Database SQL Project

Description of the Application:
The database I have chosen to create is that of an online version control system. Examples of this kind of system include GitHub, GitLab, BitBucket, Azure DevOps and Apache Subversion.

The system has a number of users. Each user can own any number of repositories. Each repository contains a branch or multiple branches. Each branch is made of a commit or number of commits, where each commit is made of multiple changes. Users can comment on commits. The following table descriptions will include "PK" for primary key and "FK" for foreign key.

A user is required to have a unique username (PK), a display name (if they wish), an email (for account verification), a description of their account (which may be an empty string), and the date and time they signed up.

Each repo has a unique identifier number (PK), **the name of the user who owns the repo** (FK), the date and time the repo was created, a name that is unique to the user (ie the user cannot have multiple repos with the same name) and a boolean value denoting if the repo is public or private.

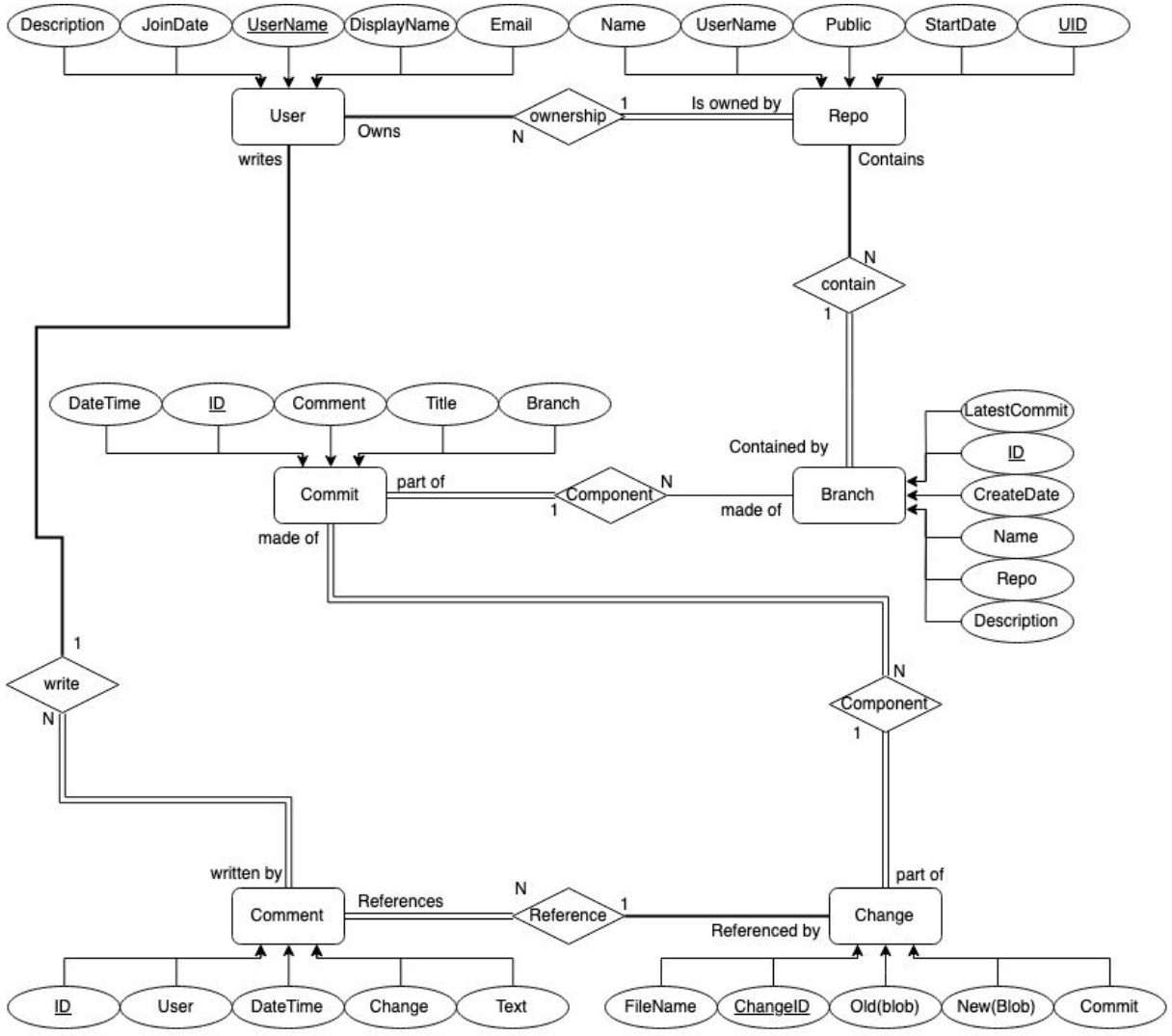
A branch has a unique ID(PK), the date it was created, a name, **an associated repo** (FK), a description of the branch and **the ID of the latest commit** (FK).

