    2
  4 | Flying Lotus   | You're Dead          |   19
  5 | Frozen         | Soundtrack           |   18
  6 | Ariana Grande  | My Everything        |   30
(6 rows)
```

```
create user music with password '2Vk0H39RLW6fA14GVd';
```

what happens if I change my mind about world?

```
postgres=# grant connect on database world to music;
GRANT
postgres=# \q
mitsugo@cs350:~/workspace $ psql -U music -h localhost
Password for user music:
```

what happens if I change my mind about world?

```
music=> \c world
You are now connected to database "world" as user "music".
world=> \d city
```

```
                                    Table "public.city"
   Column    |         Type          |                       Modifiers
-------------+-----------------------+-------------------------------------------------
 id          | integer               | not null default nextval('city_id_seq'::regclass)
 name        | character varying(35) | not null default ''::character varying
 countrycode | character(3)          | not null default ''::bpchar
 district    | character varying(20) | not null default ''::character varying
 population  | integer               | not null default 0
Indexes:
    "city_pkey" PRIMARY KEY, btree (id)
```

```
world=> select name from city where name = 'Albuquerque';
ERROR:  permission denied for relation city
world=>
```

what happens if I change my mind about world?

```
postgres=# \c world
You are now connected to database "world" as user "postgres".
world=# grant select on city to music;
GRANT
world=# \q
mitsugo@cs350:~/workspace $ psql -U music -h localhost
Password for user music:
psql (9.3.5)
Type "help" for help.

music=> \c world
You are now connected to database "world" as user "music".
world=> select name from city where name = 'Albuquerque';
    name
-------------
 Albuquerque
(1 row)
```

Root have Password?

- if not …

```
SET PASSWORD FOR ‘root’@‘localhost’ = PASSWORD(‘f3jga908’);
```

but first …

install the postgreSQL adapter

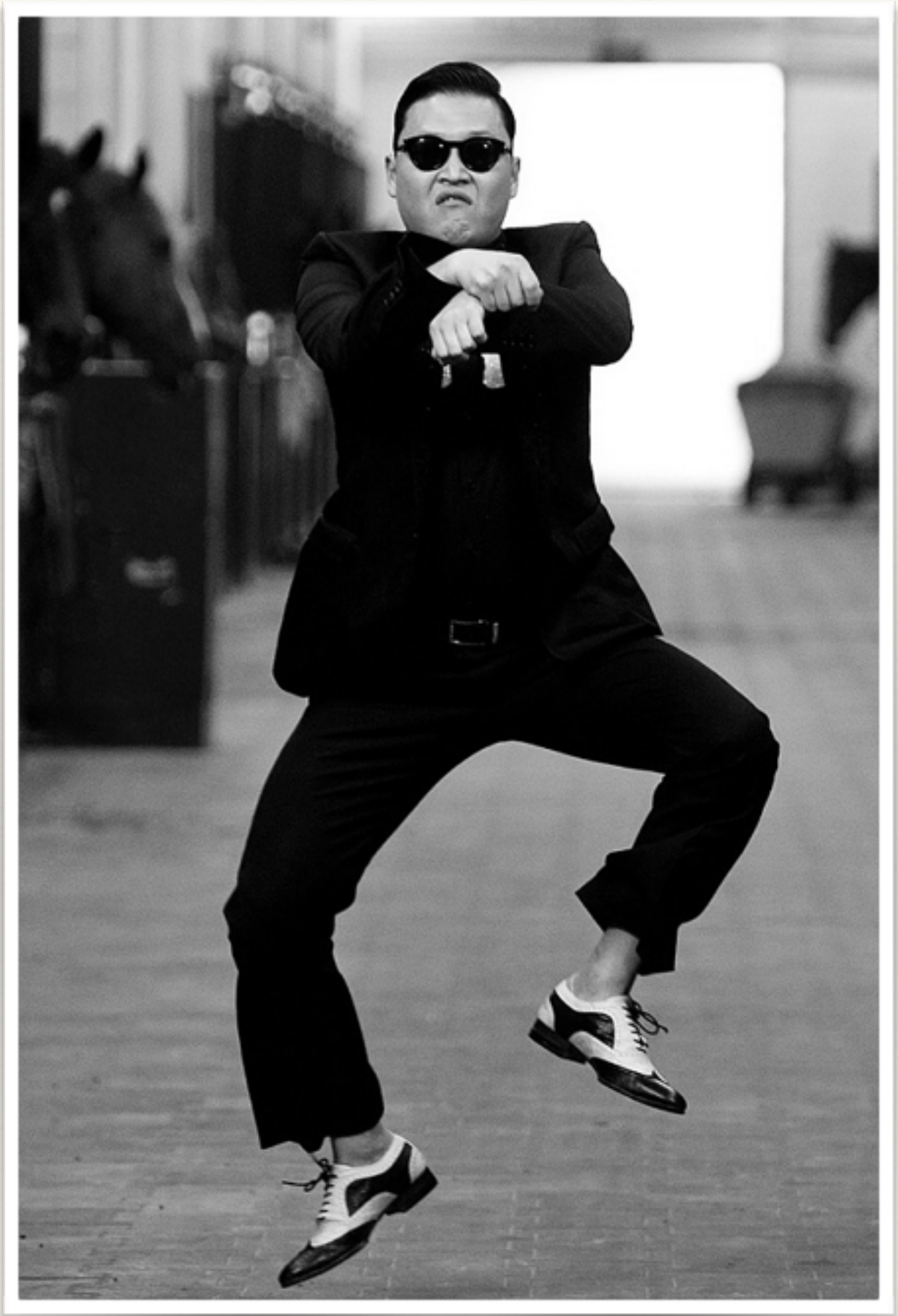
Psycopg is the most popular PostgreSQL adapter for the Python programming language. At its core it fully implements the Python DB API 2.0 specifications. Several extensions allow access to many of the features offered by PostgreSQL.

“Its main features are the complete implementation of the Python DB API 2.0 specification and the thread safety (several threads can share the same connection). It was designed for heavily multi-threaded applications that create and destroy lots of cursors and make a large number of concurrent INSERTs or UPDATEs..”

database adapter

psycopg2

