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,
      topics 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 projects SQL workspace. Then, in your own words, outline three (3) specific things that could be improved about this schema. Dont hesitate to outline more if you want to stand out!

1. On the DDL they are using TEXT to define the upvotes and downvotes. In regards to normal form it would be better for the counts to be recorded in INTERGERS.
2. Removal of BIGINT would lead to easier identification of the ID Numbers.
3. Need to add constraints to the data sets.

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 cant be empty
        iv. We wont worry about user passwords for this project
    b. Allow registered users to create new topics:
        i. topics names have to be unique.
        ii. The topicss name is at most 30 characters
        iii. The topicss name cant 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 cant be empty.
        iii. Posts should contain either a URL or a text content, **but not both**.
        iv. If a topics 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 comments text content cant 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 dont 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 havent logged in in the last year.
    b. List all users who havent created any post.
    c. Find a user by their username.
    d. List all topics that dont have any posts.
    e. Find a topics by its name.
    f. List the latest 20 posts for a given topics.
    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 dont 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: youll 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 youve taken the time to think about your new schema, write the DDL for it in the space provided here:

```
CREATE TABLE users (
 id SERIAl,
 username VARCHAR(25) UNIQUE NOT NULL,
 log_in_time TIMESTAMP,
 CONSTRAINT not_empty CHECK (length(trim(username)) > 0),
 PRIMARY KEY (id)
);
CREATE INDEX find_user_username ON users (username);
```

```
CREATE INDEX username_index on users (username);
create table topics (
  id SERIAL not null,
  topics_name VARCHAR(30) UNIQUE NOT NULL,
  topics_description VARCHAR(500),
  user_id INTEGER,
  constraint not_empty check (length(trim(topics_name)) > 0),
  primary key (id),
  foreign key (user_id) references users(id)
);
 create INDEX find_topics on topics (topics_name);
 create table posts (
   id serial not null,
   post_title varchar(100) unique not null,
   post_url text,
   post_text text,
   time_of_post timestamp,
   topics_id integer,
   user_id integer,
   constraint not_empty check(length(trim(post_title)) > 0),
  constraint url_or_text check (
  (post_url is not null and post_text is null) or (post_url is null and post_text is
not null)),
  primary key (id),
  foreign key (topics_id) references topics (id) on delete set null,
  foreign key (user_id) references users (id) on delete set null
  );
 create index find_title on posts (post_title);
 create index find_url on posts (post_url);
 create index posts_for_user on posts (topics_id);
  create table comments (
   id serial not null,
   comment Varchar unique,
   posts_id integer,
   user_id integer,
   parent_comment_id integer default null,
   constraint not_empty check(length(trim(comment)) > 0),
   primary key (id),
```

```
  foreign key (posts_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
 );
 create index children_comments on comments (parent_comment_id);
 CREATE TABLE votes (
  PRIMARY KEY (user_id, post_id),
  user_id INTEGER REFERENCES users ON DELETE SET NULL,
  post_id INTEGER REFERENCES posts ON DELETE CASCADE,
  vote INTEGER CONSTRAINT up_down_vote CHECK(vote=1 OR vote=-1)
);
```

Part III: Migrate the provided data

Now that your new schema is created, its time to migrate the data from the provided schema in the projects SQL Workspace to your own schema. This will allow you to review some DML and DQL concepts, as youll be using INSERT…SELECT queries to do so. Here are a few guidelines to help you in this process:

1. topics descriptions can all be empty
2. Since the bad_comments table doesnt 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. Dont forget that some users only vote or comment, and havent created any posts. Youll have to create those users too.
5. The order of your migrations matter! For example, since posts depend on users and topics, youll 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
"username"
From
"bad_posts"
Union
Select
"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;
Insert Into "topics" ("topics_name")
Select
Distinct "topic"
From
"bad_posts";
Insert Into "posts" (
"user_id", "topics_id", "post_title", "post_url",
"post_text"
)
Select
"u"."id",
"t"."id",
left("b"."title", 100),
"b"."url",
"b"."text_content"
From
users u
Join bad_posts b On ("u"."username" = "b"."username")
Join "topics" t On ("b"."topic" = "t"."topics_name");
Insert into "comments" (
"user_id", "posts_id"
)
Select
"u"."id",
"p"."id"
From
bad_comments bc
Join users u On "u"."username" = "bc"."username"
Join "posts" p On "p"."id" = "bc"."post_id";
INSERT INTO "votes" ("post_id", "user_id", "vote")
SELECT
bp.id,
```

```
u.id,
-1 AS vote_tally
FROM
(
SELECT
id,
regexp_split_to_table(downvotes, ',') AS username
FROM
bad_posts
) bp
INNER JOIN users u ON u.username = bp.username;
INSERT INTO "votes" ("post_id", "user_id", "vote")
SELECT
bp.id,
u.id,
1 AS vote_tally
FROM
(
SELECT
id,
regexp_split_to_table(upvotes, ',') AS username
FROM
bad_posts
) bp
INNER JOIN users u ON u.username = bp.username;
```