



University of Ontario Institute of Technology
Department of Electrical, Computer, and Software Engineering
Faculty of Engineering and Applied Science

SOFE 3700
Data Management Systems
Final Project

Make Twitter Great Again

Group 17

Member(s)		
First Name	Last Name	Student ID
Georgy	Zakharov	100588814
Justin	Kaipada	100590167
Savan	Patel	100583384
Karn	Bhavsar	100557957

README – 5

Instructions for installation on Windows:

Download and extract all files into the folder of your choice.

Open RawData folder.

Run getTweet.py script to scrape twitter for data into the data.csv file.

//create database?

Run the populate.py script to populate your database.

Back to parent directory, open FrontEndAPI.

Run web.py to access front-end.

Select queries you would like to make and submit them.

Content – 20

For this project our target is to collect data about Trump's tweets and store various information about them, this information can then be accessed to provide a variety of data through the front-end API. To achieve this we will be using HTML5 + CSS3 for the Front-end and Python + JavaScript for the back-end. We are accessing Twitter's API in order to scrape the raw data into a .csv file which will be read by a population script to fill up the database, PostgreSQL + pgadmin3 are used to build and maintain the database.

During implementation our difficulties with the front-end were mainly with hosting and formatting, because none of us have had experience with graphical design we needed to opt for a simple and functional design rather than anything professional-like and graphically impressive, with regards to hosting we chose to forego this route due to time constraints.

For back-end difficulties we had trouble figuring out the different frameworks needed to put together the functionality for the data scraping and populating of the database. We initially looked to use JSON but the querying of the Twitter API and JSON use proved more difficult than dumping the raw data into a .csv which is the method we ended up choosing.

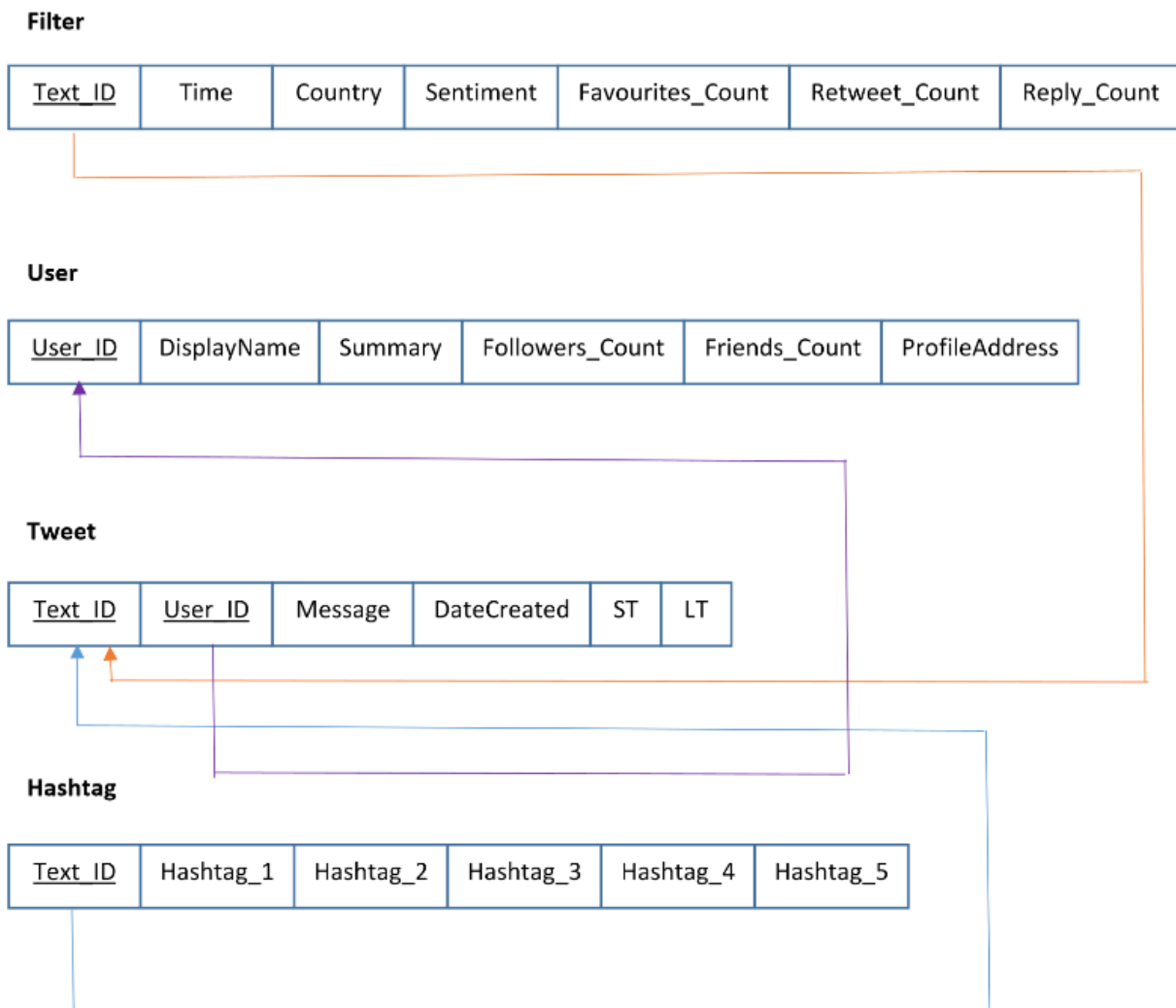
Given another swing at the project we would do a few things differently, namely starting earlier in the semester with familiarizing ourselves with the different frameworks, this would allow us to make significant progress quicker when writing/modifying the code. Additionally, it we could look into finding royalty-free design templates to be used for the front-end display in order to achieve a more clean and professional look. Lastly, we can look into hosting the website to allow users access to the database through the web instead of local setups.

