Final Practice Problems

All of the problems assume the following schema.

```
CREATE TABLE users (
    id_users BIGINT PRIMARY KEY,
    created_at TIMESTAMPTZ,
    username TEXT UNIQUE NOT NULL
);

CREATE TABLE tweets (
    id_tweets BIGINT UNIQUE NOT NULL,
    id_users BIGINT REFERENCES users(id_users),
    in_reply_to_user_id BIGINT REFERENCES users(id_users),
    created_at TIMESTAMPTZ,
    text TEXT
);

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

1. Recall that certain constraints create indexes on the appropriate columns. List the equivalent `CREATE INDEX` commands that are run by these constraints.

2. List the scan methods applicable for the following SQL query.

```
SELECT count(*)
FROM users
WHERE id_users >= :min_id_users
  AND username >= :min_username
  AND username <  :max_username
```

3. Create index(es) so that the following query can use an index only scan.

   Do not create any unneeded indexes; if no new indexes are needed, say so.

```
SELECT count(*)
FROM users
WHERE lower(username)=:username;
```

4. Create index(es) so that the following query can use an index only scan, avoid an explicit sort, and take advantage of the LIMIT clause for faster processing.

   Do not create any unneeded indexes; if no new indexes are needed, say so.

```
SELECT id_users
FROM users
WHERE created_at<=:created_at
ORDER BY created_at DESC
LIMIT 10;
```

5. Create index(es) so that the following query will run as efficiently as possible.

   Do not create any unneeded indexes; if no new indexes are needed, say so.

```
SELECT id_users,count(*)
FROM users
JOIN tweets USING (id_users)
JOIN tweet_tags ON (id_tweets)
WHERE tag LIKE :tag_prefix || '%'
GROUP BY id_users;
```

6. Consider the following SQL query.

```
SELECT tag,count(*) AS count
FROM tweet_tags
GROUP BY tag
ORDER BY count DESC
LIMIT 10;
```

a) Create index(es) so that the following query will run as efficiently as possible. Do not create any unneeded indexes; if no new indexes are needed, say so.

b) Given the indexes created above, is the LIMIT clause likely to significantly speed up the query? Explain why/why not.

7. Consider the following SQL query.

```
SELECT id_tweets
FROM tweets
WHERE to_tsvector(text) @@ to_tsquery(:tsquery)
  AND created_at > '2020-01-01'
ORDER BY created_at <=> '2020-01-01'
LIMIT 10;
```

a) Create index(es) so that the following query can use an index scan, avoid an explicit sort, and take advantage of the LIMIT clause for faster processing.

Do not create any unneeded indexes; if no new indexes are needed, say so.

b) Assume that you are unable to install the RUM extension into the database. (This could be because you don't have admin permissions, for example.) How would that change your answer above?

8. Consider the following SQL query.

```
SELECT *
FROM (
    SELECT
        id_tweets,
        unnest(tsvector_to_array(to_tsvector(text))) as lexeme
    FROM tweets
) t
WHERE lexeme = :lexeme;
```

a) The query above cannot be sped up using an index. Why?

b) Rewrite the query from the previous question into an equivalent query that can be sped up using an index. Also provide the index that would speed up the query.

9. Consider the following SQL query that uses a cross join.

```
SELECT id_tweets
FROM users, tweets
WHERE to_tsvector(text) @@ to_tsquery(username)
  AND users.id_users = :id_users;
```

a) Which join methods can be used to implement this query?

b) If it is possible to construct an index that will speed up this query, do so. Otherwise, state that it is impossible and explain why.