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!

Issue: User/topic names stored as free text

Fix:

add users/topics tables with UNIQUE + length limits and FKs.

Issue: Votes stored as comma-separated text

Fix:

add votes table with CHECK (value IN (-1,1)), UNIQUE (user_id, post_id), ON DELETE CASCADE on post.

Issue: Comments lack FKs and threading

Fix:

add comments with FK to posts (cascade), self-FK parent_comment_id (cascade).

Issue: No content constraints

Fix:

enforce title ≤100 chars, username ≤25, topic ≤30, non-empty text, and body exclusivity (url IS NOT NULL) <> (text_content IS NOT NULL).

Issue: Missing indexes

Fix:

add indexes for topic/user recent posts, URL lookup, post/comment retrieval, and vote scoring.

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:

Guideline #1: Features and Specifications

Here is a list of features and specifications that Udiddit needs in order to support its website and administrative interface:

- **Allow new users to register:**
    - Each username has to be unique
    - Usernames can be composed of at most 25 characters
    - Usernames can't be empty
    - We won't worry about user passwords for this project
- **Allow registered users to create new topics:**
    - Topic names have to be unique
    - The topic's name is at most 30 characters
    - The topic's name can't be empty
    - Topics can have an optional description of at most 500 characters
- **Allow registered users to create new posts on existing topics:**
    - Posts have a required title of at most 100 characters
    - The title of a post can't be empty
    - Posts should contain either a URL or a text content, but not both
    - If a topic gets deleted, all the posts associated with it should be automatically deleted too
    - If the user who created the post gets deleted, then the post will remain, but it will become dissociated from that user
- **Allow registered users to comment on existing posts:**
    - A comment's text content can't be empty
    - Contrary to the current linear comments, the new structure should allow comment threads at arbitrary levels
    - If a post gets deleted, all comments associated with it should be automatically deleted too
    - If the user who created the comment gets deleted, then the comment will remain, but it will become dissociated from that user
    - If a comment gets deleted, then all its descendants in the thread structure should be automatically deleted too
- **Make sure that a given user can only vote once on a given post:**

    - Hint: you can store the (up/down) value of the vote as the values 1 and -1 respectively
    - If the user who cast a vote gets deleted, then all their votes will remain, but will become dissociated from the user
    - If a post gets deleted, then all the votes for that post should be automatically deleted too

Guideline #2: Required Queries

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.

- List all users who haven't logged in in the last year
- List all users who haven't created any post
- Find a user by their username
- List all topics that don't have any posts
- Find a topic by its name
- List the latest 20 posts for a given topic
- List the latest 20 posts made by a given user
- Find all posts that link to a specific URL, for moderation purposes
- List all the top-level comments (those that don't have a parent comment) for a given post
- List all the direct children of a parent comment
- List the latest 20 comments made by a given user
- Compute the score of a post, defined as the difference between the number of upvotes and the number of downvotes

Guideline #3: Technical Requirements

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.

Guideline #4: Database Structure

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
-- 1) Users
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    username VARCHAR(25) NOT NULL,
    last_login_at TIMESTAMPTZ DEFAULT NOW(),
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    CONSTRAINT uq_users_username UNIQUE (username)
);

-- 2) Topics
CREATE TABLE topics (
    id SERIAL PRIMARY KEY,
    name VARCHAR(30) NOT NULL,
    description VARCHAR(500),
    created_by INTEGER REFERENCES users(id) ON DELETE SET NULL,
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    CONSTRAINT uq_topics_name UNIQUE (name)
);

-- 3) Posts
CREATE TABLE posts (
    id SERIAL PRIMARY KEY,
    topic_id INTEGER NOT NULL,
    user_id INTEGER,
    title VARCHAR(100) NOT NULL,
    url TEXT,
    text_content TEXT,
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    CONSTRAINT fk_posts_topic FOREIGN KEY (topic_id) REFERENCES
topics(id) ON DELETE CASCADE,
    CONSTRAINT fk_posts_user FOREIGN KEY (user_id) REFERENCES users(id)
```

```sql
ON DELETE SET NULL,
    CONSTRAINT ck_posts_body_exclusive CHECK (
        (url IS NOT NULL AND text_content IS NULL)
        OR (url IS NULL AND text_content IS NOT NULL)
    )
);
CREATE INDEX idx_posts_topic_created_at ON posts (topic_id, created_at
DESC);
CREATE INDEX idx_posts_user_created_at ON posts (user_id, created_at
DESC);
CREATE INDEX idx_posts_url ON posts (url);

