

Project Submission 3

Matthew Geary

May 2019

Overview

The problem domain for which I chose to build a database and application is that of a web forum Q&A site, such as those found under the stackexchange umbrella. A forum of this type must allow users to have accounts from which they can post, a system for tracking each users post history, a post's associated tags, the upvotes and downvotes (and overall score) associated with each post, the comments on each post, and any badges, titles, or awards held by users.

The actors which will interact with such an application and database include administrators, moderators, and users. Users should be able to post, comment, vote on posts/comments, and view the post history of other users and themselves. Moderators should be able to perform any actions that users can do, and additionally be able to suspend or ban users, delete posts and comments, and lock post/comment threads. Administrators should be able to perform any actions available to users and moderators, and additionally be able to add or remove user accounts, promote or demote moderators, and modify any other settings necessary for the administration of the application and database. For the purposes of my application, I developed an app with only an administrator account. Support for user roles has not yet been implemented; however, the current application demonstrates functionality for both admins and users with a single hard-coded account.

Possible questions asked by users: How can I make a post? How can I make a comment? How can I view my own post history? How can I view someone else's post history? How can I easily find posts related to a topic or question that I am interested in? How can I view my profile? How can I supply feedback for comments I find helpful/unhelpful? How can I tell whether a user is known as helpful/unhelpful by the community?

Possible questions asked by administrators: How can I fix system issues/glitches with accounts or posts? How can I create or remove accounts? How can I modify application settings? How can I modify data in the database when needed?

Data Source

Stack Exchange performs periodic "data dumps" of their databases related to each of their sites. Each data dump is made up of a number of XML files formatted for use with databases; "Badges", "Comments", "Post History", "Post Links", "Posts", "Tags", "Users", and "Votes". I chose to use the [woodworking.stackexchange.com](https://www.woodworking.stackexchange.com) data dump. The XML files are included in the github repository under `\app\data`.

As part of my project, I created a sub-app (`dbbuilder.py`) which parses these XML files and builds them into a postgresql Database. `dbbuilder.py` can be run against any stackexchange data dump to build a postgres database from the contents.

Application Design

My application utilizes the psycopg2 Python library to connect to a Postgres server. The entire application is written in Python. I successfully tested my application with a Postgres server running on localhost, and with a Postgres server running on a google cloud VM. This is a command-line based application; there is a sql query tool for entering queries directly, as well as menu options which call a number of prebuilt queries. The application uses pipenv, a python virtual environment tool, for dependency management. Although there is only a single dependency for this project (the psycopg2 library), I felt that it would be good practice to use the pipenv tool. The application consists of five files:

main.py

Utilized to run the application. Checks for command line arguments and instantiates connector and browser objects.

connector.py

Contains python class Connector. Objects of this class can be used to connect to a postgres database and perform queries.

browser.py

Contains python class Browser. This class does the heavy lifting for parsing user input and sending queries to the connector.

dbbuilder.py

Contains python class Dbbuilder. This class parses the XML files and builds out the database according to the design seen in the ER Diagram and Relational Schema sections of this document. ddbuilder.py can be used with command line arguments to drop the schema and all its tables, or build out only a selection of tables.

database.ini

This config file holds all credentials needed to connect to a Postgres database. Multiple sections can be placed in the file to allow for different user accounts and/or databases/schemas to be used.

Usage

Requirements:

Python 3.7

pipenv

If python is installed, pipenv can be installed with the command:

```
pip install pipenv
```

1. After cloning the git repo and installing pipenv, navigate to the root directory \postgresApp\.
2. Make a copy of "database.ini.template" and rename the copy to "database.ini".
3. Update the file with the proper credentials to connect to your postgres server. Sections are marked by [section_name]. The default section is postgres. Multiple sections can be created to allow connects with different accounts, or to different databases.
4. Default database is "postgres" and default schema is "CLforum".
5. Type the below command to install dependencies:

```
pipenv sync
```

dbbuilder.py

```
pipenv run app\dbbuilder.py[-d]<database><destroy><all><table1 table2 table3...>
```

