INFO 6210

Data Management and Database Design

Practice Exam One SQL Solutions

Student Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Professor: Nik Bear Brown

Rules:

1. NO COMPUTER, NO PHONE, NO DISCUSSION or SHARING.

2. Ask if you don't understand a question.

3. Time allowed. Last 90 minutes of class

5. Bring pen/pencil. The exam will be written on paper.

All of the questions in the exam refer to the Twitter database shown below

CREATE TABLE IF NOT EXISTS "tweets" (

"tweet\_id" varchar(25) NOT NULL,

"user\_id" varchar(25) NOT NULL,

"tweet\_content" varchar(300) NOT NULL,

"tweet\_datetime" datetime NOT NULL DEFAULT CURRENT\_TIMESTAMP,

"favorites" int,

"retweets" int,

"urls" varchar(100),

PRIMARY KEY("tweet\_id")

);

CREATE TABLE IF NOT EXISTS "users" (

"user\_id" varchar(25) NOT NULL,

"user\_name" varchar(50) NOT NULL,,

"followers" int,

"following" int,

"tweet\_count" int,

PRIMARY KEY("user\_id")

);

CREATE TABLE IF NOT EXISTS "tags" (

"tag\_id" integer NOT NULL PRIMARY KEY AUTOINCREMENT,

"tag" varchar(50) UNIQUE

);

CREATE TABLE IF NOT EXISTS "tweet\_tags" (

"tweet\_id" varchar(25),

"tag\_id" varchar(25)

);

Q1 (5 Points)

Show the user\_id, user\_name. and followers of the top 5 users by number of followers.

Solution:

SELECT user\_id, user\_name, followers from users ORDER BY followers DESC limit 5

A screenshot of a cell phone

Description automatically generated

Q2 (5 Points)

Get the top 7 days with the highest total number of retweet..

Solution:

NB: Use the DATE(tweet\_datetime) function to get a date from tweet\_datetime

SELECT DATE(tweet\_datetime) AS tweet\_date, sum(retweets) as sum\_rt from tweets Group BY tweet\_date ORDER BY sum\_rt DESC limit 7

A screenshot of a cell phone

Description automatically generated

Q3 (5 Points)

Create a column that is calculated from other columns in the tweets table.

Solution:

SELECT tweet\_id,retweets, favorites, retweets+ favorites as popularity from tweets limit 15

A screenshot of a cell phone

Description automatically generated

Q4 (5 Points)

Count all of the null values in a nullable field.

Solution:

SELECT count(\*) from tweets where urls=NULL;

A close up of a logo

Description automatically generated

Q5 (5 Points)

Computationally what is the most expensive operation in the relational data model?

Solution:

SQL joins are computationally the most expensive operation in relational databases.

If there is no indexing in the table, every record in the 1st table must be compared with each record of 2nd table which gives us the complexity of O(M\*N)

A cross joint will always be O(N^2), since it has to result in N^2 records.

Q6 (5 Points)

Write a function to get the numbers of retweets for a given user\_id.

Solution:

DELIMITER //

CREATE FUNCTION get\_retweets(idm varchar(25))

RETURNS FLOAT

BEGIN

DECLARE pm FLOAT;

SET pm:=0;

SELECT SUM(retweets) into pm from tweets where user\_id=idm;

RETURN pm;

END//

DELIMITER ;

Q7 (5 Points)

Why not put all the data in one big table and avoid all of these joins?

Solution:

Relationship

If the relationship between multiple columns is one-to-one then it better to store the data in one table, since it reduces the number of joins the table has to do.

If the relationship between two tables in one-to-many, then it will be better to split into separate tables to reduce duplicate data. Duplicate data wastes lots of storage and cache space and makes database harder to maintain.

Computation

We should always start by 3NF form and only deformalize if we find a specific performance problem.

Also, by storing all the data in a single table, makes querying the table expensive as querying a large table for a single value is more costly than querying two small tables.

Also, by storing it in one table and duplicating data, we run a risk of allowing inconsistent data to be inserted into database thus nullifying one of the core properties of Relational databases (ACID)

When we store all the data in a single large table, if a remote server needs a single value for the table, entire table needs to be transferred through network, which increases network transmission times. If the data is stored in normalized form, this can be avoided.

Thus, even though Joins are expensive, there are many factors to be considered while designing a database. Joins are needed only when we need data from two tables, where rest of the above reasons together combined can be more expensive than joins. Also, we can reduce repeated joins by creating a view of the table with joins and query it.

Q8 (5 Points)

Why create views?

Solution:

Creating views has several benifits.

1. Views can hide complexity

If we have a query thar requires joining several tables or has a comples logic/ calculations, we can create a view and query it just like tables. THus, a view is encapsulation of a complex or expensive query.

2. Views can be used as a security mechanism

With a view, we can select certain columns and/or rows from a table, and can set permissions on the view instead of the underlying tables. This allows surfacing only the data that a user needs to see and rest of the data can be hidden from the user.

3. To Denormalise data for reporting purpose

We can use a view to denormalise and/ or aggregate the tables. This results in reducing the redundancy in writing the queries also maximizing the performance of the database.

