Udiddit, a social news aggregator

Introduction

Udiddit, a social news aggregation, web content rating, and discussion website, is currently using a risky and unreliable Postgres database schema to store the forum posts, discussions, and votes made by their users about different topics.

The schema allows posts to be created by registered users on certain topics, and can include a URL or a text content. It also allows registered users to cast an upvote (like) or downvote (dislike) for any forum post that has been created. In addition to this, the schema also allows registered users to add comments on posts.

Here is the DDL used to create the schema:

```sql
CREATE TABLE bad_posts (
    id SERIAL PRIMARY KEY,
    topic VARCHAR(50),
    username VARCHAR(50),
    title VARCHAR(150),
    url VARCHAR(4000) DEFAULT NULL,
    text_content TEXT DEFAULT NULL,
    upvotes TEXT,
    downvotes TEXT
);

CREATE TABLE bad_comments (
    id SERIAL PRIMARY KEY,
    username VARCHAR(50),
    post_id BIGINT,
    text_content TEXT
);
```

Part I: Investigate the existing schema

As a first step, investigate this schema and some of the sample data in the project's SQL workspace. Then, in your own words, outline three (3) specific things that could be improved about this schema. Don't hesitate to outline more if you want to stand out!

- Bad_posts table needs a foreign key link to bad_comments (post_id) to enable link from comment to associated pos, this will also allow for comments to be deleted upon the deletion of a post through a foreign key cascade.

- In addition, 'username' in bad_posts should be placed in a separate table and upvotes/downvotes should be set as different data type (numerical/ Boolean), this would enable for votes to be associated with usernames and reflected on a post, furthermore if a user is deleted, the vote would also be through the aforementioned foreign key cascade.

- Creating a view of the post table with the joins of foreign keys for upvotes, downvotes and comment counts to make future queries easier to develop.

Part II: Create the DDL for your new schema

Having done this initial investigation and assessment, your next goal is to dive deep into the heart of the problem and create a new schema for Udiddit. Your new schema should at least reflect fixes to the shortcomings you pointed to in the previous exercise. To help you create the new schema, a few guidelines are provided to you:

1. Guideline #1: here is a list of features and specifications that Udiddit needs in order to support its website and administrative interface:
   a. Allow new users to register:
      i. Each username has to be unique
      ii. Usernames can be composed of at most 25 characters
      iii. Usernames can't be empty
      iv. We won't worry about user passwords for this project
   b. Allow registered users to create new topics:
      i. Topic names have to be unique.
      ii. The topic's name is at most 30 characters
      iii. The topic's name can't be empty
      iv. Topics can have an optional description of at most 500 characters.
   c. Allow registered users to create new posts on existing topics:
      i. Posts have a required title of at most 100 characters
      ii. The title of a post can't be empty.
      iii. Posts should contain either a URL or a text content, **but not both**.
      iv. If a topic gets deleted, all the posts associated with it should be automatically deleted too.
      v. If the user who created the post gets deleted, then the post will remain, but it will become dissociated from that user.
   d. Allow registered users to comment on existing posts:
      i. A comment's text content can't be empty.
      ii. Contrary to the current linear comments, the new structure should allow comment threads at arbitrary levels.
      iii. If a post gets deleted, all comments associated with it should be automatically deleted too.
      iv. If the user who created the comment gets deleted, then the comment will remain, but it will become dissociated from that user.

<blockquote>

       v.   If a comment gets deleted, then all its descendants in the thread structure should be automatically deleted too.

    e. Make sure that a given user can only vote once on a given post:

        i.   Hint: you can store the (up/down) value of the vote as the values 1 and -1 respectively.

       ii.   If the user who cast a vote gets deleted, then all their votes will remain, but will become dissociated from the user.

     iii.   If a post gets deleted, then all the votes for that post should be automatically deleted too.

