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!

Current Schema:

There are some glaring issues with the current schema:

- Comma-separated upvotes and downvotes in the same record.
- No proper indexing
- Data is not separated into appropriate tables that would allow us to create primary keys
- No null constraints on columns like topic, username, post titles, etc.

We want to normalize the "bad_posts" table to contain a single upvote/ downvote per record using REGEXP_SPLIT_TO_TABLE.
We then want to separate the data into 5 new tables: "users", "topics", "posts", "comments", and "votes".

New Schema:

"users" and **"topics"**

- "username" and "topic_name":
  - should be unique.
  - should have a maximum of 25 characters and a minimum of 3 characters.
  - not be null.
- We need an index for the lowercase "username" in "users".
- We need an index for the lowercase "topic_name" in "topics".
- We can add an index on lowercase "description" to be able to search the column as well.
- We need to add timestamp columns such as:
  - "last_login" in "users, defaults to now()
  - "created_date" in "users" and "topics", defaults to now()

"posts"

- "post_title" should:
  - cannot be null
  - should have a maximum of 100 characters and a minimum of 3 characters.
- The "user_id" and "topic_id" columns are foreign keys and refer ids on "users" and "topics" respectively.
- If a user or a topic is deleted, the post should be deleted automatically.
  The foreign keys should be set to ON DELETE CASCADE.

- The post should have “url” or “text_content” but not both.
  Create a CONSTRAINT where (“url” is null but not “text_content”) or (“text_content” is null but not the “url”)
- We need time stamps for "created_date" , defaults to now()
- We need an index for the lowercase "text_content" to search posts’ full text.

“comments”

- The comment “text” cannot be null and has a maximum of 100 characters.
- All comments need a “parent_comment_id” INTEGER field
- The “parent_comment_id” field is a foreign key that references a comment’s “id”.
  For new comments on a post (comments not part of a threat of comments) this field refers to the “id” on the same record.

“Votes”

- Each record has foreign keys referring to the “user_id” and “post_id”
- Each vote has a unique constraint on “user_id” and “post_id” (one vote per user per post).
- Each vote has a value of 1 or -1 (upvote v.s. downvote), which is enforced with a constraint on the “votes_value” column: INTEGER with ABS = 1.

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:
    1. Allow new users to register:
        1. Each username has to be unique
        2. Usernames can be composed of at most 25 characters
        3. Usernames can’t be empty
        4. We won’t worry about user passwords for this project
    2. Allow registered users to create new topics:
        1. Topic names have to be unique.
        2. The topic’s name is at most 30 characters
        3. The topic’s name can’t be empty
        4. 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. The title of a post can’t be empty.
        3. Posts should contain either a URL or a text content, **but not both**.
        4. If a topic gets deleted, all the posts associated with it should be automatically deleted too.
        5. 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. A comment’s text content can’t be empty.
        2. Contrary to the current linear comments, the new structure should allow comment threads at arbitrary levels.
        3. If a post gets deleted, all comments associated with it should be automatically deleted too.
        4. If the user who created the comment gets deleted, then the comment will remain, but it will become dissociated from that user.
        5. 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. Hint: you can store the (up/down) value of the vote as the values 1 and -1 respectively.
        2. If the user who cast a vote gets deleted, then all their votes will remain, but will become dissociated from the user.
        3. 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

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:

Create New Tables:

Create the tables in this order:

"users"

```
-- USERS ---
CREATE TABLE "users" (
    "id" SERIAL PRIMARY KEY,
    "username" VARCHAR(25) UNIQUE NOT NULL,
    "created_date" TIMESTAMP WITH TIME ZONE DEFAULT now(),
    "last_login" TIMESTAMP WITH TIME ZONE DEFAULT now(),
    -- the username must be 3 characters or longer --
    CONSTRAINT "username_length" CHECK (
        LENGTH(TRIM("username")) >= 3)
);
```

"topics"

```
-- TOPICS ---
CREATE TABLE "topics" (
    "id" SERIAL PRIMARY KEY,
    "topic_name" VARCHAR(30) UNIQUE NOT NULL,
    "description" VARCHAR(500) DEFAULT NULL,
    "created_date" TIMESTAMP WITH TIME ZONE DEFAULT now(),
    -- the topic_name must be 3 characters long --
    CONSTRAINT "topic_name_length" CHECK (
        LENGTH(TRIM("topic_name")) >= 3)
);
```

