CS586 - Grad Project - Submission 2
Instructor: Charles Winstead
Student: Scott Griffy

Contents

1 Intro **1**

2 Psql **1**

3 psycopg2 **2**

4 Github API **2**

5 SQL statements/schema **3**

6 ER Diagram **5**

1 Intro

I used python libraries: sqlalchemy and eralchemy (on a private server) to generate both the sql statements (which I'm submitting as the relational schema as well) and the graph.
I used python libraries: requests and psycopg2 on my linux account at school to grab the data from the github api and put it into the database.
I've been working closely with the github API to make sure that everything I've described is possible. I believe that scraping github will be much more difficult because of rate limits. This might become the bottleneck on project scale rather than database or code complexity.
The interface will be command line based, or a simple flask html app.

2 Psql

Here's some proof of what I'm doing (using the psql command line):

```
f18wdb15=> select full_name, owner, forks, stars, created_at, subscribers from
   githubsec.Repo;
     full_name      |  owner   | forks | stars | created_at | subscribers
--------------------+----------+-------+-------+------------+-------------
 openssl/openssl    | openssl  |  3997 |  8732 | 1358236800 |         838
 jedisct1/libsodium | jedisct1 |   945 |  6192 | 1358668800 |         374
```

```
(2 rows)
```

3 psycopg2

And here's the python that grabs that (this is running on the linux lab)
I've got a longer file, but it's messy and I wasn't sure if I was supposed to submit it. Let me know if you want to see the full program so far and I'll clean it up.

```
scog@ada:~$ python3 -ic 'import psycopg2;import os'
>>> pgpass = os.environ['PGPASSWORD']
>>> conn = psycopg2.connect("dbname=f18wdb15 user=f18wdb15 password="+pgpass+" host=
   dbclass.cs.pdx.edu")
>>> cur=conn.cursor()
>>> cur.execute('select full_name, owner, forks, stars, created_at, subscribers from
   githubsec.Repo;')
>>> print(cur.fetchone())
('openssl/openssl', 'openssl', 3997, 8732, 1358236800, 838)
>>> print(cur.fetchone())
('jedisct1/libsodium', 'jedisct1', 945, 6192, 1358668800, 374)
>>>
```

4 Github API

And here's the python that gets the data from github
Again, there's a longer program, but I thought this would be enough for a proof of concept:

```
scog@ada:~$ python3 -ic 'import requests;import os'
>>> auth = (os.environ['GITHUB_USER'], os.environ['GITHUB_AUTH'])
>>> res=requests.get("https://api.github.com/search/repositories?q=crypto+language:c&
   sort=stars&order=desc", auth=auth)
>>> for item in res.json()['items']:
...   print(item['full_name'])
...
openssl/openssl
jedisct1/libsodium
awslabs/s2n
ARMmbed/mbedtls
firmianay/CTF-All-In-One
B-Con/crypto-algorithms
jedisct1/libsodium-php
trezor/trezor-crypto
jedisct1/minisign
jedisct1/libhydrogen
jedisct1/swift-sodium
mwarning/KadNode
coinfoundry/miningcore
DavyLandman/AESLib
LoupVaillant/Monocypher
WickrInc/wickr-crypto-c
neurodroid/cryptonite
```

```
zhaozg/lua-openssl
haskell-crypto/cryptonite
angt/glorytun
eduardsui/tlse
aczid/crypto1_bs
gost-engine/engine
blynn/pbc
cantora/avr-crypto-lib
cryptodev-linux/cryptodev-linux
gdbinit/gopher
globalzon/yaamp
MicrochipTech/cryptoauth-openssl-engine
MicrochipTech/cryptoauthlib
>>>
```

5 SQL statements/schema

Here is the SQL statements I'll use to create the database:

