CSCI143 Final, Spring 2022

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:

---

1 True/False Questions

For each question below, circle either True or False. Each correct answer will result in +2 points, each incorrect answer will result in -2 points, and each blank answer in 0 points.

1. True False A table that takes up 832KB on disk has 109 pages.
2. True False An index only scan may have to access heap table pages to determine whether a tuple is visible. This is likely to happen on a table that has had many recent UPDATE/DELETE operations and no recent VACUUM operation.
3. True False Every heap page must have at least one live tuple.
4. True False If you run the TRUNCATE command to delete the contents of a table, you must run a subsequent VACUUM FULL command to free up the disk space for other processes to use.
5. True False Postgres automatically compresses large TEXT values.
6. True False Postgres documentation recommends disabling autovacuum if you encounter the transaction id wraparound problem.
7. True False Decreasing the `fillfactor` for a table from the default value of 100 will make HOT tuple updates more likely.
8. True False The autovacuum process runs the VACUUM FULL command at regular intervals in order to automatically free up disk space from dead tuples.
9. True False A btree index created on an SMALLINT column will have higher fanout than the same index created on a BIGINT column.
10. True False In the postgres documentation, TID is an abbreviation for transaction identifier.
11. True False Dirty reads are possible in Postgres's read committed isolation level.
12. True False For very small tables, the postgres query planner is likely to choose sequential scan instead of an index scan.
13. True False The hash index supports the bitmap scan access method.
14. True False A database stored using HDDs should have a higher value for the `random_page_cost` system parameter than a database stored using SSDs.
15. True False A denormalized representation of data tends to take up less disk space than a normalized representation.
16. True False The nested loop join strategy can be used to join tables on an equality constraint.

17. True False The hash join strategy can be used for self joins.

18. True False A hash index can be used to speed up a nested loop join.

19. True False A btree index can be used to speed up a CHECK constraint.

20. True False It is possible to INSERT a NULL value into a column labeled as the PRIMARY KEY.

21. True False It is not possible to create a GIN index to enforce a UNIQUE constraint, but it is possible to create a RUM index to enforce a UNIQUE constraint.

22. True False One advantage of the RUM index over the GIN index is that the former supports index scans and the latter does not. This implies that the RUM index can be used to speed up queries using the `LIMIT` clause, but the GIN index cannot.

23. True False If postgres crashes while a DELETE/INSERT/UPDATE statement is modifying a RUM index, the index becomes corrupted and must be regenerated.

24. True False The ANALYZE command collects statistics on the values in the table which the query planner uses when selecting which scan algorithm to use for a query.

25. True False A hash index can return tuples in sorted order.

26. True False The GroupAggregate algorithm can be used if one of the SELECT columns contains COUNT(DISTINCT *).

27. True False When a table has been CLUSTERed on an index, inserting new tuples causes them to be inserted in the order specified by the index.

28. True False In the current version of postgres (versrion 14), index only scans can be parallelized, but index scans cannot be parallelized.

29. True False If you run EXPLAIN ANALYZE on an UPDATE statement, then the database is guaranteed not to modify your data.

30. True False The order of columns matters in a multi-column GIN index.

2 Integrated Questions

The questions below relate to the following simplified normalized twitter schema. There are 12 subproblems in total, each worth 5 points.

```
CREATE TABLE users (
    id_users BIGINT PRIMARY KEY,
    created_at TIMESTAMPTZ CHECK (created_at > '2000-01-01'),
    name TEXT UNIQUE NOT NULL,
    description TEXT
);

CREATE TABLE tweets (
    id_tweets BIGINT PRIMARY KEY,
    id_users BIGINT REFERENCES users(id_users),
    retweet_count SMALLINT NOT NULL,
    country_code VARCHAR(2),
    lang VARCHAR(2) NOT NULL,
    text TEXT NOT NULL
);

CREATE TABLE tweet_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 the constraints above.

2. 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 and explain why.

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

3. 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 and explain why.

```
SELECT max(retweet_count)
FROM tweets
WHERE
    id_users = :id_users;
```

4. 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 and explain why.

```
SELECT DISTINCT id_users
FROM tweets
WHERE country_code = :country_code
   OR lang = :lang
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 and explain why.

```
SELECT id_users, id_tweets
FROM tweets
WHERE retweet_count > :minimum_retweet_count
  AND country_code = :country_code
ORDER BY retweet_count DESC, id_users ASC
LIMIT 10;
```

6. 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 and explain why.

```
SELECT name
FROM users
JOIN tweet_mentions USING (id_users)
WHERE
    id_tweets = :id_tweets
ORDER BY name;
```

7. 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 and explain why.

```
SELECT country_code, count(*)
FROM tweets
JOIN users USING (id_users)
WHERE name = :name
GROUP BY country_code
ORDER BY count(*) DESC;
```

8. 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 and explain why.

```
SELECT count(*)
FROM tweets
WHERE retweet_count > 0
  AND country_code = :country_code
  AND to_tsvector('english', text) @@ to_tsquery('english', :query)
LIMIT 10;
```

9. Consider the following SQL query.

```
SELECT count(*)
FROM users
WHERE name ILIKE '%Smith'
```

   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.

10. Consider the following SQL query, which returns tweets where either the text or the description of the user match a FTS 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.