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

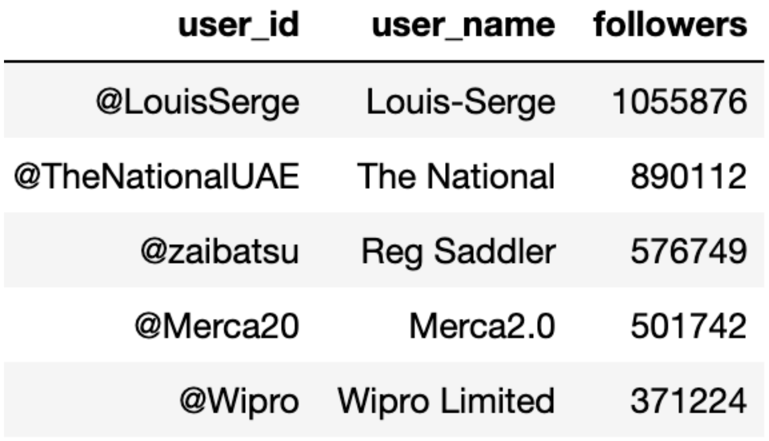
```
CREATE TABLE IF NOT EXISTS "tweets" (
	"tweet_id"	varchar(25) NOT NULL,
	"user_id"	varchar(25) NOT NULL,
	"tweet_content"	varchar(300) NOT NULL,
	"tweet_datetime" datetime NOT NULL DEFAULT CURRENT_TIMESTAMP,
	"favorites"	int,
	"retweets"	int,
	"urls"	varchar(100),
	PRIMARY KEY("tweet_id")
);

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
```



user_id	user_name	followers
@LouisSerge	Louis-Serge	1055876
@TheNationalUAE	The National	890112
@zaibatsu	Reg Saddler	576749
@Merca20	Merca2.0	501742
@Wipro	Wipro Limited	371224

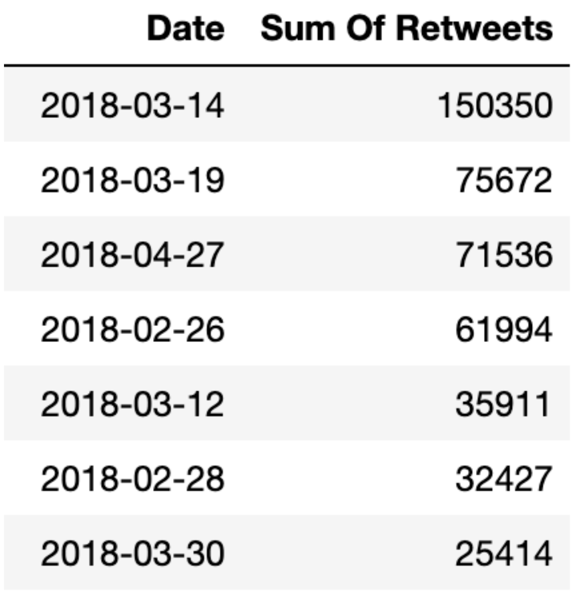
Q2 (5 Points)