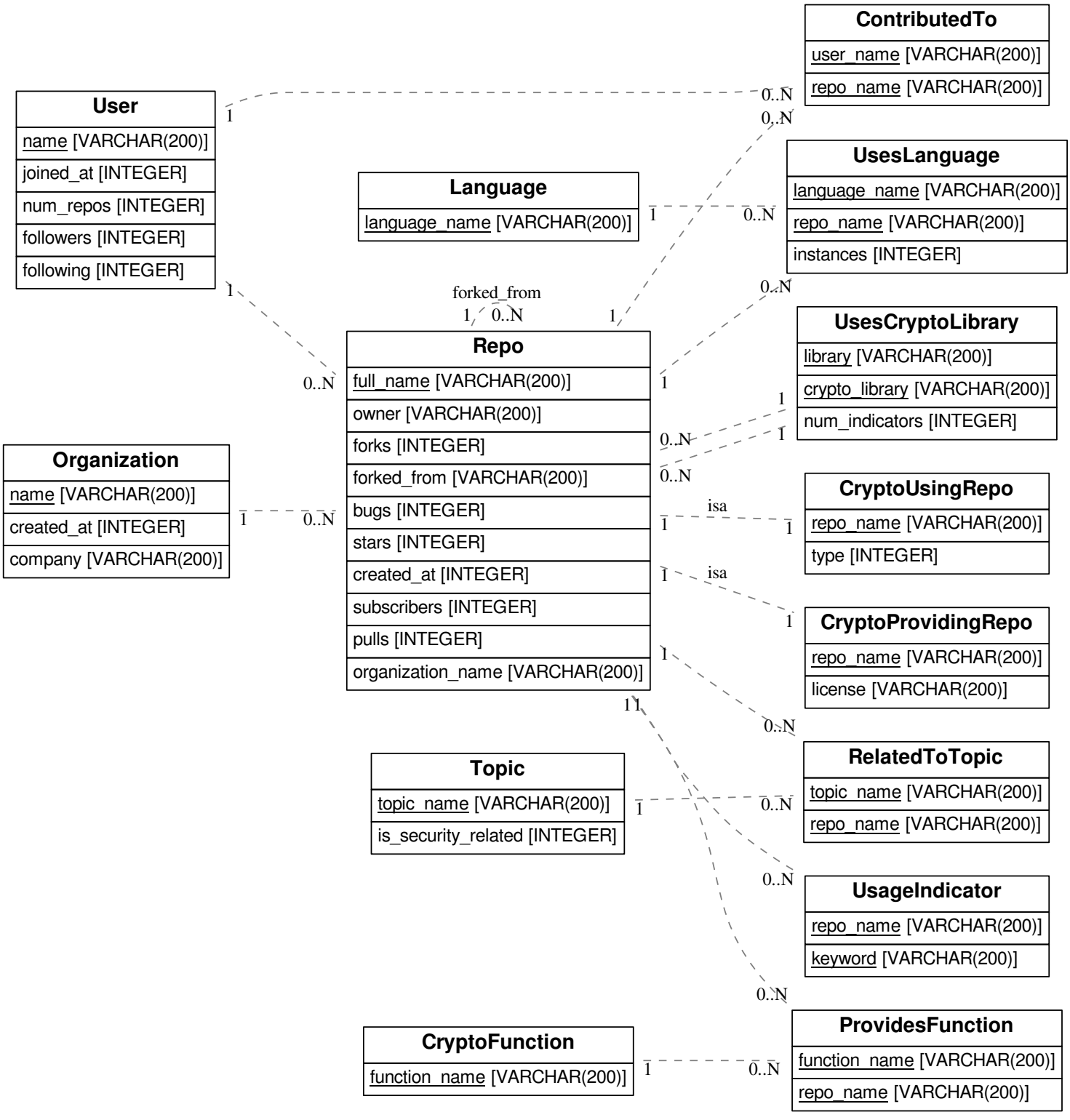
```
CREATE TABLE "CryptoUsingRepo" (
        repo_name VARCHAR(200) NOT NULL,
        type INTEGER NOT NULL,
        PRIMARY KEY (repo_name),
        FOREIGN KEY(repo_name) REFERENCES "Repo" (full_name)
)

;

CREATE TABLE "CryptoProvidingRepo" (
        repo_name VARCHAR(200) NOT NULL,
        license VARCHAR(200),
        PRIMARY KEY (repo_name),
        FOREIGN KEY(repo_name) REFERENCES "Repo" (full_name)
)

;

CREATE TABLE "Repo" (
        full_name VARCHAR(200) NOT NULL,
        owner VARCHAR(200),
        forks INTEGER,
        forked_from VARCHAR(200),
        bugs INTEGER,
        stars INTEGER,
        created_at INTEGER,
        subscribers INTEGER,
        pulls INTEGER,
        organization_name VARCHAR(200),
        PRIMARY KEY (full_name),
        FOREIGN KEY(owner) REFERENCES "User" (name),
        FOREIGN KEY(forked_from) REFERENCES "Repo" (full_name),
        FOREIGN KEY(organization_name) REFERENCES "Organization" (name)
)

;
```

```
CREATE TABLE "ContributedTo" (
        user_name VARCHAR(200) NOT NULL,
        repo_name VARCHAR(200) NOT NULL,
        PRIMARY KEY (user_name, repo_name),
        FOREIGN KEY(user_name) REFERENCES "User" (name),
        FOREIGN KEY(repo_name) REFERENCES "Repo" (full_name)
)

;

CREATE TABLE "UsesLanguage" (
        language_name VARCHAR(200) NOT NULL,
        repo_name VARCHAR(200) NOT NULL,
        instances INTEGER,
        PRIMARY KEY (language_name, repo_name),
        FOREIGN KEY(language_name) REFERENCES "Language" (language_name),
        FOREIGN KEY(repo_name) REFERENCES "Repo" (full_name)
)

;

CREATE TABLE "ProvidesFunction" (
        function_name VARCHAR(200) NOT NULL,
        repo_name VARCHAR(200) NOT NULL,
        PRIMARY KEY (function_name, repo_name),
        FOREIGN KEY(function_name) REFERENCES "CryptoFunction" (function_name),
        FOREIGN KEY(repo_name) REFERENCES "Repo" (full_name)
)

;

CREATE TABLE "Language" (
        language_name VARCHAR(200) NOT NULL,
        PRIMARY KEY (language_name)
)

;

CREATE TABLE "CryptoFunction" (
        function_name VARCHAR(200) NOT NULL,
        PRIMARY KEY (function_name)
)

;

CREATE TABLE "RelatedToTopic" (
        topic_name VARCHAR(200) NOT NULL,
        repo_name VARCHAR(200) NOT NULL,
        PRIMARY KEY (topic_name, repo_name),
        FOREIGN KEY(topic_name) REFERENCES "Topic" (topic_name),
        FOREIGN KEY(repo_name) REFERENCES "Repo" (full_name)
)

;

CREATE TABLE "Topic" (
        topic_name VARCHAR(200) NOT NULL,
        is_security_related INTEGER,
        PRIMARY KEY (topic_name)
)
```

```
;

CREATE TABLE "User" (
        name VARCHAR(200) NOT NULL,
        joined_at INTEGER,
        num_repos INTEGER,
        followers INTEGER,
        following INTEGER,
        PRIMARY KEY (name)
)

;

CREATE TABLE "Organization" (
        name VARCHAR(200) NOT NULL,
        created_at INTEGER,
        company VARCHAR(200),
        PRIMARY KEY (name)
)

;

CREATE TABLE "UsageIndicator" (
        repo_name VARCHAR(200) NOT NULL,
        keyword VARCHAR(200) NOT NULL,
        PRIMARY KEY (repo_name, keyword),
        FOREIGN KEY(repo_name) REFERENCES "Repo" (full_name)
)

;

CREATE TABLE "UsesCryptoLibrary" (
        library VARCHAR(200) NOT NULL,
        crypto_library VARCHAR(200) NOT NULL,
        num_indicators INTEGER NOT NULL,
        PRIMARY KEY (library, crypto_library),
        FOREIGN KEY(library) REFERENCES "Repo" (full_name),
        FOREIGN KEY(crypto_library) REFERENCES "Repo" (full_name)
)

;
```

