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:

```
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!

1.The username column exists in both tables. To comply with the third normal form (3NF), we should create a separate users table with user_id as the primary key and username as a second column. This reduces redundancy and improves data integrity.

2.The upvotes and downvotes columns currently store usernames separated by commas, which is not efficient for querying and aggregation. Instead, we should split these into a separate votes table and use a value of 1 for individual votes, making it easier to total.

3. The upvotes and downvotes columns are currently stored as TEXT. Changing them to INT will allow for more efficient storage and calculations.

4. The bad_comments.post_id column currently uses BIGINT, while bad_posts.id uses SERIAL, which is typically an INT. Changing bad_comments.post_id to INT ensures consistency and saves storage space.

5. The bad_posts.url column has a VARCHAR limit of 4000 characters. It's best practice to limit URLs to less than 2000 characters. Setting the limit to 500 characters ensures more efficient storage and better performance.

6. The username and title columns in bad_posts are currently nullable. Setting these columns to NOT NULL ensures that every post has a username and title, which helps maintain data integrity and makes it easier to search and filter posts.

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.
        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.

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.
    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:

```
-- Creating the users table
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    username VARCHAR(25) NOT NULL UNIQUE,
    last_login TIMESTAMP,
    CONSTRAINT chk_username_not_empty CHECK (LENGTH(TRIM(username)) > 0)
);

-- Creating the topics table
CREATE TABLE topics (
    id SERIAL PRIMARY KEY,
    name VARCHAR(30) NOT NULL UNIQUE,
    description VARCHAR(500),
    CONSTRAINT chk_topic_name_not_empty CHECK (LENGTH(TRIM(name)) > 0)
);

-- Creating the posts table
CREATE TABLE posts (
```

```
    id SERIAL PRIMARY KEY,
    title VARCHAR(100) NOT NULL,
    url VARCHAR(500) DEFAULT NULL,
    text_content TEXT DEFAULT NULL,
    topic_id INT NOT NULL,
    user_id INT DEFAULT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT chk_url_or_text CHECK (
        (url IS NOT NULL AND text_content IS NULL) OR
        (url IS NULL AND text_content IS NOT NULL)
    ),
    CONSTRAINT chk_post_title_not_empty CHECK (LENGTH(TRIM(title)) > 0),
    FOREIGN KEY (topic_id) REFERENCES topics(id) ON DELETE CASCADE,
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE SET NULL
);

-- Adding indexes to posts table
CREATE INDEX idx_posts_url ON posts(url);
CREATE INDEX idx_posts_topic_id ON posts(topic_id);
CREATE INDEX idx_posts_user_id ON posts(user_id);

-- Creating the comments table
CREATE TABLE comments (
    id SERIAL PRIMARY KEY,
    text_content TEXT NOT NULL,
    post_id INT NOT NULL,
    user_id INT DEFAULT NULL,
    parent_comment_id INT DEFAULT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT chk_comment_text_not_empty CHECK (LENGTH(TRIM(text_content)) > 0),
    FOREIGN KEY (post_id) REFERENCES posts(id) ON DELETE CASCADE,
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE SET NULL,
    FOREIGN KEY (parent_comment_id) REFERENCES comments(id) ON DELETE CASCADE
);

-- Adding indexes to comments table
CREATE INDEX idx_comments_post_id ON comments(post_id);
CREATE INDEX idx_comments_user_id ON comments(user_id);
CREATE INDEX idx_comments_parent_comment_id ON comments(parent_comment_id);

-- Creating the votes table
CREATE TABLE votes (
    id SERIAL PRIMARY KEY,
    post_id INT NOT NULL,
    user_id INT DEFAULT NULL,
    vote_value INT NOT NULL,
    CONSTRAINT chk_vote_value_valid CHECK (vote_value IN (-1, 1)),
    CONSTRAINT unique_vote_per_post_per_user UNIQUE (post_id, user_id),
    FOREIGN KEY (post_id) REFERENCES posts(id) ON DELETE CASCADE,
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE SET NULL
);
```

```sql
-- Adding indexes to votes table
CREATE INDEX idx_votes_post_id ON votes(post_id);
CREATE INDEX idx_votes_user_id ON votes(user_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:

```
--Populate`users`table with existing data
INSERT INTO users ("username")
    SELECT DISTINCT username
    FROM bad_posts
UNION
    SELECT DISTINCT username
    FROM bad_comments
UNION
    SELECT DISTINCT REGEXP_SPLIT_TO_TABLE(upvotes, ',')
    FROM bad_posts UNION
    SELECT DISTINCT REGEXP_SPLIT_TO_TABLE(downvotes, ',')
    FROM bad_posts;

--Populate`topics`table with existing data
INSERT INTO topics ("name")
    SELECT DISTINCT topic
    FROM bad_posts;


-- Populate `posts` table with existing data
INSERT INTO posts("id","title","url","text_content","topic_id", "user_id")
```

```
    SELECT
        bp.id,
        LEFT(bp.title,100),
        bp.url,
        bp.text_content,
        t.id,
        u.id
FROM bad_posts AS bp
INNER JOIN users AS u
ON bp.username=u.username
INNER JOIN topics AS t
ON bp.topic = t.name;

-- Populate `comments` table with existing data
INSERT INTO comments("text_content","post_id","user_id")
SELECT
    bc.text_content,
    bc.post_id,
    u.id
FROM bad_comments AS bc
INNER JOIN users AS u
ON bc.username = u.username;


-- Populate `votes` table with converted upvotes data
INSERT INTO votes("post_id","user_id","vote_value")
    SELECT
        bp.id,
        u.id,
        1 AS upvote FROM
        (SELECT "id", REGEXP_SPLIT_TO_TABLE("upvotes",',')AS upvote
         FROM bad_posts) bp
JOIN users AS u
ON bp.upvote = u.username;

--Populate`votes`table with converted down votes data
INSERT INTO votes ("post_id", "user_id", "vote_value")
    SELECT
    bp.id,
    u.id,
    -1ASdownvote FROM
    (SELECT "id", REGEXP_SPLIT_TO_TABLE("downvotes",',')AS downvote
    FROM bad_posts) bp
JOIN users AS u
ON bp.downvote = u.username;
```