</blockquote>

2. Guideline #2: here is a list of queries that Udiddit needs in order to support its website and administrative interface. Note that you don't need to produce the DQL for those queries: they are only provided to guide the design of your new database schema.

    a. List all users who haven't logged in in the last year.
   Example query –

    b. List all users who haven't created any post.

    c. Find a user by their username.

    d. List all topics that don't have any posts.

    e. Find a topic by its name.

    f. List the latest 20 posts for a given topic.

    g. List the latest 20 posts made by a given user.

    h. Find all posts that link to a specific URL, for moderation purposes.

    i. List all the top-level comments (those that don't have a parent comment) for a given post.

    j. List all the direct children of a parent comment.

    k. List the latest 20 comments made by a given user.

    l. Compute the score of a post, defined as the difference between the number of upvotes and the number of downvotes

3. Guideline #3: you'll need to use normalization, various constraints, as well as indexes in your new database schema. You should use named constraints and indexes to make your schema cleaner.

4. Guideline #4: your new database schema will be composed of five (5) tables that should have an auto-incrementing id as their primary key.

Once you've taken the time to think about your new schema, write the DDL for it in the space provided here:

```sql
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    username VARCHAR(25) NOT NULL UNIQUE,
    createdtime timestamp NOT NULL DEFAULT NOW(),
    lastloggedin timestamp NOT NULL DEFAULT NOW(),
    CONSTRAINT username_not_empty CHECK(LENGTH(TRIM(username)) > 0)
);


create index latest_created_users ON users (id, createdtime );


CREATE TABLE topic (
    id SERIAL PRIMARY KEY,
    topics VARCHAR(30) NOT NULL UNIQUE,
    topic_description VARCHAR(500),
    created timestamp NOT NULL DEFAULT NOW(),
    updated timestamp NOT NULL DEFAULT NOW(),
    CONSTRAINT topic_name_not_empty CHECK(LENGTH(TRIM(topics)) > 0)
);
CREATE INDEX latest_amended_topic ON topic (topics, updated);
CREATE INDEX latest_topic ON topic (topics, created);


CREATE TABLE posts (
    id SERIAL PRIMARY KEY,
    topic_link INT,
    FOREIGN KEY (topic_link) REFERENCES topic (id) ON DELETE SET NULL,
    title VARCHAR(100) NOT NULL,
    url VARCHAR(4000) DEFAULT '',
    text_content TEXT DEFAULT '',
    user_id INT,
    FOREIGN KEY(user_id) REFERENCES users (id) ON DELETE SET NULL,
    CONSTRAINT onlyUrlOrText CHECK ((NULLIF (url, '') IS NULL OR NULLIF (text_content, '') is NULL) AND
NOT (NULLIF(url, '') IS NULL AND NULLIF(text_content, '' ) IS NULL)),
```

```sql
    created timestamp NOT NULL DEFAULT NOW(),
    updated timestamp NOT NULL DEFAULT NOW(),
    CONSTRAINT title_name_not_empty CHECK(LENGTH(TRIM(title )) > 0)


);


CREATE INDEX latest_post_for_topic ON posts (topic_link, created);
CREATE INDEX latest_posts_for_user ON posts (user_id, created);
CREATE INDEX last_contributed ON posts (user_id, updated);


CREATE TABLE comments (
    id SERIAL PRIMARY KEY,
    assoc_post INT NOT NULL,
    FOREIGN KEY(assoc_post) REFERENCES posts (id) ON DELETE CASCADE,
    assoc_comment INT,
    FOREIGN KEY (assoc_comment) REFERENCES comments (id) ON DELETE CASCADE,
    assoc_user INT,
    FOREIGN KEY (assoc_user) REFERENCES users (id) ON DELETE SET NULL,
    comment_text TEXT NOT NULL,
    created timestamp NOT NULL DEFAULT NOW(),
    updated timestamp NOT NULL DEFAULT NOW(),
    CONSTRAINT comment_text_not_empty CHECK(LENGTH(TRIM(comment_text)) > 0)

);


CREATE INDEX latest_comment_reply ON comments(assoc_comment, created);
CREATE INDEX latest_comment_for_user ON comments (assoc_user, created);
CREATE INDEX last_updated_comment_by_user ON comments (assoc_user, updated);


CREATE TABLE vote (
    id SERIAL PRIMARY KEY,
    up_down_vote INT,
    voted_post INT NOT NULL,
    FOREIGN KEY (voted_post) REFERENCES posts (id) ON DELETE CASCADE,
```

```sql
    user_vote INT,
    FOREIGN KEY (user_vote) REFERENCES users (id) ON DELETE SET NULL,
    created timestamp NOT NULL DEFAULT NOW(),
    updated timestamp NOT NULL DEFAULT NOW(),
    CONSTRAINT one_vote_per_post UNIQUE(user_vote, voted_post)
);


CREATE INDEX latest_post_vote ON vote (voted_post, created);
CREATE INDEX vote_number_by_post ON vote (voted_post, id);
```