1. If "destroy" is included as a command-line argument, the schema specified in database.ini will be dropped with all its tables.
2. If "-d" is included as an argument, the argument following will be interpreted as the section of the database.ini to use for the connection. If this is not specified, it will default to "postgres".
3. Otherwise, specific desired tables can be supplied for building. Options are: badges, commented, comments, decorated, posted, posts, subcomments, subposts, tagged, tags, thread, users.
4. If no command-line arguments are supplied, dbbuilder.py will build all tables. This is the recommended option. Thus, the full database can be build simply with the following command:

```
pipenv run app\dbbuilder.py
```

main.py

```
pipenv run app\main.py<database>
```

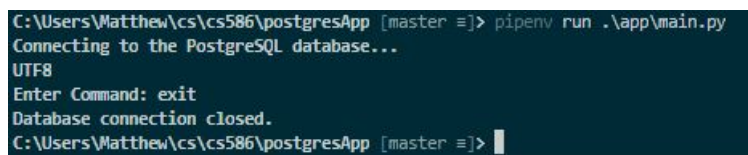
main.py takes a single optional command-line argument. If supplied, this argument will be interpreted as the section of database.ini to use for the connection. If this is not specified, it will default to "postgres". Thus, the app can be run simply with the following command:

```
pipenv run app\main.py
```

Commands

exit

Used at any time to exit the program and disconnect from the database.



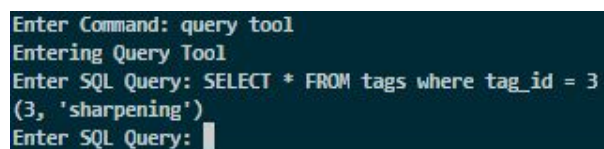
```
C:\Users\Matthew\cs\cs586\postgresApp [master =>] pipenv run .\app\main.py
Connecting to the PostgreSQL database...
UTF8
Enter Command: exit
Database connection closed.
C:\Users\Matthew\cs\cs586\postgresApp [master =>]
```

Figure 1: exit

query tool

sqlrunner

Entering one of these aliases activates a mode for entering sql queries directly from the command line. The user can type **back** to return to the application command line.



```
Enter Command: query tool
Entering Query Tool
Enter SQL Query: SELECT * FROM tags where tag_id = 3
(3, 'sharpening')
Enter SQL Query:
```

Figure 2: query tool

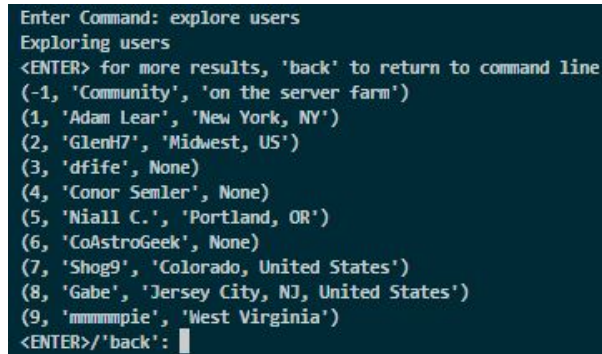
explore users

Returns lists of users in groups of ten. The user can press <ENTER> to see more results, or type **back** to return to the application command line.

```

self._exploreuserssql = (
    "SELECT user_id, user_name, location "
    "FROM Users "
    "OFFSET (%s) LIMIT (%s)"
)

```



```

Enter Command: explore users
Exploring users
<ENTER> for more results, 'back' to return to command line
(-1, 'Community', 'on the server farm')
(1, 'Adam Lear', 'New York, NY')
(2, 'GlenH7', 'Midwest, US')
(3, 'dfife', None)
(4, 'Conor Semler', None)
(5, 'Niall C.', 'Portland, OR')
(6, 'CoAstroGeek', None)
(7, 'Shog9', 'Colorado, United States')
(8, 'Gabe', 'Jersey City, NJ, United States')
(9, 'mmmmmpie', 'West Virginia')
<ENTER>/'back': 

```

Figure 3: explore users

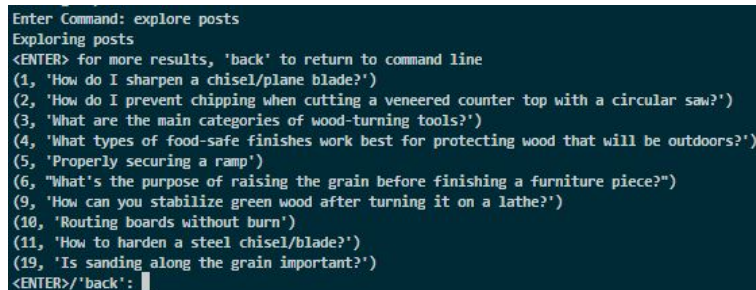
explore posts

Returns lists of posts in groups of ten. The user can press <ENTER >to see more results, or type **back** to return to the application command line.

```

self._explorepostsql = (
    "SELECT post_id, title "
    "FROM Posts "
    "WHERE title != 'None' "
    "OFFSET (%s) LIMIT (%s)"
)