Each commit has a unique ID(PK), **the branch it belongs to** (FK), a comment, a title and the date/time it was created.

A change has a unique identifier (PK), the name of the file that was changed, the old file (stored as a blob), the new file (also stored as a blob) and **the commit this change was created on** (FK).

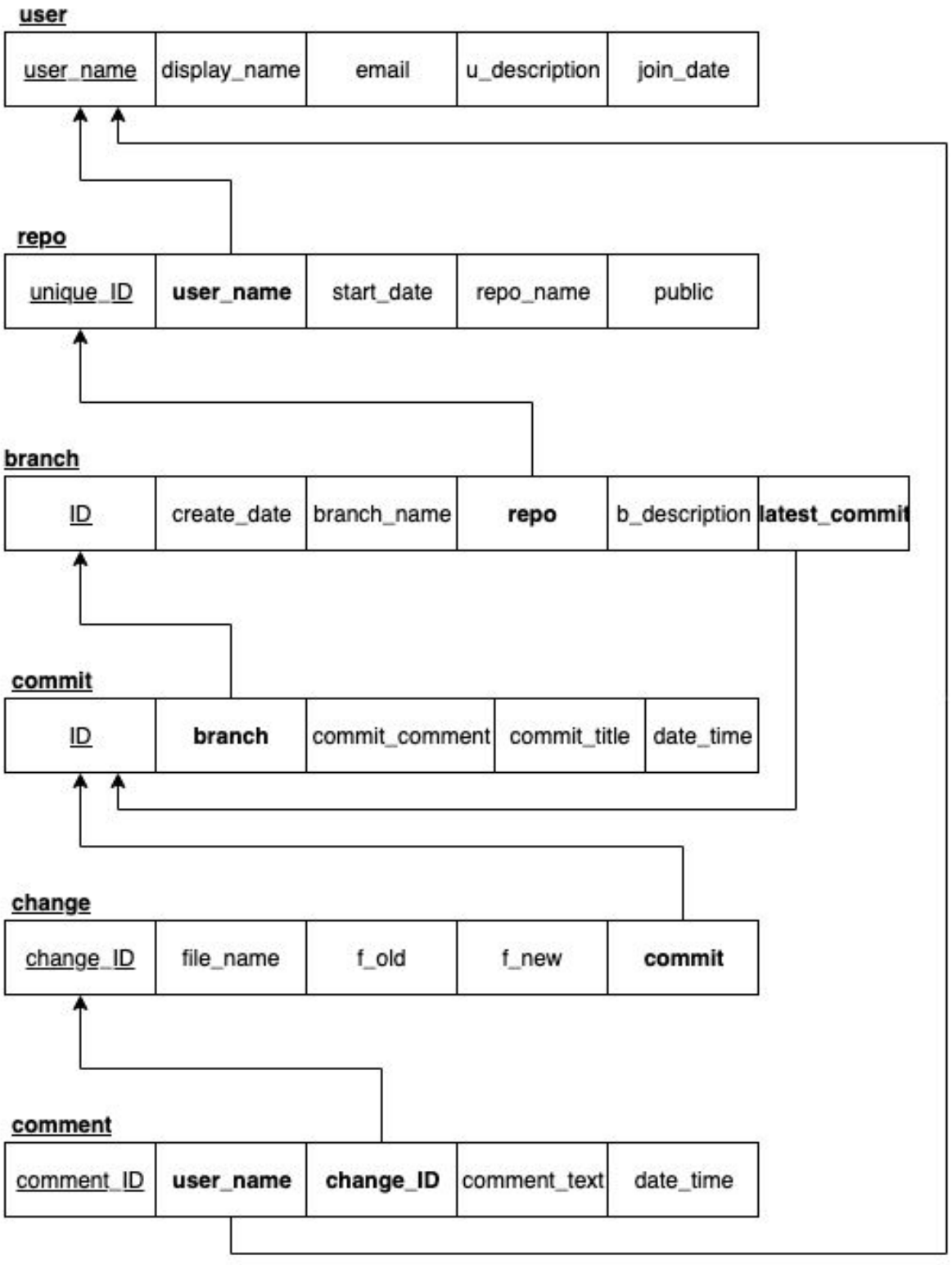
A comment has a unique ID (PK), **the name of the user that made the comment** (FK), some text and **the change that is being referenced** (FK).