4. Refactoring Database

We can hide the change so that the old code does not see it by creating a view.

Q9 (5 Points)

Show the tweet\_id, user\_name, and tweet length of the 5 longest tweets.

Solution:

SELECT tweet\_id, user\_name, LENGTH(tweet\_content) AS tweet\_length from tweet t, user u WHERE t.user\_id=u.user\_id ORDER BY tweet\_length DESC limit 5;

A screenshot of a cell phone

Description automatically generated

Q10 (5 Points)

Create a view called ‘longest\_tweets’ using the SQL in Q9

Solution:

CREATE VIEW longest\_tweets AS SELECT tweet\_id, user\_name, LENGTH(tweet\_content) AS tweet\_length from tweet t, user u WHERE t.user\_id=u.user\_id ORDER BY tweet\_length DESC limit 5;

Q11 (5 Points)

Why create temporary tables?

Solution:

A Temporary Table is only visible to the current session and is dropped automatically once the session ends. This means that two different sessions can use the same temporary table name without conflicting with each other or with an existing non-temporary table of the same name.

This has several advantages,

We can pull data from various tables, do some work on that data and then combine everything into one result set.

We can use a temporary table for tilting the data i.e. turning rows to columns, etc. which we need for advance processing.

Q12 (5 Points)

Write SQL to insert a new user into the user table.

Solution:

CREATE TABLE IF NOT EXISTS "users" (

"user\_id" varchar(25) NOT NULL,

"user\_name" varchar(50) NOT NULL,,

"followers" int,

"following" int,

"tweet\_count" int,

PRIMARY KEY("user\_id")

);

INSERT INTO users VALUES('@Bear','Bear', 100000000,10 , 9999);

Q13 (5 Points)

Create a temporary table called ‘longest\_tweets\_tmp’ using the SQL in Q9.

Solution:

CREATE TEMPORARY TABLE longest\_tweets\_tmp AS SELECT tweet\_id, user\_name, LENGTH(tweet\_content) AS tweet\_length from tweet t, user u WHERE t.user\_id=u.user\_id ORDER BY tweet\_length DESC limit 5;

Q14 (5 Points)

Update the user you added in Q12 to have 55 following.

Solution:

UPDATE users SET following=55 where user\_id='@Bear'"

Q15 (5 Points)

Update all users with null or negative followers to have 0 followers.

Solution:

UPDATE users SET followers=0 where followers < 0 or followers=NULL;

Q16 (5 Points)

Delete a user with the user\_id of '@Spammy' and then check that the user is removed.

Solution:

DELETE FROM users WHERE user\_id='@Spammy'

SELECT COUNT(\*) FROM users WHERE user\_id='@Spammy'

Q17 (5 Points)

Create a procedure called tag\_match that shows the tags in the tags table that start with a given text string.

Solution:

CREATE TABLE IF NOT EXISTS "tags" (

"tag\_id" integer NOT NULL PRIMARY KEY AUTOINCREMENT,

"tag" varchar(50) UNIQUE

);

DELIMITER //

CREATE PROCEDURE tag\_match(t varchar(50))

BEGIN

SELECT DISTINCT tag from tags where tag LIKE t%;

END//

DELIMITER ;

Q18 (5 Points)

Create a index on a non-key attribute.

Solution:

CREATE TABLE IF NOT EXISTS "users" (

"user\_id" varchar(25) NOT NULL,

"user\_name" varchar(50) NOT NULL,,

"followers" int,

"following" int,

"tweet\_count" int,

PRIMARY KEY("user\_id")

);

CREATE INDEX users\_followers\_idx ON users(followers);

Q19 (5 Points)

Create a table called ‘user\_log’ with the user id and a log\_date field that represents the current date and time. The primary key is the two fields in the table.

The create a trigger that will use the user\_log table to record the user\_id and current date and time when a new user is inserted into the user table.

Solution:

CREATE TABLE IF NOT EXISTS "user\_log" (

"user\_id" varchar(20),

"log\_date" datetime NOT NULL DEFAULT CURRENT\_TIMESTAMP,

);

PRIMARY KEY("user\_id","log\_date");

CREATE TRIGGER USERLOG AFTER INSERT

ON user

BEGIN

INSERT INTO user\_log values(new.user\_id, NOW());

END;

Q20 (5 Points)

Create a transaction that runs the SQL in Q16. That is, delete a user with the user\_id of '@Spammy' but only finalize the SQL if only one user is removed from the users table.

Update the SQL check to make sure that it counts how many users were removed if it isn’t already doing that. Only complete the transaction if only one user is removed

Assume you are doing this from the command line but can write a stored procedure to do this if you wish.

Solution:

BEGIN; (or START TRANSACTION;)

SELECT COUNT(\*) FROM users;

# Remember the count (note this is better done as a stored procedure)

DELETE FROM users WHERE user\_id='@Spammy'

SELECT COUNT(\*) FROM users;

# If you see the count is one less the commit (note this is better done as a stored procedure)

COMMIT;

# Otherwise rollback

ROLLBACK;