“posts”

```
-- POSTS ---
CREATE TABLE "posts" (
    "id" SERIAL PRIMARY KEY,
    "post_title" VARCHAR(100) NOT NULL,
    "url" VARCHAR(500) DEFAULT NULL,
    "text_content" TEXT DEFAULT NULL,
    "created_date" TIMESTAMP WITH TIME ZONE DEFAULT now(),
    "user_id" INTEGER,
    "topic_id" INTEGER,
    CONSTRAINT "url_or_text" CHECK (
        "url" IS NULL
        OR
        "text_content" IS NULL),
    FOREIGN KEY  ("user_id") REFERENCES "users" ("id") ON DELETE CASCADE,
    FOREIGN KEY  ("topic_id") REFERENCES "topics" ("id") ON DELETE CASCADE,
    -- the post_title must be 3 characters or longer --
    CONSTRAINT "post_title_length" CHECK (
        LENGTH(TRIM("post_title")) > 3)
);
```

“comments”

```
-- COMMENTS ---
CREATE TABLE "comments" (
    "id" SERIAL PRIMARY KEY,
    "text" VARCHAR(100) NOT NULL,
    "parent_comment_id" INTEGER DEFAULT NULL,
    "created_date" TIMESTAMP WITH TIME ZONE DEFAULT now(),
    "user_id" INTEGER,
    "post_id" INTEGER,
    FOREIGN KEY  ("parent_comment_id") REFERENCES "comments" ("id") ON
DELETE CASCADE,
    FOREIGN KEY  ("user_id") REFERENCES "users" ("id") ON DELETE CASCADE,
    FOREIGN KEY  ("post_id") REFERENCES "posts" ("id") ON DELETE CASCADE,
    -- comments must be 1 characters or longer - no spaces for comments --
    CONSTRAINT "post_title_length" CHECK (
        LENGTH(TRIM("text")) > 1)
);
```

“Votes”

```
-- VOTES ---
CREATE TABLE "votes" (
    "id" SERIAL PRIMARY KEY,
    "value" SMALLINT,
    "created_date" TIMESTAMP WITH TIME ZONE DEFAULT now(),
    "user_id" INTEGER,
    "post_id" INTEGER,
    FOREIGN KEY  ("user_id") REFERENCES "users" ("id") ON DELETE SET NULL,
    FOREIGN KEY  ("post_id") REFERENCES "posts" ("id") ON DELETE SET NULL,
    CONSTRAINT "votes_value" CHECK (
        ABS("value") = 1),
    CONSTRAINT "vote_once" UNIQUE ("id", "user_id" , "post_id")
);
```

Create Indexes:

```
--- CREATE INDEXES ---
CREATE INDEX "lower_username"
    ON "users" (LOWER("username"));

CREATE INDEX "lower_topic"
    ON "topics" (LOWER("topic_name"));

CREATE INDEX "search_topics_descriptions"
ON "topics" ("description" VARCHAR_PATTERN_OPS);

CREATE INDEX "url_search"
    ON "posts" (LOWER("url"));

CREATE INDEX "search_posts_text"
ON "posts" ("text_content" VARCHAR_PATTERN_OPS);

CREATE INDEX "top_level_comments"
    ON "comments" ("id",  "parent_comment_id");

CREATE INDEX "search_comments_text"
    ON "comments" ("text" VARCHAR_PATTERN_OPS);

CREATE INDEX "votes_post_id_value"
    ON "votes"( "post_id", "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 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.

Data Migration:

The data migration must be completed in order to ensure the foreign key constraints work properly. We start by normalizing the "bad_posts" table:

Normalize data

```
CREATE TABLE "normalized_bad_posts" (
    "id" SERIAL,
    "topic" VARCHAR(50),
    "username" VARCHAR(50),
    "title" VARCHAR(150),
    "url" VARCHAR(4000),
    "text_content" TEXT,
    "upvote" TEXT,
    "downvotes" TEXT
);
```

```
INSERT INTO "normalized_bad_posts" ("id",
    "topic",
    "username",
    "title",
    "url",
    "text_content",
    "upvote",
    "downvotes")

    SELECT
        "id",
        "topic",
        "username",
        "title",
        "url",
        "text_content",
        REGEXP_SPLIT_TO_TABLE("upvotes", ','),
        REGEXP_SPLIT_TO_TABLE("downvotes", ',')
    FROM "bad_posts";
```

Migrate Users

We are moving the users to a temporary table.