Entity Relationship Diagram



Primary Keys are Underlined

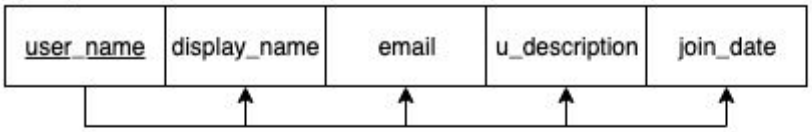
Relational Schema



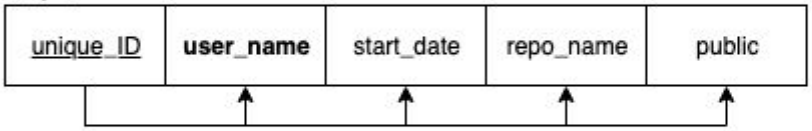
Primary Keys are Underlined
Foreign Keys are **Bold**

Functional Dependency

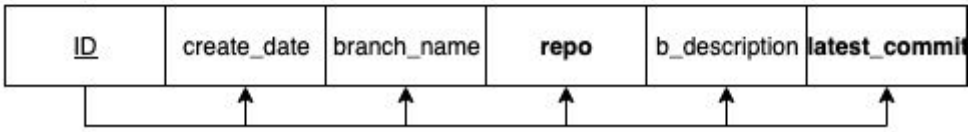
user



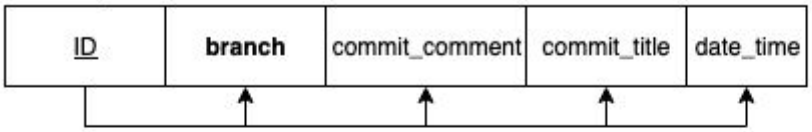
repo



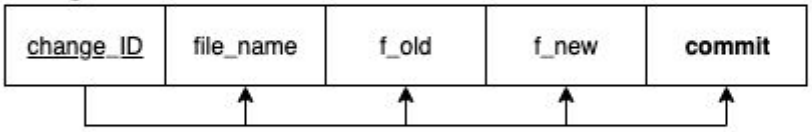
branch



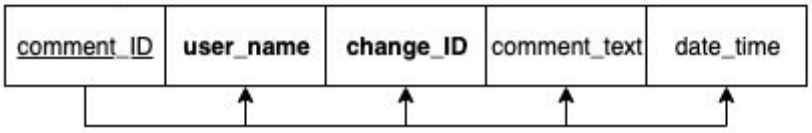
commit



change



comment



Constraints:
The primary type of constraints used in the database are: NOT NULL and UNIQUE, these enforce certain rules on the database that ensure data integrity. For example: the name of a repo cannot be NULL, or: two users cannot exist with the same email. This keeps data integrity and reduces the potential for database abuse.

Auto Increment:
In order to streamline the insertion of values into the database and creation of primary keys I used Auto-Increment for the creation of the keys. This means the database automatically updates the key to be the next value ie n, n+1, n+2... This removes the need for hardcoding keys or requiring applications to manage the creation of keys, resulting in higher reliability overall and better scalability as when adding large amounts of tuples to the table, the database manages the creation of unique keys, not the application developer.

