Revision: 10/04/2019
Changes from the previous version are highlighted in yellow.

COMS 363 Fall 2019

Project: Web database application for investigating what our state legislators said on Twitter.

Percentage in your final grade: 30%
Total points: 300

Submission due dates:

Part I: Thursday October 24, 2019 by 11:50pm
Submission of deliverables described in Tasks 1 and 2 of "Tasks and deliverables"

Part II: Friday December 13, 2019 by 11:50pm
Submission of deliverables described in Tasks 3-5 of "Tasks and deliverables"

Per course policy, late submissions are not graded.

If your team member does not contribute to the project, inform the instructor as early as possible, but no later than October 24, 2019.

This project asks a three-student team to develop a web database application for social scientists and journalists (clients) to examine Twitter communications of state legislators and presidential candidates around the 2016 presidential election. The web application accesses data from two different backend DBMS: MySQL and Neo4j. The team is given tweets of these Twitter user accounts (handlers) in two different formats: 1) csv files and 2) Neo4j graph database. The team's tasks are as follows.

- Design an ER diagram from the given database requirement.
- Design relational schemas to store the data from the csv files and import the data from the csv files into MySQL Server.
- Write SQL queries for the questions specified in this project description.
- Fine tune SQL query performance and report the performance difference before and after query optimization. Investigate the query performance between MySQL and Neo4j.
- Ensure that only authorized users are able to access the SQL database.
- Implement the web application as a set of JSP pages running on Apache Tomcat server to send SQL statements and Cypher queries to the backend databases. The web application shall run on the Google Cloud VM on one of the team members.
- Evaluate your team members about their contributions to the project.

Database Requirement

- A tweet has the following properties: id, retweet_count (the number of retweets of this tweet), retweeted (whether this tweet has been retweeted), tweet text, created_at (timestamp---number of milliseconds since 1/1/1970---in which the tweet was posted), which is converted into the corresponding day, month, and year. The id is unique among all tweets. A tweet must have the user who posted it. If the posting user is deleted, all tweets posted by the user must also be deleted. A retweet has the same tweet text, but has a different id from the original tweet. The attribute retweeted is not needed in the database.

Revision: 10/04/2019
Changes from the previous version are highlighted in yellow.

- A tweet may have zero or more hashtag in it. Each hashtag has a unique name and must be used in at least one tweet. A tweet has zero or more URLs. Each URL has a unique URL address and must appear in at least one tweet. If a hashtag is deleted, all tweets that have the deleted hashtag must also be deleted. If a URL is deleted, all the tweets that have the deleted URL must also be deleted.
- A tweet may mention zero or more user accounts. A user account can be mentioned in zero or more tweets.
- A user account has the following properties: name, screen name, followers (indicating the number of followers), following (indicating the number of people this user follows), sub_category, category, location, and the state the user lives. A user lives in at most one state. The sub_category indicates the party to which the user belongs. The values of this attribute are "GOP", "Democrat", "na", or null. The values of the category attribute are senate_group, presidential_candidate, reporter, Senator, General, or null. The name property can be null, but the screen name is unique among all users and cannot be null. Presidential candidate accounts are not associated with any state. The state of Florida appears as "Florida" or "FL". Furthermore, "na" denotes the user account without an associated state. Check the data from the csv files to understand the kind of data to be maintained.

Neo4j Graph Database

The above database requirement was modeled using a property graph data model. Figure 1 shows the conceptual schema of the graph database. Each rectangle represents a group of nodes with similar properties. We use Neo4j labels to implement the grouping. The node labels are State, User, Century, Year, Month, Day, Tweet, Url, and Hashtag. In the diagram, User (Posted) and User (Mentioned) both have the same User label. The edge labels are FROM, MENTIONED, POSTED, URL_USED, TAGGED, HAS_TWEET, HAS_YEAR, HAS_MONTH, and HAS_DAY. Edges in this database have no attributes.

Revision: 10/04/2019
Changes from the previous version are highlighted in yellow.