```
-- TEMP TABLE --
CREATE TABLE "temp_users" (
    "id" SERIAL PRIMARY KEY,
    "username" VARCHAR
);
```

We are capturing the users from both posts and comments.

```
INSERT INTO "temp_users" ("username")
    SELECT DISTINCT "username" FROM "bad_comments";

INSERT INTO "temp_users" ("username")
    SELECT DISTINCT "username" FROM "bad_posts";

INSERT INTO "users" ("username")
    SELECT DISTINCT "username" FROM "temp_users";

DROP TABLE "temp_users";
```

Migrate Topics

Check to see if there are any topics that violate the new schema constraint.

```
SELECT DISTINCT "topic" FROM "bad_posts" WHERE LENGTH("topic") < 3;
-- not topics with less than 3 characters
```

Migrate topics

```
INSERT INTO "topics" ("topic_name")
    SELECT DISTINCT "topic" FROM "normalized_bad_posts";
```

Migrate Posts

Some of the comments violate the length constraint on “text_content”:

```
-- ERROR ERROR:  value too long for type character varying(100)
-- SQL state: 22001
```

Migrate only the first 100 characters to the new “text_content” column.

```
INSERT INTO "posts" ("id", "user_id", "topic_id", "post_title", "url",
"text_content")
   (SELECT DISTINCT
            nbd."id",
            u.id "user_id",
            tp.id "topic_id",
            LEFT(nbd."title", 100),
            nbd."url",
            nbd."text_content"
   FROM "normalized_bad_posts" nbd
    JOIN "users" u
        ON u."username" = nbd."username"
    JOIN "topics" tp
        ON tp."topic_name" = nbd."topic");
```

Migrate Comments

There are comments referring to “post_id” that do not exist in the “posts” table (such posts were not present in the “bad_posts” table.)
If you try to migrate the data by joining the users only, you get an error:

```
-- ERROR:  insert or update on table "comments" violates foreign key constraint
"comments_post_id_fkey"
-- DETAIL:  Key (post_id)=(6044) is not present in table "posts".
-- SQL state: 23503
```

The correct method is:

```
INSERT INTO "comments" ("user_id", "post_id", "text")
   (SELECT  u.id "user_id",
        "post_id",
        LEFT(bd."text_content", 100)
    FROM bad_comments bd
    JOIN "users" u
        ON u."username" = bd."username"
    JOIN "posts" pt
        ON pt.id = "post_id");
```

Migrate Votes

We are going to migrate the votes into temporary tables, set the “value” to 1 or -1, then move them to the permanent table.

```
 -- TEMP TABLES --
CREATE TABLE "upvotes" (
    "user_id" INTEGER,
    "post_id" INTEGER,
    "value" SMALLINT DEFAULT 1);

CREATE TABLE "downvotes" (
    "user_id" INTEGER,
    "post_id" INTEGER,
    "value" SMALLINT DEFAULT -1);
```

Migrate to temporary tables

```
-- INSERT INTO TEMP TABLES
INSERT INTO "upvotes" ("post_id", "user_id")
    (
    SELECT nbp.id,
            u.id
    FROM "normalized_bad_posts" nbp
    JOIN "users" u
```

```
        ON u."username" = nbp."upvote"
    JOIN "posts" pt
        ON pt.id = nbp.id
    WHERE "upvote" IS NOT NULL
    );
INSERT INTO "downvotes" ("post_id", "user_id")
    (
    SELECT nbp.id,
            u.id
    FROM "normalized_bad_posts" nbp
    JOIN "users" u
        ON u."username" = nbp."downvotes"
    JOIN "posts" pt
        ON pt.id = nbp.id
    WHERE "downvotes" IS NOT NULL
    );
```

Move data from temp tables to the permanent “votes” table

```
INSERT INTO "votes" ( "user_id", "post_id", "value")
    (
        SELECT "user_id", "post_id","value"
        FROM "upvotes"
    );
INSERT INTO "votes" ( "user_id", "post_id", "value")
    (
        SELECT "user_id", "post_id","value"
        FROM "downvotes"
    );
```

Clean up

```
DROP TABLE "upvotes";
DROP TABLE "downvotes";
```