Trigger:
Several triggers are used to automate certain features of the database.
The triggers are:
repo_check, which is triggered before an insert statement in the repo table. This checks if there is already a repo with the same name and same user in the table and if there is: cancels the transaction. It also sends an error message informing the user of the mistake.

branch_check, which is triggered before an insert statement in the branch table. It checks if the repo to which the branch is being added already has a branch of the same name and if it does, cancels the transaction and informs the user of the error.

update_latest_commit, is triggered after an insert statement in the commit table. This updates a branch's latest_commit value to point to the new commit.

curr_time_*, where * is any table that has a date value. This set of triggers is triggered before the insertion of a new tuple into the table and sets the date and time of the tuple to the current system date and time. This is a very useful feature as it simplifies the insertion process for application developers and helps keep data integrity.

View:

The two views created are:

not_logged_in_repos - this is the view that is presented to views who are not currently logged into an account, they can only view a small part of the values of only public repositories, this improves security by limiting the access of users who aren't logged in.

newest_ten_commits

- this is the view of the most recently created 10 commits, this would be useful when trying to gather metadata on commits, and trying to measure trends or activity.

Normalisation:

The database was already normalised on creation so no additional work was required to normalise the database.

Appendix:

```
-- Creating the tables

CREATE TABLE user(
        user_name VARCHAR(25) not null,
        display_name VARCHAR(50),
        email VARCHAR(254) not null unique,
        u_description VARCHAR(500),
        join_date TIMESTAMP not null,
        PRIMARY KEY(user_name)
);

CREATE TABLE repo(
        unique_ID INTEGER not null AUTO_INCREMENT,
        repo_name VARCHAR(100) not null,
        user_name VARCHAR(25) not null,
        start_date TIMESTAMP not null,
        public BOOLEAN not null,
        PRIMARY KEY(unique_ID),
        FOREIGN KEY (user_name) REFERENCES user(user_name)
);

CREATE TABLE bcommit(
        ID INTEGER not null AUTO_INCREMENT,
        branch INTEGER not null,
        commit_comment VARCHAR(500),
        commit_title VARCHAR(100) not null,
        date_time TIMESTAMP not null,
        PRIMARY KEY(ID)
);

CREATE TABLE branch(
        ID INTEGER not null AUTO_INCREMENT,
        create_date TIMESTAMP not null,
        branch_name VARCHAR(50) not null,
        repo INTEGER not null,
        b_description VARCHAR(500),
        latest_commit INTEGER,
        PRIMARY KEY(ID),
        FOREIGN KEY (repo) REFERENCES repo(unique_ID),
        FOREIGN KEY (latest_commit) REFERENCES bcommit(ID)
);
```

```
ALTER TABLE bcommit
ADD FOREIGN KEY (branch) REFERENCES branch(ID);

CREATE TABLE cchange(
        change_ID INTEGER not null AUTO_INCREMENT,
