Data Storage and Retrieval – Assignment 3

Aaron Niskin

September 12, 2016

1a)

Create a database called hubway_niskin from the data in compsci01:/usr/share/databases/Hubway.

```
createdb hubway_niskin
```

1b)

Create two appropriately named tables and load the Hubway csv files into the tables. Include any reasonable integrity constraints in your "create table" commands. Please at least include a foreign key constraint. Read the README file!

First we're going to want to get rid of the apostrophe prefixes. Since the only way I can think of to find non-zip-code-preceeding apostrophies is to see which ones aren't preceeding a five digit number and sandwiched between two commas, let's just replace those and see if we have any apostrophies left.

```
cat hubway_trips.csv | sed -r "s/,'([0-9]{5}),/,\1,/g" > hubway_trips_no-apos.csv
```

Great! Now that our csv files seem to be in order, let's create our tables! Since our `trips` table will eventually refer to the station IDs, it will make our lives easier to create the stations table first, then use the station_id as a foreign key.

```
CREATE TABLE stations(
station_id INT PRIMARY KEY,
terminal CHARACTER(6) CHECK (terminal <> ''),
station CHARACTER VARYING(100),
municipal CHARACTER VARYING(20),
lat DOUBLE PRECISION,
lng DOUBLE PRECISION,
status CHARACTER VARYING(8));
```

Then we copy the data in:

```
\copy stations(station_id, terminal, station, municipal, lat, lng, status)
  FROM 'hubway_stations.csv' DELIMITER ',' CSV HEADER
```

Now we can make our trips table:

```
CREATE TABLE trips (seq_id SERIAL primary key,
hubway_id BIGINT,
status CHARACTER VARYING(10),
duration INTEGER,
start_date TIMESTAMP WITHOUT TIME ZONE,
start_statn INTEGER REFERENCES stations(station_id),
end_date TIMESTAMP WITHOUT TIME ZONE,
end_statn  INTEGER REFERENCES stations(station_id),
bike_nr CHARACTER VARYING(20),
subsc_type CHARACTER VARYING(20),
zip_code  CHARACTER VARYING(6),
birth_date  INTEGER,
gender CHARACTER VARYING(10));
```

NOTE: The start station column is NOT named strt__statn, but rather start__statn.

And copy the data:

```
\copy trips(seq_id, hubway_id, status, duration,
            start_date, start_statn, end_date, end_statn,
            bike_nr, subsc_type, zip_code, birth_date, gender)
    FROM 'hubway_trips_no-apos.csv' DELIMITER ',' CSV HEADER
```

GREAT! Now we can move on to question 2!

2a)

Find the first 10 station names whose status correspond to "Removed", sorted by station, ascending.

```
SELECT station FROM stations
  WHERE lower(status) LIKE 'removed'
  ORDER BY station
  LIMIT 10;
```

The output was:

Andrew Station - Dorchester Ave at Humboldt Pl

Boston Medical Center - 721 Mass. Ave.

Boylston at Fairfield

Brookline Town Hall / Library Washington St

Charles Circle - Charles St. at Cambridge St.

Dudley Square

Harvard University River Houses at DeWolfe St / Cowperthwaite St

Mayor Thomas M. Menino - Government Center

New Balance - 38 Guest St.

Overland St at Brookline Ave

2b)

Find the first 10 station names that are located inside the bounding box formed by two given (latitude, longditude) points, sorted by station ascending.

For this, the two points I'm chosing are (42.3, -71.1) and (42.4, -71).

```sql
SELECT station FROM stations
  WHERE lat BETWEEN 42.3 AND 42.4
    AND lng BETWEEN -71.1 AND -71
  ORDER BY station_id
  LIMIT 10;
```

The output was:

Tremont St. at Berkeley St.

Northeastern U / North Parking Lot

Cambridge St. at Joy St.

Fan Pier

Ruggles Station / Columbus Ave.

Boston Medical Center - 721 Mass. Ave.

Back Bay / South End Station

Aquarium Station - 200 Atlantic Ave.

Prudential Center / Belvidere

South Station - 700 Atlantic Ave.

2c)

Find the first 10 trip IDs (hubway_id) that started or ended at stations within a bounding box formed by two given (latitude, longditude) points, sorted by id, ascending.