```



```

Enter Command: explore posts
Exploring posts
<ENTER> for more results, 'back' to return to command line
(1, 'How do I sharpen a chisel/plane blade?')
(2, 'How do I prevent chipping when cutting a veneered counter top with a circular saw?')
(3, 'What are the main categories of wood-turning tools?')
(4, 'What types of food-safe finishes work best for protecting wood that will be outdoors?')
(5, 'Properly securing a ramp')
(6, 'What's the purpose of raising the grain before finishing a furniture piece?')
(9, 'How can you stabilize green wood after turning it on a lathe?')
(10, 'Routing boards without burn')
(11, 'How to harden a steel chisel/blade?')
(19, 'Is sanding along the grain important?')
<ENTER>/'back': 

```

Figure 4: explore posts

explore tags

Returns lists of tags in groups of ten. The user can press <ENTER >to see more results, or type **back** to return to the application command line.

```

self._exploretagsql = (
    "SELECT tag_id, tag_name "
    "FROM Tags "
    "OFFSET (%s) LIMIT (%s)"
)

```

```

Enter Command: explore tags
Exploring tags
<ENTER> for more results, 'back' to return to command line
(2, 'plane')
(3, 'sharpening')
(4, 'technique')
(5, 'circular-saw')
(6, 'veneer')
(8, 'tools')
(9, 'finishing')
(10, 'plywood')
(11, 'toy')
(12, 'finish')
<ENTER>/'back': 

```

Figure 5: explore tags

view user <user_id>

Allows the user a detailed view of the profile for the selected user_id, including user badges.

```

self._viewusersql = (
    "SELECT users.user_id, user_name, location,"
    "reputation, creation_date, last_active_date "
    "FROM Users "
    "WHERE users.user_id = (%s)"
)
self._userbadgessql = (
    "SELECT badge_name "
    "FROM Users, Decorated, Badges "
    "WHERE users.user_id = (%s) "
    "AND decorated.user_id=(%s) "
    "AND badges.badge_id=decorated.badge_id"
)

```

```

Enter Command: view user
Please define both a context and an id to view
Enter Command: view user 1
Viewing user with ID 1
Id:          1
Name:        Adam Lear
Location:    New York, NY
Badges:      ['Autobiographer']
Reputation:  101
Joined:      2015-03-17 14:49:42.463000
Last Active: 2016-06-10 22:48:57.133000
Enter Command: 

```

Figure 6: view user

view post <post_id>

Allows the user a detailed view of the post, subposts, and comments for the selected post_id.

```

self._viewpostsql = (
    "SELECT creation_date, last_edit_date,"
    "favorite_count, view_count, score, title, post_id, body "
    "FROM posts "
    "WHERE post_id = (%s)"
)

```

```

self._viewpostersql = (
    "SELECT users.user_name "
    "FROM users, posted "
    "WHERE posted.post_id = (%s) "
    "AND users.user_id=posted.user_id"
)
self._viewsubpostssql = (
    "SELECT creation_date, last_edit_date, "
    "favorite_count, view_count, score, title, posts.post_id, body "
    "FROM posts, subposts "
    "WHERE subposts.parent_id = (%s) "
    "AND Posts.post_id = Subposts.child_id"
)
self._findparentsql = (
    "SELECT subposts.parent_id "
    "FROM Subposts "
    "WHERE subposts.child_id = (%s)"
)
self._viewcommentssql = (
    "SELECT thread.post_id, comments.comment_id,"
    "comments.score, comments.creation_date, comments.text "
    "FROM Comments, Thread, Posts "
    "WHERE posts.post_id = (%s) "
    "AND posts.post_id = thread.post_id "
    "AND thread.comment_id = comments.comment_id"
)
self._viewcommentersql = (
    "SELECT users.user_name "
    "FROM Commented, Users "
    "WHERE commented.comment_id = (%s) "
    "AND commented.user_id = users.user_id"
)

