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!

The current schema of the `bad_posts` and `bad_comments` tables can be improved in the following ways:

1.Normalization of Votes:

1. **Problem**: The `upvotes` and `downvotes` columns in `bad_posts` are stored as comma-separated strings, which is inefficient and difficult to query. This design violates the principles of database normalization, specifically the first normal form, which requires atomicity in columns.
2. **Improvement**: Create a separate `votes` table to store each vote as a separate record. This will make it easier to perform operations such as counting votes or determining if a specific user has voted.

```
CREATE TABLE votes (
    id SERIAL PRIMARY KEY,
    post_id INT NOT NULL,
    user_id INT NOT NULL,
    value INT NOT NULL CHECK (value IN (1, -1)),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (post_id) REFERENCES bad_posts(id),
    FOREIGN KEY (user_id) REFERENCES users(id),
    UNIQUE (post_id, user_id)
);
```

2.Introduction of a Users Table:

- **Problem**: There is no `users` table to ensure unique usernames and manage user data effectively. Usernames are repeated in both `bad_posts` and `bad_comments`, leading to redundancy and potential data inconsistencies.
- **Improvement**: Create a `users` table to store user-related data, ensuring usernames are unique and making it easier to manage user-specific information.

```sql
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    username VARCHAR(25) UNIQUE NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

3.Support for Threaded Comments:

- **Problem**: The current `bad_comments` table structure does not support threaded comments, limiting the ability to create discussion threads or replies within comments.
- **Improvement**: Add a `parent_id` column to the `comments` table, allowing comments to reference parent comments and support nested threads.

```sql
CREATE TABLE comments (
    id SERIAL PRIMARY KEY,
    post_id INT NOT NULL,
    user_id INT NOT NULL,
    parent_id INT DEFAULT NULL,
    text_content TEXT NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (post_id) REFERENCES bad_posts(id),
    FOREIGN KEY (user_id) REFERENCES users(id),
    FOREIGN KEY (parent_id) REFERENCES comments(id)
);
```

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:

```sql
-- Drop tables if they already exist to ensure clean creation
DROP TABLE IF EXISTS votes;
DROP TABLE IF EXISTS comments;
DROP TABLE IF EXISTS posts;
DROP TABLE IF EXISTS topics;
DROP TABLE IF EXISTS users;

-- 1. Guideline #1: Create Tables with Features and Specifications

-- a. Allow new users to register
CREATE TABLE users (
    id SERIAL PRIMARY KEY,  -- Auto-incrementing primary key
    username VARCHAR(25) UNIQUE NOT NULL,  -- Unique and non-empty username
with a maximum of 25 characters
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP -- Timestamp for user
creation
);

-- b. Allow registered users to create new topics
CREATE TABLE topics (
    id SERIAL PRIMARY KEY,  -- Auto-incrementing primary key
    name VARCHAR(30) UNIQUE NOT NULL,  -- Unique and non-empty topic name with
a maximum of 30 characters
    description VARCHAR(500) DEFAULT NULL,  -- Optional description with a
maximum of 500 characters
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP -- Timestamp for topic
creation
);

-- c. Allow registered users to create new posts on existing topics
CREATE TABLE posts (
    id SERIAL PRIMARY KEY,  -- Auto-incrementing primary key
    topic_id INT NOT NULL,  -- Foreign key to topics
    user_id INT DEFAULT NULL,  -- Foreign key to users, nullable to dissociate
if user is deleted
    title VARCHAR(100) NOT NULL,  -- Required non-empty title with a maximum of
100 characters
    url VARCHAR(4000) DEFAULT NULL,  -- URL or text content must be provided,
not both
    text_content TEXT DEFAULT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP, -- Timestamp for post
creation
```

```
    FOREIGN KEY (topic_id) REFERENCES topics(id) ON DELETE CASCADE, -- Cascade
delete if topic is deleted
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE SET NULL,  -- Set NULL
if user is deleted
    CHECK ((url IS NOT NULL AND text_content IS NULL) OR (url IS NULL AND
text_content IS NOT NULL)), -- Ensure only one of url or text_content is
present
    CONSTRAINT chk_post_content CHECK (url IS NOT NULL OR text_content IS NOT
NULL) -- Ensure at least one content is present
);

-- d. Allow registered users to comment on existing posts
CREATE TABLE comments (
    id SERIAL PRIMARY KEY,  -- Auto-incrementing primary key
    post_id INT NOT NULL,  -- Foreign key to posts
    user_id INT DEFAULT NULL,  -- Foreign key to users, nullable to dissociate
if user is deleted
    parent_id INT DEFAULT NULL,  -- Self-referential foreign key for comment
threads
    text_content TEXT NOT NULL,  -- Required non-empty text content
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP, -- Timestamp for comment
creation
    FOREIGN KEY (post_id) REFERENCES posts(id) ON DELETE CASCADE, -- Cascade
delete if post is deleted
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE SET NULL, -- Set NULL
if user is deleted
    FOREIGN KEY (parent_id) REFERENCES comments(id) ON DELETE CASCADE --
Cascade delete if parent comment is deleted
);