First, to make our lives easier (and computation faster), let's create a table of station IDs for stations within the box determined by the same (longditude, latitude) points from 2b:

```sql
CREATE TABLE q2c AS
  SELECT station_id FROM stations
    WHERE lat BETWEEN 42.3 AND 42.4
      AND lng BETWEEN -71.1 AND -71;
```

Then, since PostGreSQL doesn't like using one column tables, we have to select the station_id still from within the select:

```sql
SELECT hubway_id FROM trips
  WHERE start_statn IN (SELECT station_id FROM q2c)
     OR end_statn IN (SELECT station_id FROM q2c)
  ORDER BY hubway_id
  LIMIT 10;
```

And the results are:

```
     8
     9
    10
    11
    12
```

```
13
14
15
16
17
```

3)

So, as it turns out, every occurence of "Registered" users, are either followed by a digit, or by null. this can be found by executing

```
egrep "Registered,[^0-9,]" hubway_trips_no-apos.csv | less
```

Something a bit more interesting was that some casual users have zip codes:

```
egrep "Casual,[^,]" hubway_trips.csv | less
```

We can check even further:

```
egrep ",'[0-9]*[^0-9,]+" hubway_trips.csv | less
```

Which returns with nothing.

And these zip codes are seemingly indestinguishable from the registered users'. Another interesting fact is that apparently New England has quite a few zip codes starting with 0. Since this compnay is based out of Boston, we can't discount those zip codes. It seems as though all garbage values are empty values (as far as the CSV goes).

As far as suspect values, it seems that the only suspect values would be those Casual users with zip codes. How did they get them? (It's probably from the credit card information, but that's the closest thing to suspect I can find, so let's go with that).

```
CREATE TABLE q3zip AS
  SELECT
    CASE
      WHEN
        lower(subsc_type) LIKE 'registered' AND
        zip_code IS NULL
        THEN 'Registered user missing zip-code'
      WHEN
        lower(subsc_type) LIKE 'casual' AND
        zip_code IS NOT NULL
        THEN 'Casual user has zip code'
    END problem,
    seq_id,
    hubway_id,
    subsc_type,
    zip_code
  FROM trips
    WHERE (lower(subsc_type) LIKE 'casual'
              AND zip_code IS NOT NULL)
      OR (lower(subsc_type) LIKE 'registered'
              AND zip_code IS NULL);
```

As far as duration is concerned, they were apparently using the wrong data type and now they have negative numbers. So we have to identify those records. And some of the records' durations do not match up with their durations when calculated via end_time - start_time:

```
CREATE TABLE q3duration AS
  SELECT
    CASE WHEN duration < 0
              THEN 'The duration is negative'
         WHEN duration != EXTRACT (EPOCH FROM end_date) - EXTRACT(EPOCH FROM start_date)
              THEN 'The duration is not equal to the end_date - start_date'
      END problem,
      seq_id,
      duration,
      start_date,
      end_date
    FROM trips
    WHERE duration < 0
    OR duration != EXTRACT(EPOCH FROM end_date) - EXTRACT(EPOCH FROM start_date);
```

And now for our results:

```
hubway_niskin=> SELECT * FROM q3duration WHERE problem LIKE '%not equal%' LIMIT 5;
                        problem                         | seq_id  | duration |     start_date      |
--------------------------------------------------------+---------+----------+---------------------+---
 The duration is not equal to the end_date - start_date | 1515121 |     4620 | 2013-11-03 01:19:00 | 20
 The duration is not equal to the end_date - start_date | 1515122 |     4560 | 2013-11-03 01:20:00 | 20
 The duration is not equal to the end_date - start_date | 1515126 |     4440 | 2013-11-03 01:22:00 | 20
 The duration is not equal to the end_date - start_date | 1515127 |     4620 | 2013-11-03 01:23:00 | 20
 The duration is not equal to the end_date - start_date | 1515128 |     4380 | 2013-11-03 01:23:00 | 20

hubway_niskin=> SELECT * FROM q3duration WHERE problem LIKE '%is negative%' LIMIT 5;
         problem          | seq_id  | duration |     start_date      |      end_date
--------------------------+---------+----------+---------------------+---------------------
 The duration is negative | 1515158 |    -5700 | 2013-11-03 01:42:00 | 2013-11-03 01:07:00
 The duration is negative | 1515161 |    -5760 | 2013-11-03 01:44:00 | 2013-11-03 01:08:00
 The duration is negative | 1515162 |    -5820 | 2013-11-03 01:44:00 | 2013-11-03 01:07:00
 The duration is negative | 1515163 |    -5820 | 2013-11-03 01:44:00 | 2013-11-03 01:07:00
 The duration is negative | 1515166 |    -6240 | 2013-11-03 01:47:00 | 2013-11-03 01:03:00

 hubway_niskin=> SELECT * FROM q3zip WHERE problem LIKE '%has%' LIMIT 5;
         problem          | seq_id  | hubway_id | subsc_type | zip_code
--------------------------+---------+-----------+------------+----------
 Casual user has zip code | 1369995 |   1526340 | Casual     | 02360
 Casual user has zip code | 1370015 |   1526361 | Casual     | 02128
 Casual user has zip code | 1370068 |   1526418 | Casual     | 01810
 Casual user has zip code | 1370099 |   1526450 | Casual     | 02116
 Casual user has zip code | 1370105 |   1526456 | Casual     | 02118

hubway_niskin=> SELECT * FROM q3zip WHERE problem LIKE '%missing%' LIMIT 5;
             problem              | seq_id  | hubway_id | subsc_type | zip_code
----------------------------------+---------+-----------+------------+----------
 Registered user missing zip-code | 1369994 |   1526339 | Registered |
 Registered user missing zip-code | 1370014 |   1526360 | Registered |
 Registered user missing zip-code | 1370067 |   1526417 | Registered |
 Registered user missing zip-code | 1370098 |   1526449 | Registered |
 Registered user missing zip-code | 1370106 |   1526457 | Registered |
```