6 ER Diagram

On the next page is the ER diagram of these statements:


User
name [VARCHAR(200)]
joined_at [INTEGER]
num_repos [INTEGER]
followers [INTEGER]
following [INTEGER]
Organization
name [VARCHAR(200)]
created_at [INTEGER]
company [VARCHAR(200)]
Repo
full_name [VARCHAR(200)]
owner [VARCHAR(200)]
forks [INTEGER]
forked_from [VARCHAR(200)]
bugs [INTEGER]
stars [INTEGER]
created_at [INTEGER]
subscribers [INTEGER]
pulls [INTEGER]
organization_name [VARCHAR(200)]
forked_from
Language
language_name [VARCHAR(200)]
Topic
topic_name [VARCHAR(200)]
is_security_related [INTEGER]
CryptoFunction
function_name [VARCHAR(200)]
ContributedTo
user_name [VARCHAR(200)]
repo_name [VARCHAR(200)]
UsesLanguage
language_name [VARCHAR(200)]
repo_name [VARCHAR(200)]
instances [INTEGER]
UsesCryptoLibrary
library [VARCHAR(200)]
crypto_library [VARCHAR(200)]
num_indicators [INTEGER]
CryptoUsingRepo
repo_name [VARCHAR(200)]
type [INTEGER]
isa
CryptoProvidingRepo
repo_name [VARCHAR(200)]
license [VARCHAR(200)]
isa
RelatedToTopic
topic_name [VARCHAR(200)]
repo_name [VARCHAR(200)]
UsageIndicator
repo_name [VARCHAR(200)]
keyword [VARCHAR(200)]
ProvidesFunction
function_name [VARCHAR(200)]
repo_name [VARCHAR(200)]
1
0..N