# CPSC 476 - Back-End Engineering - Spring 2018

## Project 4, due May 15

### Introduction

In this project you will add application-level sharding to MiniTwit, partitioning data across three SQLite databases.

### Project Code

If you completed Project 2, you should build on that foundation (though you may choose to run single instances of `minitwit.py` and `mt_api.py` rather than a load-balanced cluster). If necessary, however, you may start from Project 1's codebase or the original MiniTwit code.

### Test Environment

You may use any platform for development, but note that per the Syllabus the test environment for projects in this course is the Ubuntu MATE VM for available from http://michael.shafae.com/#resources.

### Databases

Replace the single SQLite `DATABASE` configuration with three SQLite databases. You will need to modify `get_db()` and `close_database()` to save all three connections in the current application context.

### Shard Key and Mapping

Split your data set into shards based on the user ID in each table (`user.user_id`, `message.author_id`, and `follower.who_id`) so that all of an individual user's data is stored in the same database. Map the shard key modulo 3 to the database.

### Sharded Data Model

Since user ID is used as the shard key, several operations should only require you to work with a single database. For example logging in, posting, following and unfollowing, and retrieving a user's timeline.

Other queries, however, will require you to query multiple databases. For example, retrieving the public timeline or a user's home timeline.

## Primary Keys

Since the SQLite databases are independent, using `integer autoincrement` values as primary keys will lead to duplicate values. One way to solve this problem is to use globally unique identifiers as keys.

While SQLite does not support directly support GUIDs, see the following answer on StackOverflow for an example of configuring the Python `sqlite3` module to use `uuid.UUID` objects as primary keys: Proper way to store GUID in sqlite.

## Database Population

You will need to update your Flask CLI `initdb` command to create the same schema in each shard and your `populatedb` command to insert each data item into the correct shard.

## Submission

Turn in the code for your project by placing `schema.sql`, `population.sql`, `mt_api.py`, your SQL schema files, and any other relevant files in the `project4/` subdirectory of the folder that was shared with you on Dropbox. If you needed to modify any other code, include the updated code and documentation. You may work alone, or make a single submission for a team of 2-3 students. If you work in a team, make only one submission.

To complete your submission, print this sheet, fill out the spaces below, and submit it to the professor in class by the deadline. Failure to follow the instructions exactly will incur a **10%** penalty on the grade for this project for all students on the team.

# Project Submission

**CPSC 476**, **Section 1**

**Project Number _____**

Names of up to three students for this submission

1. _____
2. _____
3. _____

CSUF email of the Dropbox account containing the project files for this submission

_____@csu.fullerton.edu

Comments on your submission

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____