```

```

Enter Command: view post 1
Viewing post with ID 1
-----
Title: How do I sharpen a chisel/plane blade?
I'm getting started with woodworking and am looking to refine my technique for
sharpening chisel/plane blades. Currently, I just have a cheap double-sided
water stone, but am now looking to get a "proper" setup which will hold me
moving forward and then also refining my technique. Any suggestions? Edit:
Specifically, what type of water stones and diamond stone do I need, should I
get a honing guide, what else is must have vs a nice to have, etc. Then what
process do I go through to "correctly" use these tools.
By: anthonyv ID: 1 Score: 21 Views: 1538 Favorites: 4
Posted: 2015-03-17 15:21:33.900000 Last Edited: 2015-12-02 22:25:12.907000
-----
This is basically the technique I use.
http://www.thewoodwhisperer.com/videos/my-sharpening-
system/?as=sharpening&mode=posts&ap=1
By: Web ID: 2 Score: 2
Posted: 2015-03-17 15:23:35.850000
-----
Welcome to your new SE community! Can you give a little more info about how
you'd like your technique to be refined? Basically, what are you trying to do
better? Being a bit more specific will help everyone give you better answers.
:)
By: Ana ID: 5 Score: 3
Posted: 2015-03-17 15:25:59.820000
-----
@Web Sending users elsewhere to find that information isn't really what this
site is about (see http://woodworking.stackexchange.com/tour). Also, please
post your answers below. Comments do not have the same features as answers to
help vet that content. Thanks.
By: Robert Cartaino ID: 69 Score: 2
Posted: 2015-03-17 17:17:44.327000
-----
You mention that you have a sharpening stone. One improvement might also be
to make an angled guard that you run atop the stone that helps to impart the
proper chisel angle. Something like the below might help to ensure you get a
razor-sharp edge at the proper angle.
By: Peter Grace ID: 14 Score: 7 Views: None Favorites: None
Posted: 2015-03-17 15:31:47.130000 Last Edited: 2015-03-17 15:31:47.130000
-----
I have that exact honing guide, and I'm not terribly impressed by it. It only
has one roller wheel in the center, and has the tendency to wobble if you're

```

Figure 7: view post

view tag <tag.id>

Returns a list of the posts affiliated with a selected tag.id.

```

self._viewtagpostssql = (
    "SELECT tags.tag_name, posts.post_id, posts.title "
    "FROM Tags, Posts, Tagged "
    "WHERE tags.tag_id = (%s) "
    "AND tags.tag_id = tagged.tag_id "
    "AND tagged.post_id = posts.post_id OFFSET (%s) LIMIT (%s)"
)

```

```

Enter Command: view tag 3
Viewing tag with ID 3
('sharpening', 1, 'How do I sharpen a chisel/plane blade?')
('sharpening', 11, 'How to harden a steel chisel/blade?')
('sharpening', 20, 'What angles should I be honing to for what type of chisels/planes/uses?')
('sharpening', 60, 'High grit sandpaper to sharpen a v-gouge')
('sharpening', 73, 'How can I adjust my hand plane to take an even cut across its width?')
('sharpening', 124, 'How to hand-sharpen a badly worn gouge?')
('sharpening', 320, 'How to sharpen a circular saw or mitre saw blade')
End of results
Enter Command:

```

Figure 8: view tag

new post

Allows the user to create a new post. This demonstrates a stored procedure. I would still like to port more of these sql queries over to stored procedures, but for now this serves in that role. Known bug: if the user enters a tag name not in the database, the application will throw an error and exit.

```
self._newpostidsql = (
    "SELECT max(post_id) "
    "FROM Posts"
)
self._newpostsql = (
    "CALL newpost(%s, %s, %s, %s, %s, %s, %s, %s)"
)
self._newpostprocsql = (
    "CREATE OR REPLACE PROCEDURE newpost("
        "post_id int, creation_date timestamp, last_edit_date timestamp,"
        "favorite_count int, view_count int, score int,"
        "title varchar(100), body varchar(5000))"
    "AS $$"
    "BEGIN "
    "INSERT INTO Posts ("
        "post_id, creation_date, last_edit_date,"
        "favorite_count, view_count, score,"
        "title, body)"
    "VALUES ("
        "post_id, creation_date, last_edit_date,"
        "favorite_count, view_count, score,"
        "title, body);"
    "END;"
    "$$ LANGUAGE plpgsql;"
)
self._newpostedsql = (
    "INSERT INTO Posted "
    "(user_id, post_id) "
    "VALUES (%s, %s)"
)
self._posttagsql = (
    "INSERT INTO Tagged "
    "(tag_id, post_id) "
    "VALUES (%s, %s)"
)
```



```

Enter Command: new post
Enter Post Title: TEST POST
Enter Post Body: TEST POST
Enter Post Tags as <Tag1,Tag2,Tag3...>: tools,sharpening
Created new post with ID 401
Enter Command: view post 401
Viewing post with ID 401
-----
Title: TEST POST
TEST POST
By: Admin ID: 401 Score: 0 Views: 0 Favorites: 0
Posted: 2019-06-07 23:27:29.303931 Last Edited: 2019-06-07 23:27:29.303931
-----
Enter Command:

