Final Practice Problems 1

All of the problems assume the following schema.

```
CREATE TABLE users (
    id_users BIGINT PRIMARY KEY,
    created_at TIMESTAMPTZ,
    username TEXT
);

CREATE TABLE tweets (
        id_tweets BIGINT PRIMARY KEY,
        id_users BIGINT REFERENCES users(id_users),
        in_reply_to_user_id BIGINT REFERENCES users(id_users),
        created_at TIMESTAMPTZ,
        text TEXT
);

CREATE TABLE tweets_mentions (
        id_tweets BIGINT REFERENCES tweets(id_tweets),
        id_users BIGINT REFERENCES users(id_users),
        PRIMARY KEY (id_tweets, id_users)
);
```

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 all of the scan methods that could possibly be used for the following SQL query.

```
SELECT username FROM tweets WHERE id_user=:id_user;
```

3. List all of the scan methods that could possibly be used for the following SQL query.

```
SELECT id_users,username FROM users WHERE id_user=:id_user;
```

4. Explain why the following SQL query is likely to be inefficient, and create an index that will speed up the query.

```
SELECT id_tweets
FROM tweets_mentions
WHERE id_users=:id_users;
```

5. Create index(es) so that the following query could use an index only scan.

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

```
SELECT count(*)
FROM tweets
WHERE id_user=:id_user
  AND created_at < :hi
  AND created_at >= :lo;
```

6. 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_tweets, created_at
FROM tweets
WHERE id_users=:is_users
ORDER BY created_at DESC
LIMIT 10;
```

7. Construct index(es) so that the following query can use an index only scan, and the `users(username)` column will have a UNIQUE constraint.

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

```
SELECT created_at FROM users WHERE username=:username;
```

8. Construct a single index so that the following query can be answered as quickly as possible.

```
SELECT id_tweets
FROM tweets
WHERE id_user=:id_user
  AND created_at >= '2020-01-01 00:00:00'
  AND created_at <  '2021-01-01 00:00:00'
ORDER BY
  created_at ASC,
  id_reply_to_user_id DESC
```

9. Construct index(es) to speed up the following JOIN, assuming a merge join is used.

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

```
SELECT *
FROM tweets_mentions
JOIN tweets USING (id_tweets);
```

10. Construct index(es) to speed up the following JOIN, assuming a merge join is used. Your index(es) should take advantage of the WHERE clause.

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

```
SELECT id_tweets
FROM tweets_mentions
JOIN users USING (id_users)
WHERE username=:username;
```

11. Your goal is to answer the following query quickly.

```
SELECT count(*)
FROM tweets
WHERE id_user=:id_user
  AND created_at >= '2020-01-01 00:00:00';
```

You have the option of creating either of the following two indexes:

```
CREATE INDEX tweets_idx1 ON tweets(id_user)
        WHERE created_at >= '2020-01-01 00:00:00';
CREATE INDEX tweets_idx2 ON tweets(id_user,created_at);
```

If the tweets table is large (several TBs), which index will result in the fewest page reads when answering the SELECT query? Why?

Which index will use the least amount of disk space?

12. You are considering adding more information to the tweets table by redefining the schema as:

```
CREATE TABLE tweets (
    id_tweets BIGINT PRIMARY KEY,
    id_users BIGINT REFERENCES users(id_users),
    created_at TIMESTAMPTZ,
    in_reply_to_status_id BIGINT,
    in_reply_to_user_id BIGINT REFERENCES users(id_users),
    quoted_status_id BIGINT,
    retweet_count SMALLINT,
    favorite_count SMALLINT,
    quote_count SMALLINT,
    withheld_copyright BOOLEAN,
    withheld_in_countries VARCHAR(2)[],
    source TEXT,
    text TEXT,
    country_code VARCHAR(2),
    state_code VARCHAR(2),
    lang TEXT,
    place_name TEXT,
    geo geometry
);
```

How will this affect the number of pages accessed during (and therefore the runtime of) a

1. sequential scan?

2. index only scan?

3. index scan?

4. bitmap scan?

5. GroupAggregate?

6. HashAggregate?

7. Nested Loop join?

8. Hash join?

9. Merge join?