4.0

Create a database called `fanfiction_niskin`.

```
CREATE DATABASE fanfiction_niskin;
\connect fanfiction_niskin
```

4.1

Create a stories_orig table with url as the primary key.

NOTES FOR CLEANING DATA: Replace ",True," with ",TRUE,"

4a)

Use \copy to load Fanfiction/stories_orig.csv into the table. Despite what Postgres may say, the problem isn't that url is the primary key; the problem is in the data. Explain.

```
CREATE TABLE
  stories_orig(rating CHARACTER VARYING(3),
          updated date,
          favorites INTEGER,
          starring_chars CHARACTER VARYING(100),
          chapters INTEGER,
          complete BOOLEAN,
          collected_info TEXT,
          genre CHARACTER VARYING(50),
          description TEXT,
          language CHARACTER VARYING(20),
          author CHARACTER VARYING(50),
          url CHARACTER VARYING(200) PRIMARY KEY,
          follows INTEGER,
          title CHARACTER VARYING(150),
          reviews INTEGER,
          published DATE,
          words BIGINT);

\copy stories_orig(rating, updated, favorites, starring_chars, chapters, complete, collected_info, genre
  FROM 'Fanfiction/stories_orig.csv' DELIMITER ',' CSV HEADER

ERROR:  duplicate key value violates unique constraint "stories_orig_pkey"
DETAIL:  Key (url)=(http://www.fanfiction.net/s/9096319/1/Green-Eyed-Monster) already exists.
CONTEXT:  COPY stories_orig, line 277
```

Well, much as the error code suggested, line 275 has the same URL as line 277. Upon closer inspection we find that the entire entry is repeated. So this suggests that the URL should still be unique, but only if we eliminate duplicate entries.

4b)

So, to do this, we're going to want to alter the table a couple of times:

```
ALTER TABLE stories_orig DROP CONSTRAINT stories_orig_pkey;

ALTER TABLE stories_orig ADD col_id SERIAL PRIMARY KEY;
```

Then I realized that I didn't make the url column long enough, so:

```
ALTER TABLE stories_orig ALTER COLUMN url TYPE TEXT;
```

And we do the same for the author column (because people) and we import again and everything is fine.

4c)

Fix the data problem using a select statement that joins stories_orig with itself (!), checking for records with the same url. Try your sql with explain with and without an index, but run it with an index.

```
EXPLAIN SELECT * FROM stories_orig s
  WHERE col_id = (SELECT MIN(col_id) FROM stories_orig b
                         WHERE s.url = b.url);
                                  QUERY PLAN
--------------------------------------------------------------------------------
 Seq Scan on stories_orig s  (cost=0.00..44374344946.44 rows=3022 width=484)
   Filter: (col_id = (SubPlan 1))
   SubPlan 1
     ->  Aggregate  (cost=73424.42..73424.43 rows=1 width=4)
           ->  Seq Scan on stories_orig b  (cost=0.00..73424.41 rows=1 width=4)
                 Filter: (s.url = url)

CREATE INDEX story_url_index ON stories_orig (url);

EXPLAIN SELECT * FROM stories_orig s
  WHERE col_id = (SELECT MIN(col_id)
                   FROM stories_orig b WHERE s.url = b.url);

                                            QUERY PLAN
----------------------------------------------------------------------------------------------------
 Seq Scan on stories_orig s  (cost=0.00..5339130.21 rows=3069 width=484)
   Filter: (col_id = (SubPlan 1))
   SubPlan 1
     ->  Aggregate  (cost=8.57..8.58 rows=1 width=4)
           ->  Index Scan using story_url_index on stories_orig b  (cost=0.55..8.57 rows=1 width=4)
                 Index Cond: (s.url = url)
```