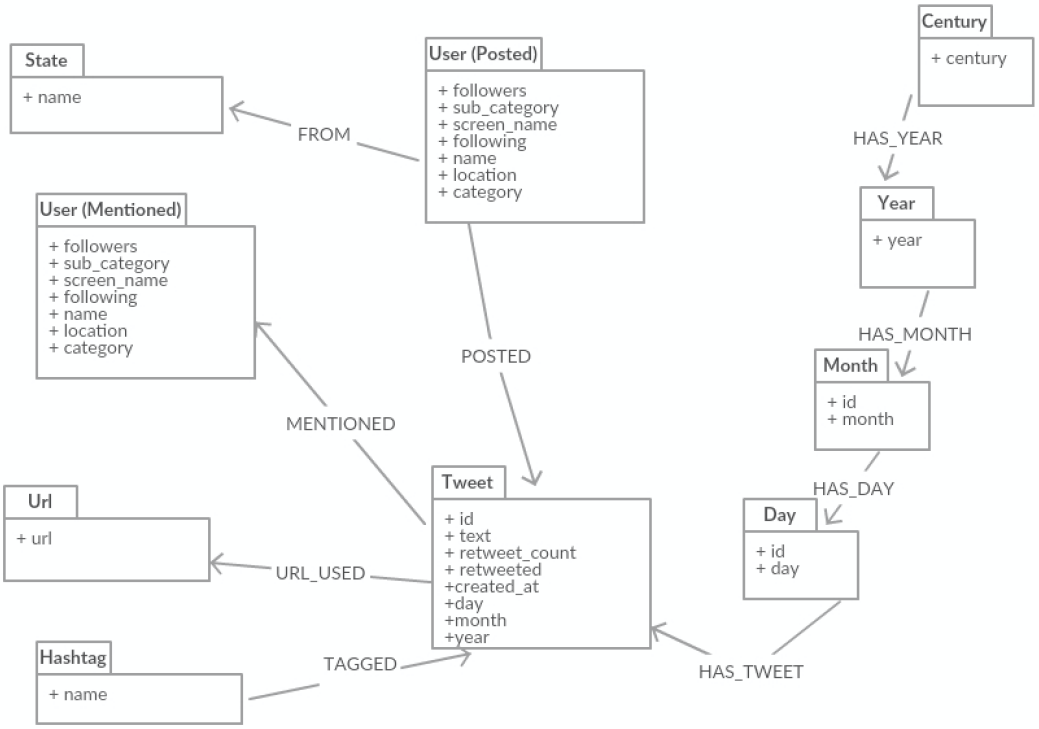

Figure 1: Schema of the graph database of tweets of state legislatures and presidential candidates for the 2016 presidential election.

The design of this data model follows the basic guidelines for graph database design. Nodes represent entities and a directed edge represents a relationship between two nodes. Nodes also represent properties of a tweet with possible multiple values such as hashtags and URLs. Tweet nodes have properties: id, retweet_count (the number of retweets of this tweet), retweeted (whether this tweet has been retweeted), tweet text, created_at (of type long representing the timestamp---the number of milliseconds since 1/1/1970---in which the tweet was posted), day, month, and year. User nodes represent user accounts and have the following properties: name, screen_name, followers (indicating the number of followers), following (indicating the number of people this user follows), sub_category, category, and location. The sub_category indicates the party to which the user belongs: 'GOP', 'Democrat', 'na', or null. The category values are senate_group, presidential_candidate, reporter, Senator, General, or null. The name property can have an empty string as a value.

State nodes have the name property containing the name of the state. There are 52 State nodes with the state of Florida appears twice as "Florida" and "FL"; furthermore, there is one "na" State node for the user without state information such as presidential candidates. Hashtag nodes have the tag name property. Url nodes have the url property that can also be an empty string.

This model also uses a time tree created to support time range queries (e.g., list tweets posted during some months). The century node has the property century with an integer value of 21 to represent the current century. Each year node has edges only to the month nodes of that year. Each month node has edges only to the day nodes of that month only. In other words, the day nodes are not shared across multiple months. A day node has edges to tweets posted on that particular day of that particular month, year, and century.

Revision: 10/04/2019
Changes from the previous version are highlighted in yellow.

