# 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 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
);
```

However, you have to keep in mind that you need to create different tables for USERS, TOPICS, POSTS, COMMENTS & VOTES.

## 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 'upvotes' column in the bad_posts table should be normalised into a new table, really.
2. So also, the 'downvotes' should be normalized
3. In the bad_comments table, the username there is a bit repetitive; we can normalise it to use username_id instead
4. There are multiple 'usernames' in the bad_posts column that can be wrangled

## 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: 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 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: 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 posts.
    - [ ] 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: You'll need to use normalisation, 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: 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) UNIQUE NOT NULL,
    "last_logged_in" TIMESTAMP,
    CONSTRAINT "non_empty_username" CHECK (LENGTH(TRIM(username)) > 0)
);


-- List all users who haven't logged in in the last year.
CREATE INDEX last_login_idx ON users ("last_logged_in");
-- Find a user by their username.
Already an index (due to the UNIQUE constraint!)


CREATE TABLE topics(
    "id" SERIAL PRIMARY KEY,
    "user_id" INTEGER REFERENCES "users"("id"),
    "topic_name" VARCHAR(30) UNIQUE NOT NULL,
    "description" VARCHAR(500) NULL,
    CONSTRAINT "non_empty_topic" CHECK (LENGTH(TRIM("topic_name")) > 0)
);


-- Find a topic by its name.
ALready an index (UNIQUE constraint during creation!)


CREATE TABLE posts(
    "id" SERIAL PRIMARY KEY,
    "user_id" INTEGER REFERENCES "users"("id") ON DELETE SET NULL,
    "topic_id" INTEGER REFERENCES "topics"("id") ON DELETE CASCADE NOT NULL,
    "title" VARCHAR(100) NOT NULL,
    "url" TEXT,
    "text_content" TEXT,
    "post_time" TIMESTAMP,
    CONSTRAINT "url_or_text_content" CHECK (
        ("url" IS NOT NULL AND "text_content" IS NULL) OR
        ("url" IS NULL AND "text_content" IS NOT NULL)
    ),
    CONSTRAINT "post_title_not_empty" CHECK (LENGTH(TRIM("title")) > 0)
);


-- List all users who haven't created any posts.
CREATE INDEX user_post_idx ON "posts" ("user_id");
-- List all topics that don't have any posts.
CREATE INDEX post_topics_idx ON "posts" ("topic_id");
```

```sql
-- List the latest 20 posts for a given topic.
CREATE INDEX post_topic_time_idx ON posts("topic_id", "post_time");
-- List the latest 20 posts made by a given user.
CREATE INDEX user_posttime_idx ON "posts" ("user_id", "post_time");
-- Find all posts that link to a specific URL, for moderation purposes
CREATE INDEX post_url_idx ON "posts" ("url");

CREATE TABLE comments(
    "id" SERIAL PRIMARY KEY,
    "user_id" INTEGER REFERENCES "users" ("id") ON DELETE SET NULL,
    "post_id" INTEGER REFERENCES "posts" ("id") ON DELETE CASCADE NOT NULL,
    "text_content" TEXT NOT NULL CHECK (LENGTH(TRIM(text_content)) > 0),
    "parent_id" INTEGER REFERENCES "comments"("id") ON DELETE CASCADE,
    "time" TIMESTAMP
);


-- List all the top-level comments (those that don't have a parent comment) for a given post.
CREATE INDEX top_level_comments_idx ON "comments" ("post_id") WHERE "parent_id" IS NULL;
-- List all the direct children of a parent comment.
CREATE INDEX child_idx ON "comments" ("parent_id");
-- List the latest 20 comments made by a given user.
CREATE INDEX comment_time_idx ON "comments" ("user_id", "time");


CREATE TABLE votes (
    id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES users(id) ON DELETE SET NULL,
    post_id INTEGER REFERENCES posts(id) ON DELETE CASCADE,
    vote_value INTEGER CHECK (vote_value IN (1, -1)),
    CONSTRAINT unique_user_post_vote UNIQUE (user_id, post_id)
);



-- Compute the score of a post, defined as the difference between the number of upvotes and the number of
downvotes
CREATE INDEX vote_score_idx ON "votes" ("post_id", "vote_value");
```

## 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 matters! 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:

```
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";
```

```sql
INSERT INTO "topics" ("topic_name")
SELECT DISTINCT "topic"
FROM "bad_posts"
WHERE LENGTH(TRIM("topic")) > 0 AND LENGTH("topic") < 31;


INSERT INTO "posts" ("user_id", "topic_id", "title", "url", "text_content")
SELECT
    "users"."id",
    "topics"."id",
    LEFT("bad_posts"."title", 100),
    "bad_posts"."url",
    "bad_posts"."text_content"
FROM "bad_posts"
JOIN "users" ON "users"."username" = "bad_posts"."username"
JOIN "topics" ON "topics"."topic_name" = "bad_posts"."topic";




INSERT INTO "comments" ("text_content", "post_id", "user_id")
SELECT
    "bad_comments"."text_content",
    "bad_comments"."post_id",
    "users"."id"
FROM "bad_comments"
JOIN "users" ON "users"."username" = "bad_comments"."username"
WHERE "bad_comments"."post_id" IN (SELECT "id" FROM "posts");
```

```sql
INSERT INTO votes (user_id, post_id, vote_value)
SELECT
    users.id,
    bad_posts.id,
    1
FROM bad_posts
CROSS JOIN regexp_split_to_table(bad_posts.upvotes, ',') AS upvote_username
JOIN users ON users.username = upvote_username
UNION ALL
SELECT
    users.id,
    bad_posts.id,
    -1
FROM bad_posts
CROSS JOIN regexp_split_to_table(bad_posts.downvotes, ',') AS downvote_username
JOIN users ON users.username = downvote_username;
```