-- 4) Comments
CREATE TABLE comments (
    id SERIAL PRIMARY KEY,
    post_id INTEGER NOT NULL,
    user_id INTEGER,
    parent_comment_id INTEGER,
    text_content TEXT NOT NULL,
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    CONSTRAINT fk_comments_post FOREIGN KEY (post_id) REFERENCES
posts(id) ON DELETE CASCADE,
    CONSTRAINT fk_comments_user FOREIGN KEY (user_id) REFERENCES
users(id) ON DELETE SET NULL,
    CONSTRAINT fk_comments_parent FOREIGN KEY (parent_comment_id)
REFERENCES comments(id) ON DELETE CASCADE
);
CREATE INDEX idx_comments_post_parent ON comments (post_id,
parent_comment_id);
CREATE INDEX idx_comments_user_created_at ON comments (user_id,
created_at DESC);

-- 5) Votes
CREATE TABLE votes (
    id SERIAL PRIMARY KEY,
    post_id INTEGER NOT NULL,
    user_id INTEGER,
    value SMALLINT NOT NULL,
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    CONSTRAINT ck_votes_value CHECK (value IN (-1, 1)),
    CONSTRAINT fk_votes_post FOREIGN KEY (post_id) REFERENCES posts(id)
ON DELETE CASCADE,
    CONSTRAINT fk_votes_user FOREIGN KEY (user_id) REFERENCES users(id)
ON DELETE SET NULL,
    CONSTRAINT uq_votes_user_post UNIQUE (user_id, post_id)
);
CREATE INDEX idx_votes_post ON votes (post_id);
CREATE INDEX idx_votes_user 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:

- Topic descriptions can all be empty
- 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
- You can use the Postgres string function `regexp_split_to_table` to unwind the comma-separated votes values into separate rows
- Don't forget that some users only vote or comment, and haven't created any posts. You'll have to create those users too
- The order of your migrations matter! For example, since posts depend on users and topics, you'll have to migrate the latter first

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.

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:

```
TRUNCATE votes, comments, posts, topics, users RESTART IDENTITY;

-- 1) Users from posts, comments, vote lists
WITH all_usernames AS (
    SELECT username FROM bad_posts
    UNION
    SELECT username FROM bad_comments
    UNION
    SELECT btrim(regexp_split_to_table(upvotes, ',')) FROM bad_posts
WHERE upvotes IS NOT NULL
    UNION
    SELECT btrim(regexp_split_to_table(downvotes, ',')) FROM bad_posts
WHERE downvotes IS NOT NULL
)
```

```sql
INSERT INTO users (username, last_login_at)
SELECT DISTINCT username, NOW()
FROM all_usernames
WHERE username IS NOT NULL AND btrim(username) <> '';

-- 2) Topics
INSERT INTO topics (name, description)
SELECT DISTINCT topic AS name, '' AS description
FROM bad_posts
WHERE topic IS NOT NULL AND btrim(topic) <> '';

-- 3) Posts (reuse IDs; enforce length/exclusivity)
INSERT INTO posts (id, topic_id, user_id, title, url, text_content,
created_at)
SELECT
    bp.id,
    t.id AS topic_id,
    u.id AS user_id,
    LEFT(COALESCE(NULLIF(bp.title, ''), '(untitled)'), 100) AS title,
    CASE WHEN bp.url IS NOT NULL AND btrim(bp.url) <> '' THEN bp.url
END AS url,
    CASE
        WHEN bp.url IS NOT NULL AND btrim(bp.url) <> '' THEN NULL
        ELSE COALESCE(NULLIF(bp.text_content, ''), 'No content
provided')
    END AS text_content,
    NOW()
FROM bad_posts bp
JOIN topics t ON t.name = bp.topic
JOIN users u ON u.username = bp.username;
SELECT setval('posts_id_seq', (SELECT MAX(id) FROM posts));

-- 4) Comments (all top-level)
INSERT INTO comments (id, post_id, user_id, parent_comment_id,
text_content, created_at)
SELECT
    bc.id,
    p.id AS post_id,
    u.id AS user_id,
    NULL AS parent_comment_id,
    bc.text_content,
    NOW()
FROM bad_comments bc
JOIN posts p ON p.id = bc.post_id
JOIN users u ON u.username = bc.username;
SELECT setval('comments_id_seq', (SELECT MAX(id) FROM comments));

-- 5) Votes (unwind and dedupe)
WITH raw_votes AS (
    SELECT bp.id AS post_id, btrim(regexp_split_to_table(bp.upvotes,
',')) AS username, 1 AS value
    FROM bad_posts bp
    WHERE bp.upvotes IS NOT NULL
    UNION ALL
    SELECT bp.id AS post_id, btrim(regexp_split_to_table(bp.downvotes,
',')) AS username, -1 AS value
    FROM bad_posts bp
    WHERE bp.downvotes IS NOT NULL
),
deduped AS (
    SELECT DISTINCT post_id, username, value
    FROM raw_votes
    WHERE username IS NOT NULL AND username <> ''
)
INSERT INTO votes (post_id, user_id, value, created_at)
SELECT
    d.post_id,
    u.id AS user_id,
    d.value,
    NOW()
FROM deduped d
JOIN users u ON u.username = d.username
JOIN posts p ON p.id = d.post_id;
SELECT setval('votes_id_seq', (SELECT MAX(id) FROM votes));
```