```
sudo apt-get install python-psycopg2
```



Back to postgreSQL & Flask

Follow along in the music code in alienAbduction!!

6 Basic Steps

1. import the psycopg2 library modules
2. connect with psycopg2.connect
3. create a cursor object
4. assemble query string
5. execute the query with cur.execute
6. fetch the results

5 Basic Steps

1. import the MySQLdb library modules
2. connect with MySQLdb.connect
3. assemble query string
4. execute the query with cur.execute
5. fetch the results

1) import modules

```
import psycopg2
import psycopg2.extras
```

2) connect with psycopg2.connect

```
def connectToDB():
  connectionString = 'dbname=music user=musicUser
             password=1337music host=localhost'
  try:
    return psycopg2.connect(connectionString)
  except:
    print("Can't connect to database")
```

the abbreviated version

```
connection = psycopg2.connect('dbname=music …')
```

3) create a cursor object

```
conn = psycopg2.connect(…)
cur = conn.cursor()
```

In computer science, a database cursor is a control structure that enables traversal over the records in a database. Cursors facilitate subsequent processing in conjunction with the traversal, such as retrieval, addition and removal of database records. …
http://en.wikipedia.org/wiki/Database_cursor

3) create a cursor object

```
conn = psycopg2.connect(…)
cur = conn.cursor()
```

In computer science, a database cursor is a control structure that enables traversal over the records in a database. Cursors facilitate subsequent processing in conjunction with the traversal, such as retrieval, addition and removal of database records. …
http://en.wikipedia.org/wiki/Database_cursor

It is a structure we will use to interact with the database

4) assemble query string

```
query = "SELECT population FROM Country WHERE name = 'Haiti'"

query = "SELECT * FROM users WHERE name = '" +
         request.form['name'] + "'"

print query
```

Don’t do this

5) execute the query with cur.execute

```
cur.execute(query)

cur.execute("select artist, name from albums")
```

5) execute the query with cur.execute

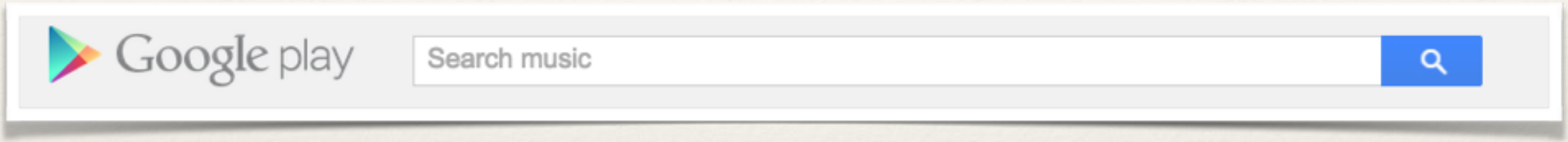
So why shouldn't we do this?