```

Figure 9: new post

new post <post_id>

Allows the user to create a subpost on post_id. Uses many of the same queries as **new post**, but also inserts the post into Subposts.

```

self._newsbpostsql = (
    "INSERT INTO Subposts "
    "(parent_id, child_id) "
    "VALUES (%s, %s)"
)

```

```

Enter Command: new post 401
Enter Post Body: TEST SUBPOST
Created new post with ID 402 on Parent 401
Enter Command: view post 401
Viewing post with ID 401
-----
Title: TEST POST
TEST POST
By: Admin ID: 401 Score: 0 Views: 0 Favorites: 0
Posted: 2019-06-07 23:27:29.303931 Last Edited: 2019-06-07 23:27:29.303931
-----
TEST SUBPOST
By: Admin ID: 402 Score: 0 Views: 0 Favorites: 0
Posted: 2019-06-07 23:27:53.464854 Last Edited: 2019-06-07 23:27:53.464854
-----
Enter Command:

```

Figure 10: new subpost

new comment <post_id>

Allows the user to create a comment on post_id.

```

self._newcommentsql = (
    "INSERT INTO Comments "
    "(comment_id, score, creation_date, text) "
    "VALUES (%s, %s, %s, %s)"
)
self._newcommentidsql = (
    "SELECT max(comment_id) "
    "FROM Comments"
)
self._newcommentedsq = (
    "INSERT INTO Commented "
    "(user_id, comment_id) "

```

```

        "VALUES (%s, %s)"
    )
    self._newthreadsql = (
        "INSERT INTO Thread "
        "(post_id, comment_id) "
        "VALUES (%s, %s)"
    )

```

```

Enter Command: new comment 402
402
Enter Post Body: TEST COMMENT
Created new Comment with ID 401 on Parent 402
Enter Command: view post 401
Viewing post with ID 401
-----
Title: TEST POST
TEST POST
By: Admin ID: 401 Score: 0 Views: 0 Favorites: 0
Posted: 2019-06-07 23:27:29.303931 Last Edited: 2019-06-07 23:27:29.303931
-----
TEST SUBPOST
By: Admin ID: 402 Score: 0 Views: 0 Favorites: 0
Posted: 2019-06-07 23:27:53.464854 Last Edited: 2019-06-07 23:27:53.464854
-----
TEST COMMENT
By: Admin ID: 401 Score: 0
Posted: 2019-06-07 23:28:21.144465
-----
Enter Command:

```

Figure 11: new comment

delete post <post_id>

Allows the user to delete a post with post_id.

```

self._deletefromtaggedsql = (
    "DELETE FROM Tagged "
    "WHERE Tagged.post_id = (%s)"
)
self._deletefrompostsssql = (
    "DELETE FROM Posts "
    "WHERE Posts.post_id=(%s)"
)
self._deletefromsubpostsql = (
    "DELETE FROM Subposts "
    "WHERE parent_id = (%s) "
    "OR child_id = (%s)"
)
self._deletefrompostedsql = (
    "DELETE FROM Posted "
    "WHERE user_id = (%s) "
    "OR post_id = (%s)"
)

```

```

Enter Command: delete post 401
Deleted post with ID 401
Enter Command: view post 401
Viewing post with ID 401
No results
Can't view post
Enter Command: █

```

Figure 12: delete post

delete comment <comment_id>

Allows the user to delete a comment with comment_id.

```

self._deletefromcommentedssql = (
    "DELETE FROM Commented "
    "WHERE comment_id = (%s)"
)
self._deletefromcommentssql = (
    "DELETE FROM Comments "
    "WHERE comment_id = (%s)"
)
self._deletefromthreadssql = (
    "DELETE FROM Thread "
    "WHERE comment_id = (%s) "
    "OR post_id = (%s)"
)