Table 1: Functionalities of the web application

ID	Description
Q1	List *k* most retweeted tweets in a given month and a given year; show the retweet count, the tweet text, the posting user's screen name, the posting user's category, the posting user's sub-category in descending order of the retweet count values **Input:** Value of *k* (e.g., 10), month (e.g., 1), and year (e.g., 2016) **Rationale:** This query finds *k* most influential tweets in a given time frame and the users who posted them.
Q3	Find *k* hashtags that appeared in the most number of states in a given year; list the total number of states the hashtag appeared, the list of the distinct states it appeared (FL is the same as Florida), and the hashtag itself in descending order of the number of states the hashtag appeared. **Input:** Value of *k* and year (e.g., 2016) **Rationale:** This query finds *k* hashtags that are used across the most number of states, which could indicate a certain agenda (e.g., education, healthcare) that is widely discussed. **Hint:** Use group_concat() function to create a list
Q6	Find *k* users who used at least one of the hashtags in a given of hashtags in their tweets. Show the user's screen name and the state the user lives in descending order of the number of this user's followers. **Input:** Value of *k* and list of hashtags (e.g., [GOPDebate, DemDebate]) **Rationale:** This is to find *k* users with similar interests.
Q9	Find top *k* most followed users in a given party. Show the user's screen name, the user's party, and the number of followers in descending order of the number of followers. **Input**: Value of category (e.g., 'GOP', 'Democrat') **Rationale**: This query finds the most influential users measured by the number of followers
Q10	Find the list of distinct hashtags that appeared in one of the states in a given list in a given month of a given year; show the list of the hashtags and the names of the states in which they appeared. **Input:** list of the state, (e.g., [Ohio, Alaska, Alabama]), month, year **Rationale:** This is to find common interests among the users in the states of interest.
Q11	Find *k* tweets (with the given hashtag) posted by republican (GOP) or democrat members of a given state in a given month of a given year. Show the tweet text, the hashtag, the screen name of the posting user, and the users' party **Input**: Values of *k*, state (e.g., Ohio), month, year, hashtag Rationale: This query explores the context in which the hashtag was used
Q15	Find users in a given sub-category along with the list of URLs used in the user's tweets in a given month of a given year. Show the user's screen name, the state the user lives, and the list of URLs **Input:** Values of a sub-category (e.g., 'GOP'), month, year **Rationale:** This query finds URLs shared by a party.
Q18	Find *k* users who were mentioned the most in tweets of users of a given party in a given month of a given year. Show tweet text, user's screen name, user's state, and the name of the user(s) mentioned in the tweet in ascending order of the posting dates of the tweets. Input: Value of k, sub-category (e.g., 'GOP'), month, year
Q23	Find *k* most used hashtags with the count of tweets it appeared posted by a given sub-category of users in a list of months. Show the hashtag name and the count in descending order of the count. **Input:** Values of k, sub-category (e.g., 'GOP'), a list of months (e.g., [1, 2, 3]), year=2016
I	Insert information of a new user into the database. **Input:** All relevant attribute values of a user
D	Delete a given user and all the tweets the user has tweeted, relevant hashtags, and users mentioned **Input:** screen name of the user to be deleted Must check that a user is valid before doing so. If the user's screen name is not valid, abort the transaction.

- The value of *k* is between 1 and 100.

Revision: 10/04/2019
Changes from the previous version are highlighted in yellow.

Tasks and deliverables

1. (20 points) Design an ER diagram to model the database requirement. Use only the ER notations we studied in class.

   ### Submission per group
   - A pdf file of your ER diagram

   ### Checklist to avoid point deduction
   - Use the ER notations studied in class only. Do not submit a hand-written ER diagram.
   - Make sure that the primary key of each entity set is indicated in the diagram.
   - Make sure that no relationship sets have a primary key attribute because each relationship is uniquely identified by the entities participating in the relationship.
   - If there is any candidate key, write the name of the attribute(s) that form the candidate key and the name of the entity set.
   - Make sure that all the given constraints (e.g., key constraint, overlapping constraint, covering constraint) are specified in the ER diagram.
   - Make sure that arrows (if any) are pointed toward the diamond notation for the corresponding relationship set.
   - Do not provide an extended ER diagram that includes types of attributes, indexing, the database engine use, etc. Such a diagram includes physical database design choices in it.

2. (30 points) Design a relational database to store the data from the given csv files. The design needs to minimize unnecessary redundancy and to ensure that all constraints are enforced. Import the data from the csv files into MySQL Server 8.0.

   ### Submission per group
   - (20 points) ProjectDDL.sql that consists of SQL DDL statements that create the schemas and integrity constraints.
   - (6 points) ProjectInsert.sql to insert the data from the given .csv files into MySQL Server. Consult this page for bulk loading of data (https://dev.mysql.com/doc/refman/8.0/en/load-data.html). Implement any trigger required to enforce the constraint that cannot be enforced using any other means.
   - (2 points) CreateDBUsers.sql that creates a user "cs363@%1" with the standard authentication method. This user has the privilege only to view, drop, create, insert, and delete the data in your MySQL database.
   - (2 points) CreateApplicationUsers.sql that creates a relation "appusers" to store a login name of your application user, corresponding encrypted password using SHA, and the privilege ("all" or "readonly") of this user. This script also creates two application users: one with the privilege 'all' and the other with the privilege "readonly".