```
query = "SELECT * FROM users WHERE name = '" +
        request.form['name'] + "'"

print query
```

5) execute the query with cur.execute

Suppose we want an html form where the user will type in the name of an artist:

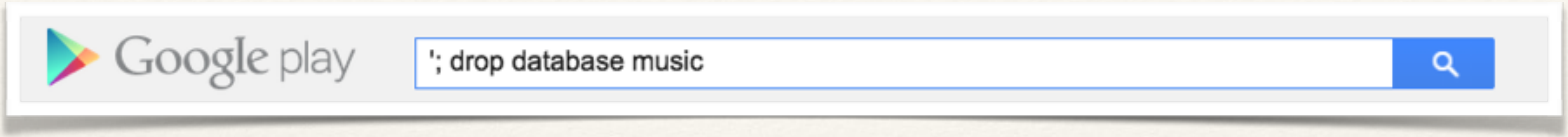


```
<input type="text" name="artist>"

query = "select artist, name from albums
     where artist = ‘" + request.form[‘artist']

cur.execute(query)
```

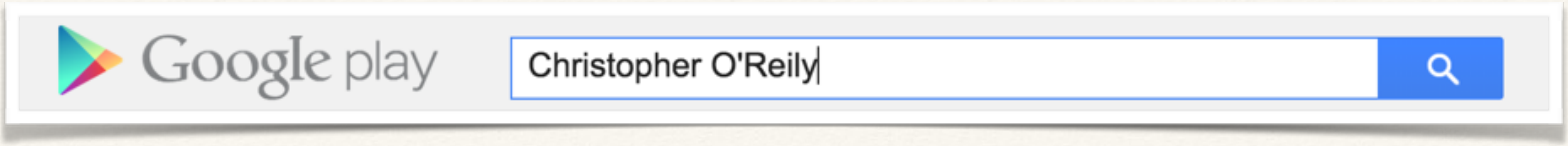
queries



'; drop database music

```
select artist, name from albums where artist = ‘’;
drop database music;
```

danger of SQL injection



Christopher O'Reily

```
query = "select * from music where artist = '" +
        request.form[‘artist’] + "'"

>>> query
"select * from music where artist = 'Christopher O'Reily'"

"select * from music where artist = 'Christopher O'Reily'"
```

```
cur.execute("""INSERT INTO albums (artist, name, rank)
       VALUES (%s, %s, %s);""",
       (request.form['artist'], request.form['album'],
request.form['rank']) )
    except:
      print("ERROR inserting into albums")
```

6) fetch the results

```
item = cur.fetchone()

or

results = cur.fetchall()


return render_template('music.html', albums=results)
```

```
{% for album in albums %}
<tr><td>{{album[0]}}</td><td>{{album[1]}}</td></tr>
{% endfor %}
```

6) fetch the results

fetch results as list of python dictionaries

```
cur = conn.cursor(cursor_factory=psycopg2.extras.DictCursor)
try:
  cur.execute("select artist, name from albums")
except:
  print("Error executing select")
results = cur.fetchall()
print results
return render_template('music2.html', albums=results)
```

```
{% for album in albums %}

<tr><td>{{album['artist']}}</td><td>{{album['name']
td></tr>

{% endfor %}
```

compare the results

```
{% for album in albums %}
<tr><td>{{album[0]}}</td><td>{{album[1]}}</td></tr>
{% endfor %}

{% for album in albums %}
<tr><td>{{album['artist']}}</td><td>{{album['name']}}<
td></tr>
{% endfor %}
```

```
music=# select artist, name from albums;
    artist     |         name
---------------+---------------------
 Jason Aldean  | Old Boots, New Dirt
 Taylor Swift  | 1989
 Hozier        | Hozier
 Flying Lotus  | You're Dead
 Frozen        | Soundtrack
 Ariana Grande | My Everything
(6 rows)
```

```
[['Jason Aldean', 'Old Boots, New Dirt'],
 ['Taylor Swift', '1989'],
 ['Hozier', 'Hozier'],
 ['Flying Lotus', "You're Dead"],
 ['Frozen', 'Soundtrack'],
 ['Ariana Grande', 'My Everything']]
```

```
music=# select artist, name from albums;
    artist     |         name
---------------+---------------------
 Jason Aldean  | Old Boots, New Dirt
 Taylor Swift  | 1989
 Hozier        | Hozier
 Flying Lotus  | You're Dead
 Frozen        | Soundtrack
 Ariana Grande | My Everything
(6 rows)
```

```
[['Jason Aldean', 'Old Boots, New Dirt'],
 ['Taylor Swift', '1989'],
 ['Hozier', 'Hozier'],
 ['Flying Lotus', "You're Dead"],
 ['Frozen', 'Soundtrack'],
 ['Ariana Grande', 'My Everything’]]
```

```
results[0][‘artist’] = 'Jason Aldean’
results[0][‘name’] = 'Jason Aldean’
```

code

- /music -> uses cursor()
- /music2 -> uses cursor(cursor_factory=psycopg2.extras.DictCursor)
- /music3 -> shows how to insert entries into db.

6 steps to posgres bliss

1. import the psycopg2 library modules
2. connect with psycopg2.connect
3. create a cursor object
4. assemble query string
5. execute the query with cur.execute
6. fetch the results

alienAbduction2

All this in the demo

/music
/music2
/music3

The Team Task

- review and understand the music examples
- posgreSQL-ify alien abductions
  - determine the structure of the db & construct one
  - create a postgreSQL user & restrict access
  - when a person reports an abduction, add it to db.
  - get "see list of abductions" tab working