-- e. Make sure that a given user can only vote once on a given post
CREATE TABLE votes (
    id SERIAL PRIMARY KEY,  -- Auto-incrementing primary key
    post_id INT NOT NULL,  -- Foreign key to posts
    user_id INT DEFAULT NULL,  -- Foreign key to users, nullable to dissociate
if user is deleted
    value INT NOT NULL CHECK (value IN (1, -1)),  -- Vote value must be 1
(upvote) or -1 (downvote)
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP, -- Timestamp for vote
creation
    FOREIGN KEY (post_id) REFERENCES posts(id) ON DELETE CASCADE, -- Cascade
delete if post is deleted
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE SET NULL, -- Set NULL
if user is deleted
```

```sql
    UNIQUE (post_id, user_id),  -- Ensure a user can only vote once per post
    CONSTRAINT chk_vote_value CHECK (value IN (1, -1)) -- Ensure valid vote
values
);

-- 2. Guideline #2: Queries (Conceptual)

-- 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: Normalization, Constraints, and Indexes

-- Indexes for optimization
CREATE INDEX idx_user_username ON users(username); -- Index for quick username
lookup
CREATE INDEX idx_topic_name ON topics(name); -- Index for quick topic name
lookup
CREATE INDEX idx_post_title ON posts(title); -- Index for quick post title
lookup
CREATE INDEX idx_comment_hierarchy ON comments(parent_id); -- Index for quick
comment hierarchy traversal
CREATE INDEX idx_vote_post_user ON votes(post_id, user_id); -- Index for quick
vote lookup by post and user

-- Add comments to explain constraints
COMMENT ON CONSTRAINT chk_post_content ON posts IS 'Ensures posts have either a
URL or text content, but not both.';
COMMENT ON CONSTRAINT chk_vote_value ON votes IS 'Ensures votes have a value of
either 1 (upvote) or -1 (downvote).';

-- 4. Guideline #4: Use of Auto-incrementing Primary Key
```

```
-- The primary keys for all tables are set as SERIAL, which is an auto-
incrementing integer in PostgreSQL.
```

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:

```
-- Migrate users from bad_posts and bad_comments
INSERT INTO users (username)
SELECT DISTINCT username
FROM (
    SELECT username FROM bad_posts
    UNION
    SELECT username FROM bad_comments
    UNION
    SELECT unnest(string_to_array(upvotes, ',')) AS username FROM bad_posts
    UNION
    SELECT unnest(string_to_array(downvotes, ',')) AS username FROM bad_posts
) AS all_users
WHERE username IS NOT NULL;
```

```sql
-- Migrate topics
INSERT INTO topics (name)
SELECT DISTINCT topic
FROM bad_posts;

-- Migrate posts
INSERT INTO posts (topic_id, user_id, title, url, text_content)
SELECT
    t.id AS topic_id,
    u.id AS user_id,
    b.title,
    b.url,
    b.text_content
FROM
    bad_posts b
JOIN
    topics t ON b.topic = t.name
LEFT JOIN
    users u ON b.username = u.username;

-- Migrate comments
INSERT INTO comments (post_id, user_id, text_content)
SELECT
    p.id AS post_id,
    u.id AS user_id,
    c.text_content
FROM
    bad_comments c
JOIN
    posts p ON c.post_id = p.id
LEFT JOIN
    users u ON c.username = u.username;

-- Migrate votes (upvotes)
INSERT INTO votes (post_id, user_id, value)
SELECT
    p.id AS post_id,
    u.id AS user_id,
    1 AS value
FROM (
    SELECT
        title,
        unnest(string_to_array(upvotes, ',')) AS username
```

```
    FROM
        bad_posts
) AS votes
JOIN
    posts p ON votes.title = p.title
LEFT JOIN
    users u ON votes.username = u.username;

-- Migrate votes (downvotes)
INSERT INTO votes (post_id, user_id, value)
SELECT
    p.id AS post_id,
    u.id AS user_id,
    -1 AS value
FROM (
    SELECT
        title,
        unnest(string_to_array(downvotes, ',')) AS username
    FROM
        bad_posts
) AS votes
JOIN
    posts p ON votes.title = p.title
LEFT JOIN
    users u ON votes.username = u.username;
```

Explanation
User Migration: We extract all unique usernames from bad_posts and bad_comments and include those from upvotes and downvotes, using unnest to split the comma-separated values.

Topic Migration: Extract unique topic names from bad_posts.

Post Migration: Associate each post with a topic and user using the titles and usernames.

Comment Migration: Insert comments with post and user associations. All comments are migrated as top-level comments due to the lack of threading in the old schema.

Vote Migration: Use unnest on the upvotes and downvotes to split the values into individual rows, and then associate them with posts and users. We use a value of 1 for upvotes and -1 for downvotes.