So, it's about 4 ORDERS OF MAGNITUDE smaller after the indexing.

```
CREATE TABLE q4c AS SELECT * FROM stories_orig s
  WHERE col_id = (SELECT MIN(col_id) FROM stories_orig b
                      WHERE s.url = b.url);
```

5

Write two python programs to fix the data problem. These are two strategies:

5a)

One program sorts the rows of stories_orig.csv and traverse rows in sorted order.

I've included the code here, but there will be a copy on the server in /usr/share/databases/aniskin/a3q5.py

```
# Read the file into an array of lines:
lines = [line.rstrip('\r\n') for line in open('Fanfiction/stories_orig.csv')]
# remove the header and save it for later
header = lines.pop(0)

# Split each line on quotes (so that the odd ones will be the ones in quotes and we can replace any com
lines = [line.split('"') for line in lines]

# Now, let's replace the commas in every odd element by an unused character. It turns out that this fil
def replaceOdds(arr, a, b):
    #python doesn't come with a native copy function, so the line below is a way of copying the list
    retVal = arr[:]
    for i in range(len(retVal)):
        if (i % 2 == 1):
            retVal[i] = retVal[i].replace(a, b).strip()
        else:
            retVal[i] = retVal[i].strip()
    return retVal
lines = [replaceOdds(line, ",", "<") for line in lines]
lines = ["\"".join(line) for line in lines]
lines = [line.split(",") for line in lines]
# Now to sort by url which is at index 11 (from inspection).
lines = sorted(lines, key=lambda arr: arr[11])

# Filter out duplicates
def keepUniques(arr):
    enumArr = list(enumerate(arr))
    return [line for i,line in enumArr if line != arr[i-1]]
lines = keepUniques(lines)
lines = [",".join(line) for line in lines]

f = open('Fanfiction/stories_orig_no-duplicates.csv', 'w')
f.seek(0)
f.truncate()
f.write(header + "\n")
f.write("\n".join(lines))
f.close()
```

5b)

THe other program uses a hash:

```
# Read the file into an array of lines:
lines = [line.rstrip('\r\n') for line in open('Fanfiction/stories_orig.csv')]
# remove the header and save it for later
header = lines.pop(0)

# Split each line on quotes (so that the odd ones will be the ones in quotes and we can replace any com
lines = [line.split('"') for line in lines]

# Now, let's replace the commas in every odd element by an unused character. It turns out that this fil
def replaceOdds(arr, a, b):
    #python doesn't come with a native copy function, so the line below is a way of copying the list
    retVal = arr[:]
    for i in range(len(retVal)):
        if (i % 2 == 1):
            retVal[i] = retVal[i].replace(a, b).strip()
        else:
            retVal[i] = retVal[i].strip()
    return retVal
lines = [replaceOdds(line, ",", "<") for line in lines]
lines = ["\"".join(line) for line in lines]
lines = [line.split(",") for line in lines]

modulus = 2**20 - 1

def hashFunc(line):
    return int('0' + line[15].replace("-","") + line[16]) % modulus

hashTable = [[] for _ in range(modulus)]

# pardon my side effects
def addLineToTable(line, fn, hashTab):
    ind = fn(line)
    hashTab[ind].append(line)

for line in lines:
    addLineToTable(line, hashFunc, hashTable)

retVal = []

for arr in hashTable:
    for i in range(len(arr)):
        if (arr[i] not in arr[:i]):
            retVal.append(arr[i])

retVal = [",".join(line) for line in retVal]

f = open('Fanfiction/stories_orig_no-duplicates2.csv', 'w')
f.seek(0)
f.truncate()
f.write(header + "\n")
f.write("\n".join(retVal))
f.close()
```

5c)

Which program runs faster? Explain.

Well, they both actually run in a comperable amount of time, but I think that's because of all the expensive preprocessing I'm doing and the fact that I used a loop in part B but a list comprehension in part A. List comprehensions are supposed to be super fast in python, but I couldn't quite get the list comprehension to work on the second one (and I've been told by the lords over at StackExchange that it's poor form to have side effects from within a list comprehension).