CSCI143 Final, Spring 2023

Collaboration policy:

You may NOT:

1. discuss the exam with any human other than Mike; this includes:
   (a) asking your friend for clarification about what a problem is asking
   (b) asking your friend if they've completed the exam
   (c) posting questions to github

You may:

1. take as much time as needed
2. use any written notes / electronic resources you would like
3. use the lambda server
4. ask Mike to clarify questions via email

Name:

The questions below relate to the following simplified normalized twitter schema.

```
CREATE TABLE users (
    id_users BIGINT PRIMARY KEY,
    name TEXT UNIQUE NOT NULL,
    description TEXT
);

CREATE TABLE tweets (
    id_tweets BIGINT PRIMARY KEY,
    id_users BIGINT REFERENCES users(id_users),
    created_at TIMESTAMPTZ CHECK (created_at > '2000-01-01'),
    country_code VARCHAR(2) NOT NULL,
    lang VARCHAR(2) NOT NULL,
    text TEXT NOT NULL
);

CREATE TABLE tweet_tags (
    id_tweets BIGINT,
    tag TEXT,
    PRIMARY KEY (id_tweets, tag),
    FOREIGN KEY (id_tweets) REFERENCES tweets(id_tweets)
);
```

1. (8pts) Recall that certain constraints create indexes on the appropriate columns. List the equivalent CREATE INDEX commands that are run by the constraints above.

2. (8pts) Create index(es) so that the following query will run as efficiently as possible. Do not create any unneeded indexes.

   HINT: Pay careful attention to the column list.

```
SELECT DISTINCT tag
FROM tweet_tags
WHERE
    lower(tag) LIKE 'corona%';
```

3. (16pts) Consider the following two queries, which differ only by the conjunction operation used in the WHERE clause.

   For each query: (1) Create index(es) so that the query will run as efficiently as possible. (2) State which scanning strategy you expect the Postgres query planner will use and explain why. (3) Describe which clauses of the query will be sped up with the table scanning strategy you selected in (2), and which clauses (if any) will not be sped up.

   HINT: One of these queries can be implemented very efficiently with an index only scan, and the other query cannot.

   a.
```
SELECT id_tweets
FROM tweets
WHERE country_code = :country_code
  AND lang = :lang
ORDER BY created_at
LIMIT 10;
```

b.
```
SELECT id_tweets
FROM tweets
WHERE country_code = :country_code
   OR lang = :lang
ORDER BY created_at
LIMIT 10;
```

4. (8pts) Create index(es) so that the following query will run as efficiently as possible. Do not create any unneeded indexes.

```
SELECT count(*)
FROM tweets
JOIN tweet_tags USING (id_tweets)
WHERE
    tag = :tag;
```

5. (8pts) Create index(es) so that the following query will run as efficiently as possible. Do not create any unneeded indexes.

```
SELECT name, count(*)
FROM users
JOIN tweets USING (id_users)
JOIN tweet_tags USING (id_tweets)
WHERE tag = :tag
GROUP BY name;
```

6. (16pts) The following query returns tweets where either the text or the description of the user match a full text search query.

```
SELECT id_tweets
FROM tweets, users
WHERE ( to_tsvector('english', text)
     || to_tsvector('english', description)
      )
      @@ to_tsquery('english', :query);
```

   a. This query cannot be sped up using an index. Why?

b. Rewrite the query above into an equivalent query that can be sped up with an index. Also provide the index that would speed up the query and explain why the modified query can be sped up.