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
```



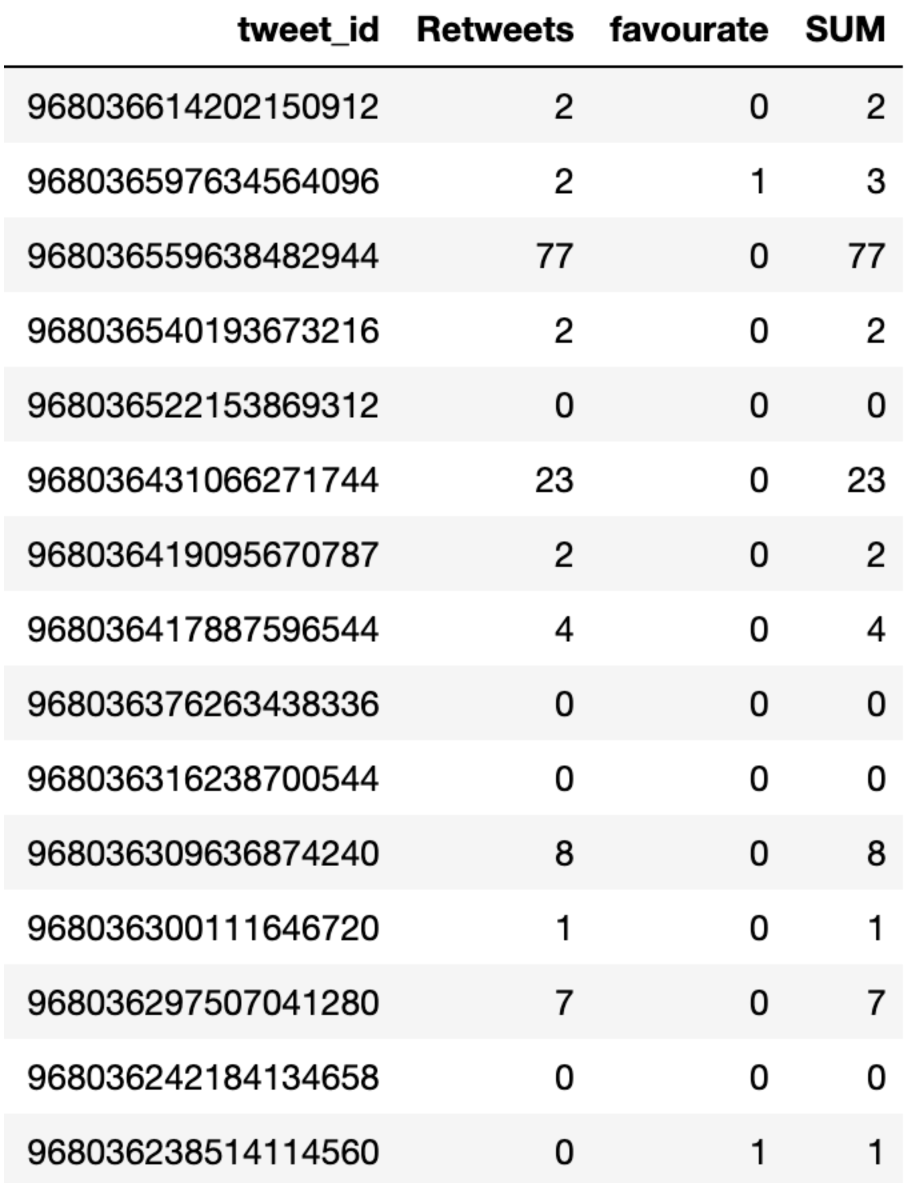
Date	Sum Of Retweets
2018-03-14	150350
2018-03-19	75672
2018-04-27	71536
2018-02-26	61994
2018-03-12	35911
2018-02-28	32427
2018-03-30	25414

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
```



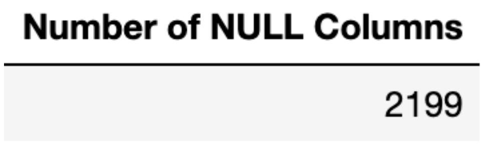
tweet_id	Retweets	favourate	SUM
968036614202150912	2	0	2
968036597634564096	2	1	3
968036559638482944	77	0	77
968036540193673216	2	0	2
968036522153869312	0	0	0
968036431066271744	23	0	23
968036419095670787	2	0	2
968036417887596544	4	0	4
968036376263438336	0	0	0
968036316238700544	0	0	0
968036309636874240	8	0	8
968036300111646720	1	0	1
968036297507041280	7	0	7
968036242184134658	0	0	0
968036238514114560	0	1	1

Q4 (5 Points)

Count all of the null values in a nullable field.

Solution:

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



Number of NULL Columns
2199

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.

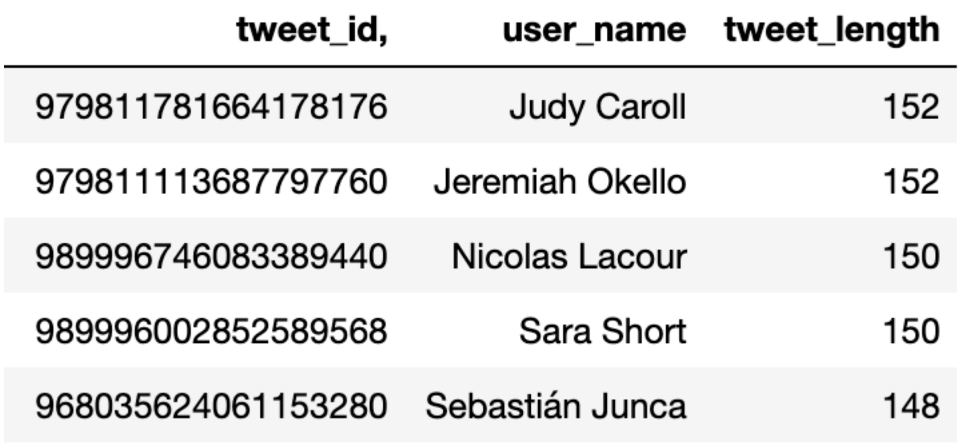
4. Refactoring Database
We can hide the change so that the old code does not see it by creating a view.

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;
```



tweet_id,	user_name	tweet_length
979811781664178176	Judy Caroll	152
979811113687797760	Jeremiah Okello	152
989996746083389440	Nicolas Lacour	150
989996002852589568	Sara Short	150
968035624061153280	Sebastián Junca	148

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;
```