```

```

Enter Command: delete comment 401
Deleted comment with ID 401
Enter Command: view comment 401
Viewing comment with ID 401
Can't view comment
Enter Command: view post 402
Viewing post with ID 402

-----
Title: Subpost of Post 401
TEST SUBPOST
By: Admin ID: 402 Score: 0 Views: 0 Favorites: 0
Posted: 2019-06-07 23:27:53.464854 Last Edited: 2019-06-07 23:27:53.464854
-----
Enter Command: █

```

Figure 13: delete comment

Table Creation Statements

```

self._createtagssql = (
    "CREATE TABLE Tags ("
    "tag_id int PRIMARY KEY,"
    "tag_name varchar({0}))".format(lims[0])
)
self._createuserssql = (
    "CREATE TABLE Users ("
    "user_id int PRIMARY KEY,"
    "user_name varchar({0}),"
)

```

```

        "location varchar({0}),"
        "reputation int,"
        "creation_date timestamp,"
        "last_active_date timestamp,"
        "about varchar({1}))".format(lims[0], lims[3])
    )
self._createpostssql = (
    "CREATE TABLE Posts ("
    "post_id int PRIMARY KEY,"
    "creation_date timestamp,"
    "last_edit_date timestamp,"
    "favorite_count int,"
    "view_count int,"
    "score int,"
    "title varchar({0}),"
    "body varchar({1}))".format(lims[0], lims[3])
)
self._createcommentssql = (
    "CREATE TABLE Comments ("
    "comment_id int PRIMARY KEY,"
    "score int,"
    "creation_date timestamp,"
    "text varchar({0}))".format(lims[3])
)
self._createbadgessql = (
    "CREATE TABLE Badges ("
    "badge_id int PRIMARY KEY,"
    "badge_name varchar({0}))".format(lims[0])
)
# relationship table creation statements
self._createpostedsql = (
    "CREATE TABLE Posted ("
    "user_id int REFERENCES Users(user_id),"
    "post_id int REFERENCES Posts(post_id))"
)
self._createtaggedsql = (
    "CREATE TABLE Tagged ("
    "tag_id int REFERENCES Tags(tag_id),"
    "post_id int REFERENCES Posts(post_id))"
)
self._createcommentedsql = (
    "CREATE TABLE Commented ("
    "user_id int REFERENCES Users(user_id),"
    "comment_id int REFERENCES Comments(comment_id))"
)
self._createthreadsql = (
    "CREATE TABLE Thread ("
    "post_id int REFERENCES Posts(post_id),"
    "comment_id int REFERENCES Comments(comment_id))"
)
self._createdecoratedsql = (
    "CREATE TABLE Decorated ("
    "user_id int REFERENCES Users(user_id),"

```

```

        "badge_id int REFERENCES Badges(badge_id),"
        "date_awarded timestamp)"
    )
self._createsubpostssql = (
    "CREATE TABLE Subposts ("
    "parent_id int REFERENCES Posts(post_id),"
    "child_id int REFERENCES Posts(post_id))"
    )
self._createsubcommentssql = (
    "CREATE TABLE Subcomments ("
    "parent_id int REFERENCES Comments(comment_id),"
    "child_id int REFERENCES Comments(comment_id))"
    )