        file_name VARCHAR(100) not null,
        f_old BLOB,
        f_new BLOB not null,
        commit_ID INTEGER not null,
        PRIMARY KEY(change_ID),
        FOREIGN KEY (commit_ID) REFERENCES bcommit(ID)
);

CREATE TABLE ccomment(
        comment_ID INTEGER not null AUTO_INCREMENT,
        user_name VARCHAR(25) not null,
        change_ID INTEGER not null,
        comment_text VARCHAR(1000) not null,
        date_time TIMESTAMP not null,
        PRIMARY KEY(comment_ID),
        FOREIGN KEY (change_ID) REFERENCES cchange(change_ID),
        FOREIGN KEY (user_name) REFERENCES user(user_name)
);

-- Defining triggers

delimiter //
CREATE TRIGGER repo_check BEFORE INSERT
ON repo FOR EACH ROW
BEGIN
   IF EXISTS (SELECT * FROM repo WHERE (repo_name = NEW.repo_name) AND
(user_name = NEW.user_name))
   THEN SIGNAL sqlstate '45001' set message_text = "User cannot have multiple repos
with the same name!";
   END IF;
END;
//
```

```
CREATE TRIGGER branch_check BEFORE INSERT
ON branch FOR EACH ROW
BEGIN
   IF EXISTS (SELECT * FROM branch WHERE (branch_name = NEW.branch_name)
AND (repo = NEW.repo))
   THEN SIGNAL sqlstate '45001' set message_text = "Repo cannot have multiple
branches with the same name!";
   END IF;
END;
//

CREATE TRIGGER update_latest_commit AFTER INSERT
ON bcommit FOR EACH ROW
BEGIN
   UPDATE branch
   SET latest_commit = NEW.ID
   WHERE ID = NEW.branch;
END;
//

CREATE TRIGGER curr_time_user BEFORE INSERT
ON user FOR EACH ROW
BEGIN
   SET NEW.join_date = CURRENT_TIMESTAMP;
END;
//

CREATE TRIGGER curr_time_repo BEFORE INSERT
ON repo FOR EACH ROW
BEGIN
   SET NEW.start_date = CURRENT_TIMESTAMP;
END;
//

CREATE TRIGGER curr_time_branch BEFORE INSERT
ON branch FOR EACH ROW
BEGIN
   SET NEW.create_date = CURRENT_TIMESTAMP;
END;
//
```

```
CREATE TRIGGER curr_time_bcommit BEFORE INSERT
ON bcommit FOR EACH ROW
BEGIN
   SET NEW.date_time = CURRENT_TIMESTAMP;
END;
//

CREATE TRIGGER curr_time_ccomment BEFORE INSERT
ON ccomment FOR EACH ROW
BEGIN
   SET NEW.date_time = CURRENT_TIMESTAMP;
END;
//

delimiter ;

-- Create views

CREATE VIEW not_logged_in_repos(repo_name, user_name, start_date) AS
SELECT repo_name, user_name, start_date FROM repo WHERE (public = true);

CREATE VIEW newest_ten_commits(ID, branch, commit_comment, commit_title,
date_time) AS
SELECT ID, branch, commit_comment, commit_title, date_time FROM bcommit ORDER
BY ID DESC LIMIT 10;

-- Insert data
insert into user values
('darts', 'Senan', 'darts@tcd.com', '3rd Year TCD Computer Science', null),
('randolph', 'Random Dolphin', 'dolphinr@gmail.com', 'Some dude who writes code',
null),
('torvalds', 'Linus', 'torvalds@gmail.com', 'Just a small-time dev.', null),
('xHackerX', 'Aanon', 'a@non.com', 'Elite Member of Anonymous', null),
('kamiKam', 'Kamil Kamil Niamh McNamara', 'mcnamarakamkam@tcd.com', 'Google
engineer', null),
('alig', 'You Know Who', 'laddi@g.com', 'I\'m the bossman', null);

insert into repo values
(null, 'CSU34041', 'darts', null, True),
(null, 'OpenLamp', 'darts', null, True),
(null, 'PersonalProject', 'darts', null, False),
(null, 'Train Sim', 'randolph', null, False),
```