   ### Checklist to avoid point deduction
   - ProjectDDL.sql and ProjectInsert.sql must run successfully. The content of this file ProjectDDL.sql must not include any physical database design choices such as indexing and or which MySQL engine to use.
   - In ProjectDDL.sql, each table name must be dropped before it is recreated.
   - The design at this stage must aim to reduce unnecessary redundancy as the main objective.

Revision: 10/04/2019
Changes from the previous version are highlighted in yellow.

3. (70 points) Implement all functionalities in Table 1 except I and D in SQL against the database you design and Cypher queries against the given Neo4j database.

Submission per group

- Before.sql that has SQL queries before query optimization.
- PhysicalDesign.sql that implements physical database design choices (e.g., creating indexes) to reduce query response times.
- After.sql that implements optimized SQL queries using the indexes created in PhysicalDesign.sql
- Queries.cipher that has all the Cypher queries against the given Neo4j database. The team does not need to do any optimization.
- PerfSQL.pdf that has Table 2 filled up
- PerfNeo4j.pdf that has Table 3 filled up

Table 2: Average server execution time reported by MySQL server over 5 runs for each query

Buffer pool size: _____ GBytes

Query ID	Optimization method	Before optimization (ms)	After optimization (ms)
Q1			
Q3			
Q6			
Q9			
Q10			
Q11			
Q15			
Q18			
Q23			

Table 3: Average server execution time reported by Neo4j server over 5 runs for each query

Buffer pool size: _____ GBytes

Query ID	Average response time (ms)
Q1	
Q3	
Q6	
Q9	
Q10	
Q11	
Q15	
Q18	
Q23	

Revision: 10/04/2019
Changes from the previous version are highlighted in yellow.

4. (150 points) Write a web-database application that provides all the functionalities listed in Table 1. The application consists of Java Server Page (JSP) pages running on Apache Tomcat 8 server. Some pages issue SQL statements to the backend MySQL database while some other pages issue Cypher queries to the given Neo4j database through JDBC API.

   For backward compatibility to allow the current version of Neo4j to work with the given graph database created by an older version of Neo4j, follow the documentation to install Neo4j and configure its parameters in neo4j.conf.

   One submission per group

   Source code: Project folder with JSP source files. Make sure that your application run on the Google Cloud VM of one of the team members. In the JSP pages, separate styles from JSP code. Comment the code and provide the name of the authors for each page.

   README file: This file documents what you need the teaching staff to know in order to install your code and run your code successfully.

   Grading criteria on the application

   - The application must take advantage of MySQL as much as possible. The application that brings irrelevant data from the database and uses Java code to filter out irrelevant data will lose major points even the output is correct.
   - The application must ensure that each functionality in Table 1 is in its own transaction.
   - The application must prevent against SQL injection caused by invalid input except cross-site scripting attacks.
   - Each JSP page must include the author(s) of the page and must include some comments on the code.
   - The application must use cs363 account to make database connections. The password for the account must not be in the top 25 most popular passwords according to this site.
     - https://www.welivesecurity.com/2018/12/17/most-popular-passwords-2018-revealed/
   - The application must have a login page for authorized users. Application users with "all" privilege can perform every functionality, but application users with the "readonly" privilege can only view query results, but cannot add data (Table 1 functionality I) into the database or delete data (Table 1 functionality D) from the database.
   - The passwords for application users must be encrypted (in SHA) and stored in the appusers relation.

   User interface friendliness is not graded, but it needs to be obvious for application users to perform the required functionality.

5. (30 points) Individual contribution; each team member must be involved in every part of the project.

   One submission per group

Revision: 10/04/2019
Changes from the previous version are highlighted in yellow.

- (10 points) Team work log in text format that provides the meeting time, estimated duration, present team members, tasks assigned; the team will also lose points if starting the project in the last two weeks of the semester.

Example TeamWorklog.txt

Week of Feb. 11, 2019

- Team received the project description and the data
- Team meeting (1 hour) and individual tasks assigned
  - James's task:
    - Design the ER diagram;
    - Check the schemas designed by Liam
  - Liam's task:
    - Proofread the ER diagram
    - Design the relational schemas from the ER diagram

- (10 points) Work logs of individual team members in text format; members who do not submit their individual work log will lose points on their contribution. Name the file as <netid>Worklog.txt. Each log must show continuation of the member effort throughout the project.

Example of individual Worklog.txt

James:

Week of Feb. 11, 2019

- Designed the ER diagram (4 hours). Sent it to Liam for review.

One submission per team member

- (10 points) Private evaluation of the other two members per criteria provided by the instructor