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:*

|  |
| --- |
| * By using a “\d+” query, it looks like the schema has no constraints whatsoever. For example, the “URL” column where we might want to ensure the URL is input in a valid format. This could be done using a POSIX regular expression. * “upvotes” and “downvotes” columns have a TEXT data type. This sounds inappropriate as we are not expecting it to be a long text or paragraph but a number. It would make more sense for it to be a INT. This would cover both positive and negative numbers. I would opt for INT rather than SMALLINT as sometimes posts/comments can become very un/popular and get thousands and thousands of votes. If the feedback leans heavily on either side, the SMALLINT might not be enough. * The “bad\_posts” table shows several columns that should really be split into separate table, e.g. “Users”, “Topics”, “Votes”. |

## 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. 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:
   1. Allow new users to register:
      1. Usernames can be composed of at most 25 characters
      2. We won’t worry about user passwords for this project
   2. Allow registered users to create new topics:
      1. The topic’s name is at most 30 characters
      2. Topics can have an optional description of at most 500 characters.
   3. Allow registered users to create new posts on existing topics:
      1. Posts have a required title of at most 100 characters
      2. Posts should contain either a URL or a text content, but not both.
      3. If a topic gets deleted, all the posts associated with it should be automatically deleted too.
      4. If the user who created the post gets deleted, then the post will remain, but it will become dissociated from that user.
   4. Allow registered users to comment on existing posts:
      1. Contrary to the current linear comments, the new structure should allow comment threads at arbitrary levels.
      2. If a post gets deleted, all comments associated with it should be automatically deleted too.
      3. If the user who created the comment gets deleted, then the comment will remain, but it will become dissociated from that user.
      4. If a comment gets deleted, then all its descendants in the thread structure should be automatically deleted too.
   5. Make sure that a given user can only vote once on a given post:
      1. If the user who cast a vote gets deleted, then all their votes will remain, but will become dissociated from the user.
      2. 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.
   1. List all users who haven’t logged in in the last year.
   2. List all users who haven’t created any post.
   3. Find a user by their username.
   4. List all topics that don’t have any posts.
   5. Find a topic by its name.
   6. List the latest 20 posts for a given topic.
   7. List the latest 20 posts made by a given user.
   8. Find all posts that link to a specific URL, for moderation purposes.
   9. List all the top-level comments (those that don’t have a parent comment) for a given post.
   10. List all the direct children of a parent comment.
   11. List the latest 20 comments made by a given user.
   12. Compute the score of a post, defined as the difference between the number of upvotes and the number of downvotes

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

|  |
| --- |
| CREATE TABLE "users" ( "id" SERIAL PRIMARY KEY UNIQUE, "username" VARCHAR(25) UNIQUE NOT NULL, "last\_login" TIMESTAMP );  CREATE UNIQUE INDEX unique\_username\_ind ON "users" (LOWER("username") VARCHAR\_PATTERN\_OPS); ALTER TABLE "users" ADD CONSTRAINT "username\_not\_empty" CHECK (LENGTH(TRIM("username")) > 0);   CREATE TABLE "topics" ( "id" SERIAL PRIMARY KEY UNIQUE, "name" VARCHAR(30) UNIQUE NOT NULL, "description" VARCHAR(500) );  Create INDEX topics\_name\_ind ON "topics" ("name"); ALTER TABLE "topics" ADD CONSTRAINT "topic\_name\_not\_empty" CHECK (LENGTH(TRIM("name")) > 0);  CREATE TABLE "posts" ( "id" SERIAL PRIMARY KEY UNIQUE, "title" VARCHAR(100) UNIQUE NOT NULL, "url" VARCHAR(500),  "text\_content" TEXT, "topic\_id" INTEGER REFERENCES "topics" ("id") ON DELETE CASCADE, "user\_id" INTEGER REFERENCES "users" ("id") ON DELETE SET NULL, "created\_on" TIMESTAMP );  CREATE INDEX post\_url\_ind ON "posts" ("url"); ALTER TABLE "posts"  ADD CONSTRAINT "url\_or\_text" CHECK (("url" IS NOT NULL AND "text\_content" IS NULL) OR ("url" IS NULL AND "text\_content" IS NOT NULL)); ALTER TABLE "posts" ADD CONSTRAINT "post\_name\_not\_empty" CHECK (LENGTH(TRIM("title")) > 0);  CREATE TABLE "comments" ( "id" SERIAL PRIMARY KEY UNIQUE, "text\_content" TEXT NOT NULL, "post\_id" INTEGER REFERENCES "posts" ("id") ON DELETE CASCADE, "user\_id" INTEGER REFERENCES "users" ("id") ON DELETE SET NULL, "created\_on" DATE, "parent\_comment\_id" INT REFERENCES "comments" ("id") ON DELETE CASCADE );  CREATE INDEX comment\_ind ON "comments" ("parent\_comment\_id", "id");   CREATE TABLE "votes" ( "id" SERIAL PRIMARY KEY UNIQUE, "value" SMALLINT, "user\_id" INTEGER REFERENCES "users" ("id") ON DELETE SET NULL, "post\_id" INTEGER REFERENCES "posts" ("id") ON DELETE CASCADE );  CREATE INDEX post\_vote\_ind ON "votes" ("post\_id");  ALTER TABLE "votes" ADD CONSTRAINT "up\_down\_vote" CHECK ("value" = 1 OR "value" = -1); ALTER TABLE "votes" ADD CONSTRAINT "one\_vote" UNIQUE ("user\_id", "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.*

*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( SELECT DISTINCT username FROM bad\_posts  UNION SELECT DISTINCT username FROM bad\_comments UNION SELECT DISTINCT regexp\_split\_to\_table("upvotes", ',') AS "username" FROM bad\_posts UNION SELECT DISTINCT regexp\_split\_to\_table("downvotes", ',') AS "username" FROM bad\_posts) sub1 order by 1;  INSERT INTO "topics" ("name") SELECT DISTINCT topic FROM "bad\_posts";   INSERT INTO "posts" ("title", "url", "text\_content", "topic\_id", "user\_id") SELECT LEFT(bp.title, 100), bp.url, bp.text\_content, topics.id, users.id FROM bad\_posts bp JOIN topics ON bp.topic = topics.name JOIN users ON bp.username = users.username;   INSERT INTO "comments" ("text\_content", "post\_id", "user\_id") SELECT bc.text\_content, posts.id, users.id FROM bad\_comments bc JOIN users ON bc.username = users.username JOIN posts ON posts.id = bc.post\_id;  INSERT INTO "votes" ("value", "user\_id", "post\_id") SELECT 1 AS upvotes, users.id, bp.id FROM (SELECT id, regexp\_split\_to\_table("upvotes", ',') AS "username" FROM bad\_posts) bp JOIN users ON bp.username = users.username  UNION  SELECT -1 AS downvotes, users.id, bp.id FROM (SELECT id, regexp\_split\_to\_table("downvotes", ',') AS "username" FROM bad\_posts) bp JOIN users ON bp.username = users.username; |