(null, 'GraphJS', 'randolph', null, True),
(null, 'KeyCrypt', 'randolph', null, False),
(null, 'Linux', 'torvalds', null, True),
(null, 'GIT', 'torvalds', null, True),
(null, 'grep', 'torvalds', null, True),
(null, 'l33tHax', 'xHackerX', null, False),
(null, 'ezCodes', 'xHackerX', null, False),
(null, 'LOIC', 'xHackerX', null, True),
(null, 'mongo', 'kamiKam', null, True),
(null, 'graphing-git', 'kamiKam', null, True),
(null, 'grep', 'kamiKam', null, True),
(null, 'hits', 'alig', null, True),
(null, 'myNextHit', 'alig', null, False),
(null, 'upperLower', 'alig', null, False);

insert into branch values
(null, null, 'master', (select unique_ID from repo where user_name = 'darts' and repo_name = 'CSU34041'), '3rd year TCD Information management module', null),
(null, null, 'tmp', (select unique_ID from repo where user_name = 'darts' and repo_name = 'CSU34041'), 'Temporary branch.', null),
(null, null, 'theLamp', (select unique_ID from repo where user_name = 'darts' and repo_name = 'OpenLamp'), 'The title says it all', null),
(null, null, 'oldLamp', (select unique_ID from repo where user_name = 'darts' and repo_name = 'OpenLamp'), 'Built for a previous version of the switch', null),
(null, null, 'project A', (select unique_ID from repo where user_name = 'darts' and repo_name = 'PersonalProject'), 'The first project', null),
(null, null, 'project elrond', (select unique_ID from repo where user_name = 'darts' and repo_name = 'PersonalProject'), 'A slingshot maneouver', null),
(null, null, 'choo choo', (select unique_ID from repo where user_name = 'randolph' and repo_name = 'Train Sim'), 'The actual application', null),
(null, null, 'JS', (select unique_ID from repo where user_name = 'randolph' and repo_name ='GraphJS'), 'A javascript graphing library', null),
(null, null, 'kernel', (select unique_ID from repo where user_name = 'torvalds' and repo_name = 'Linux'), 'An open source kernel', null),
(null, null, 'deadKernel', (select unique_ID from repo where user_name = 'torvalds' and repo_name = 'Linux'), 'A previous version of the kernel', null),
(null, null, 'GIT_SOME', (select unique_ID from repo where user_name = 'torvalds' and repo_name = 'GIT'), 'A version control experiment', null),
(null, null, 'great_returns', (select unique_ID from repo where user_name = 'torvalds' and repo_name = 'grep'), 'The search for life', null),
(null, null, 'masterHax', (select unique_ID from repo where user_name = 'xHackerX' and repo_name = 'l33tHax'), 'Secret stuff', null),

(null, null, 'some code', (select unique_ID from repo where user_name = 'xHackerX' and repo_name = 'ezCodes'), 'Testing new print function', null),
(null, null, 'LOIC base', (select unique_ID from repo where user_name = 'xHackerX' and repo_name = 'LOIC'), null, null),
(null, null, 'mongod', (select unique_ID from repo where user_name = 'kamikam' and repo_name = 'mongo'), null, null),
(null, null, 'mongoloid', (select unique_ID from repo where user_name = 'kamikam' and repo_name = 'mongo'), null, null),
(null, null, 'graph', (select unique_ID from repo where user_name = 'kamikam' and repo_name = 'graphing-git'), null, null),
(null, null, 'getget', (select unique_ID from repo where user_name = 'kamikam' and repo_name = 'grep'), 'All my code', null),
(null, null, 'smack', (select unique_ID from repo where user_name = 'alig' and repo_name = 'hits'), 'smackdowns', null),
(null, null, 'nextWhackyCode', (select unique_ID from repo where user_name = 'alig' and repo_name = 'myNextHit'), 'Some really crazy code, gonna be a hit', null),
(null, null, 'upperLower', (select unique_ID from repo where user_name = 'alig' and repo_name = 'upperLower'), 'turn upper to lower case', null);

insert into bcommit values
(null, 1, 'first commit', 'My First Commit', null),
(null, 1, 'second commit', 'My second Commit', null),
(null, 1, 'some super awesome changes', 'Bug Fixes & features', null),
(null, 2, '', 'init', null),
(null, 2, 'second commit', 'Best Commit', null),
(null, 2, 'Finally fixed all the bugs!', 'Bug Fixes!!!', null),
(null, 3, 'started repo', 'commit numero uno', null),
(null, 3, 'this is getting tough', 'commit numero dos', null),
(null, 4, 'had this great idea', 'laid groundwork', null),
(null, 4, 'more setup', 'this is rough', null),
(null, 5, 'should have started with verion control', 'the whole project', null),
(null, 6, 'started and finished', 'ezclap', null),
(null, 8, '', 'setup', null),
(null, 8, '', 'some changes', null),
(null, 8, '', 'more fixes', null),
(null, 8, 'Fixed replication bug, should have added descriptions to other commits', 'Final commit', null),
(null, 9, 'built the kernel, was easy', 'the whole thing', null),
(null, 11, 'started', 'setup', null),
(null, 11, '', 'fix a thing', null),
(null, 11, '', 'unbreak a thing', null),
(null, 11, 'add stuff', 'added stuff', null),