Possible future improvements include adding additional users and their tweets to the database (besides Trump), include additional user requested information fields, and expand on the types of information offered by implementing a photo/video storage solution.

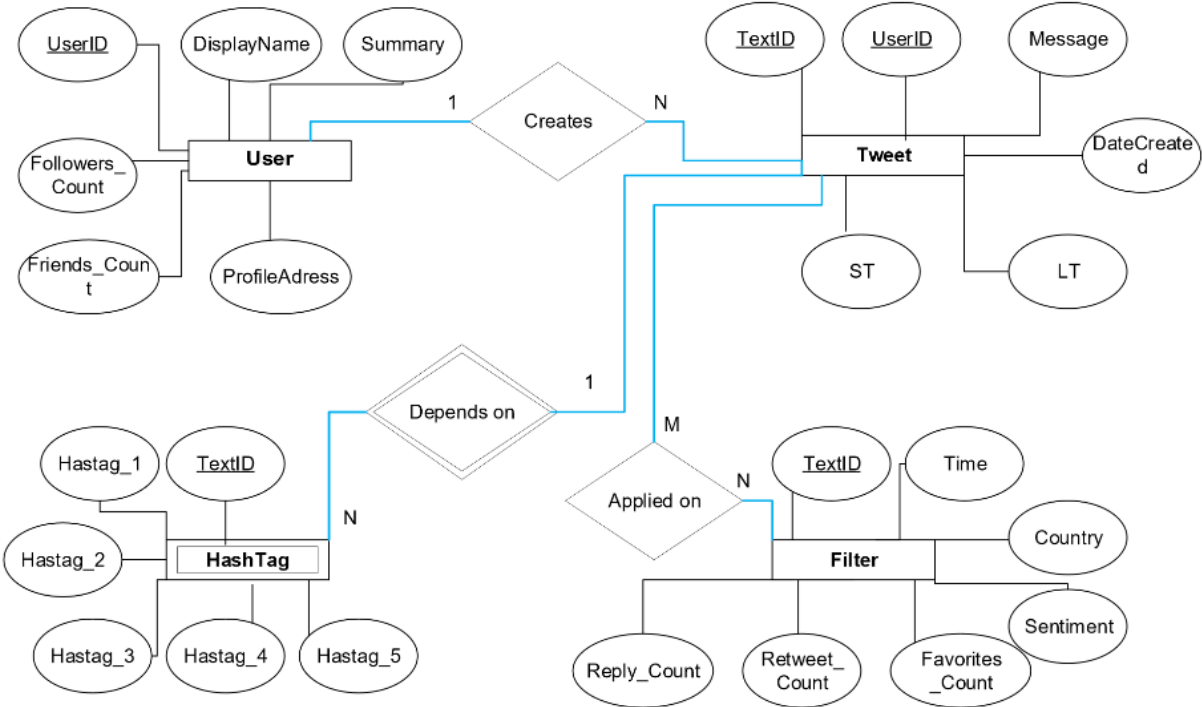
Overall, we are satisfied with the quick and effective access of the data from the Twitter API and the wide selection of data that we could choose from, this wide selection allows us to provide different views of the data which can be useful to see trends and patterns.

Some inconsistencies may be expected in the future if Twitter significantly changes its API functionalities.

Diagrams – 10



ER Schema Diagram



Query Snapshots – 5

View 1: Computes a join of at least three tables

```
SELECT *  
FROM Filter as F INNER JOIN Tweet as T  
ON F.Text_ID = T.Text_ID  
INNER JOIN Users as U  
ON T.User_ID = U.User_ID
```

View 2: Uses nested queries with the ANY or ALL operator and uses a GROUP BY clause

```
SELECT DisplayName  
FROM Users  
WHERE User_ID = ANY (SELECT User_ID FROM Tweet WHERE User_ID > 10)  
GROUP BY DisplayName
```

View 3: A correlated nested query

```
SELECT AVG(Reply_Count)  
FROM Filter, Tweet  
WHERE Text_ID = Tweet.Text_ID
```

View 4: Uses a FULL JOIN

```
SELECT *  
FROM Tweet  
FULL OUTER JOIN Users  
ON Tweet.Text_ID = Users.User_ID
```

View 5: Uses nested queries with any of the set operations UNION, EXCEPT, or INTERSECT

```
SELECT User_ID FROM Tweet  
UNION  
SELECT User_ID FROM Users  
ORDER BY User_ID
```

View 6: Computes average retweets count

```
SELECT AVG (Retweet_Count)
FROM Filter, Tweet
WHERE Text_ID = Tweet.Text_ID
```

View 7: Computes highest reply count

```
SELECT COUNT (DISTINCT Reply_Count)
FROM Filter
```

View 8: Finds highest amount of tweets in a single day

```
SELECT COUNT (DISTINCT Text_ID)
FROM Tweet
GROUP BY DateCreated
```

View 9: Shows first tweet, and latest tweet

```
SELECT TOP 1 *
FROM Tweet
ORDER BY DateCreated DESC
```

```
SELECT TOP 1 *
FROM Tweet
ORDER BY DateCreated ASC
```

View 10: Count total tweets

```
SELECT COUNT (Text_ID)
From Tweet
```