Part III: Migrate the provided data

Now that your new schema is created, it's time to migrate the data from the provided schema in the project's SQL Workspace to your own schema. This will allow you to review some DML and DQL concepts, as you'll be using INSERT…SELECT queries to do so. Here are a few guidelines to help you in this process:

1. Topic descriptions can all be empty
2. Since the bad_comments table doesn't have the threading feature, you can migrate all comments as top-level comments, i.e. without a parent
3. You can use the Postgres string function **regexp_split_to_table** to unwind the comma-separated votes values into separate rows
4. Don't forget that some users only vote or comment, and haven't created any posts. You'll have to create those users too.
5. The order of your migrations matter! For example, since posts depend on users and topics, you'll have to migrate the latter first.
6. Tip: You can start by running only SELECTs to fine-tune your queries, and use a LIMIT to avoid large data sets. Once you know you have the correct query, you can then run your full INSERT…SELECT query.
7. **NOTE**: The data in your SQL Workspace contains thousands of posts and comments. The DML queries may take at least 10-15 seconds to run.

Write the DML to migrate the current data in bad_posts and bad_comments to your new database schema:

```
USERS

----------------------------------------

INSERT INTO users (username)

    SELECT DISTINCT username

    FROM bad_posts UNION SELECT DISTINCT username FROM bad_comments UNION SELECT DISTINCT

username FROM (SELECT DISTINCT regexp_split_to_table(downvotes, ',') AS username FROM bad_posts

UNION SELECT DISTINCT regexp_split_to_table(upvotes, ',') FROM bad_posts) bp1;


----------------------------------------

TOPIC
```

```sql
------------------------------------
INSERT INTO topic (topics)

    SELECT DISTINCT topic

    AS topics

    FROM bad_posts;
------------------------------------

POSTS

------------------------------------

INSERT INTO posts (topic_link, user_id, title, url, text_content)

    SELECT t.id, u.id,

        LEFT(title,100), url, bp.text_content

    FROM bad_posts bp

    JOIN topic t

    ON bp.topic = t.topics

    JOIN users u

    ON bp.username = u.username;



------------------------------------

VOTE

------------------------------------


INSERT INTO vote (up_down_vote, voted_post, user_vote)

SELECT -1 AS up_down_vote,

downvotes.id AS voted_post,

users.id AS user_vote

FROM (SELECT DISTINCT regexp_split_to_table(downvotes, ',') AS downvote_user,

id FROM bad_posts)

downvotes

INNER JOIN  users

ON downvotes.downvote_user = users.username;


INSERT INTO vote (up_down_vote, voted_post, user_vote)

SELECT 1 AS up_down_vote,

upvotes.id AS voted_post,

users.id AS user_vote
```

```sql
FROM (SELECT DISTINCT regexp_split_to_table(upvotes, ',') AS upvote_user,

id FROM bad_posts)

upvotes

INNER JOIN users

ON upvotes.upvote_user = users.username;



----------------------------------------

COMMENTS

----------------------------------------


INSERT INTO comments (assoc_post, assoc_user, comment_text)

SELECT bc.post_id, u.id, bc.text_content

FROM bad_comments bc

INNER JOIN users u

ON u.username = bc.username;
```