```

ER Diagram

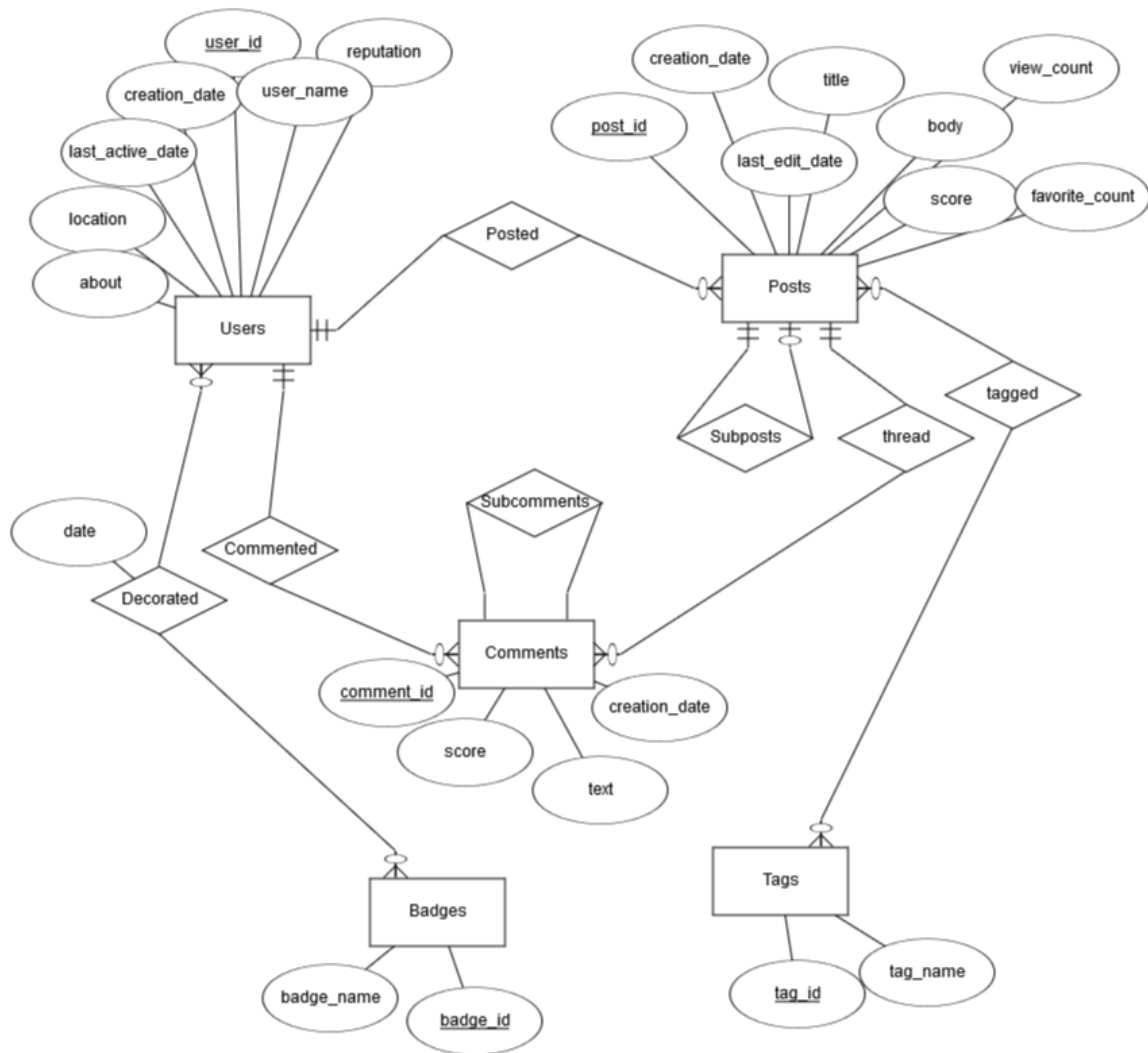


Figure 14: ER Diagram for Forum Database

Relational Schema

Users(user_id, user_name, location, reputation, creation_date, last_active_date, about)

Posts(post_id, creation_date, last_edit_date, favorite_count, view_count, score, title, body)

Posted(user_id, post_id)

user_id IS A FOREIGN KEY TO Users

post_id IS A FOREIGN KEY TO Posts

Tags(tag_id, tag_name)

Tagged(post_id, tag_id)

post_id IS A FOREIGN KEY TO Posts

tag_id IS A FOREIGN KEY TO Tags

Comments(comment_id, text, score, creation_date)

Commented(user_id, comment_id)

user_id IS A FOREIGN KEY TO Users

comment_id IS A FOREIGN KEY TO Comments

Thread(post_id, comment_id)

post_id IS A FOREIGN KEY TO Posts

comment_id IS A FOREIGN KEY TO Comments

Badges(badge_id, badge_name)

Decorated(user_id, badge_id, date)

user_id IS A FOREIGN KEY TO Users

badge_id IS A FOREIGN KEY TO Badges

Subposts(parent_id, child_id)

parent_id and child_id ARE FOREIGN KEYS to Posts.post_id

Subcomments(parent_id, child_id)

parent_id and child_id ARE FOREIGN KEYS to Comments.comment_id