(null, 11, '', 'donezo', null),
(null, 13, '', 'started', null),
(null, 13, 'code is self-documenting', 'more features', null),
(null, 13, 'developed new algorithm', 'check stackoverflow', null),
(null, 13, 'fixed algorithm', 'I\'m a genius', null),
(null, 13, '', 'fixed it properly this time', null),
(null, 14, '', 'My First Commit', null),
(null, 14, '', 'My second Commit', null),
(null, 15, '', 'best code', null),
(null, 15, '', 'even better code', null),
(null, 15, '', 'I have acheived perfection', null),
(null, 16, '', 'nope', null),
(null, 16, '', 'no', null),
(null, 16, '', 'n', null),
(null, 17, 'this is genius', 'I am genius', null),
(null, 17, 'really hard', 'this is hard', null),
(null, 18, '', 'start and finish', null),
(null, 19, '', 'best project', null),
(null, 21, 'crazy cool', 'cool', null),
(null, 21, 'crazy whack', 'mad', null),
(null, 21, 'wildly wonky', 'out of this world', null),
(null, 22, 'my mona lisa', 'this will be great', null),
(null, 22, 'rome wasn\'t built in a day', 'almost done', null),
(null, 22, '', 'DOOOONNNNNNNNNEEEE!!!!', null);

-- MySQL treats blobs as very large VARCHARs, for demonstration purposes I am using strings
insert into cchange values
(null, 'code.py', null, 'newtext', 1),
(null, 'code.py', 'newtext', 'newertext', 2),
(null, 'code.py', 'newertext', 'pure awesome', 3),
(null, 'script.js', null, 'newtext', 4),
(null, 'superScript.js', null, 'supercode', 4),
(null, 'superScript.js', 'supercode', 'superercode', 5),
(null, 'script.js', 'supercode', 'supermancode', 6),
(null, 'hacks.java', null, 'code', 7),
(null, 'hacks.java', 'code', 'newtext', 8),
(null, 'a.py', null, 'newtext', 9),
(null, 'code.py', 'newtext', 'print(\'athing\')', 10),
(null, 'writey.py', null, 'codes', 11),
(null, 'hacks.py', null, 'execute order 66', 12),
(null, 'code.py', null, 'newtext', 13),

(null, 'cody.py', null, 'l33t works', 14),
(null, 'cody.py', 'l33t works', 'boeing', 15),
(null, 'code.py', 'newtext', 'newtext', 16),
(null, 'app.js', null, 'clg($22)', 17),
(null, 'code.py', null, 'newtext', 18),
(null, 'what.py', null, 'wut', 19),
(null, 'what.py', 'wut', 'yes', 20),
(null, 'code.py', 'newtext', 'better text', 21),
(null, 'code.py', 'better text', 'super text', 22),
(null, 'dave.pl', null, 'my name is dave', 23),
(null, 'dave.pl', 'my name is dave', 'I like to rave', 24),
(null, 'dave.pl', 'I like to rave', 'i also like to pave', 25),
(null, 'dave.pl', 'i also like to pave', 'I love to pave', 26),
(null, 'dave.pl', 'I love to pave', 'Thats what makes me dave', 27),
(null, 'code.py', null, 'newtext', 28),
(null, 'code.py', 'newtext', 'bluetext', 29),
(null, 'hey.s', null, 'yo', 30),
(null, 'hey.s', 'yo', 'dawg', 31),
(null, 'hey.s', 'dawg', 'wassup', 32),
(null, 'code.py', null, 'newtext', 33),
(null, 'code.py', 'newtext', 'create docs', 34),
(null, 'code.py', 'create docs', 'upgrades', 35),
(null, 'code.js', null, 'newtext', 36),
(null, 'l33t_skills.js', null, 'more text', 37),
(null, 'crayCray.c', null, 'wild stuff', 38),
(null, 'supercode.cp', null, 'Insane performance', 39),
(null, 'code.py', null, 'newtext', 40),
(null, 'codes.py', null, 'newtext', 41),
(null, 'codeer.py', null, 'newtext', 42),
(null, 'codey.py', null, 'newtext', 43),
(null, 'codea.py', null, 'code', 44),
(null, 'codera.py', null, 'more code', 45);

insert into ccomment values
(null, 'darts', 1, 'Super cool work, great job me!', null),
(null, 'kamikam', 1, 'Nice', null),
(null, 'randolph', 1, 'I don\'t liek', null),
(null, 'randolph', 2, 'Dislike', null),
(null, 'kamikam', 2, 'I also dislike', null),
(null, 'alig', 5, 'whack', null),
(null, 'alig', 5, 'super whack in fact', null);