

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  
);
```

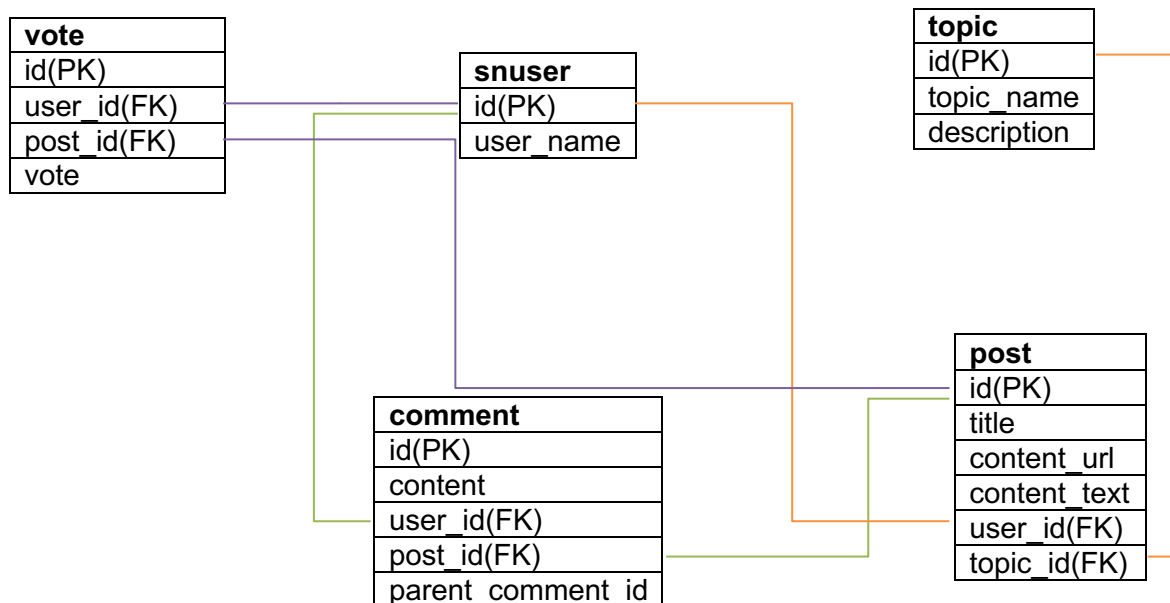
Part I: Investigate the existing schema

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!

Note: I already answer this part in my first submission, I don't know why you did not see it

1. The 'bad_db' is not normalized. It lacks foreign keys to connect two tables. The user_name column is not unique. There should be a "User" table to connect the posts and comments. Currently, the comments can exists without any posts.
2. The 'bad_posts' table contain both url columns and content columns. There can be duplicates if both has contents for one post.
3. 'upvotes' and 'downvotes' in the 'bad_posts' should not use "TEXT" as its type. It makes these two columns hard to count the total votes and it will allow duplicated votes from the same person for unlimited times.
4. in the 'bad_db', when a posts got deleted, the comments associated with it cannot be deleted at the same time.

New_db simplified **entity-relationship diagram** (ERD):



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 can't be empty
 - iv. We won't worry about user passwords for this project
 - b. Allow registered users to create new topics:
 - i. Topic names have to be unique.
 - ii. The topic's name is at most 30 characters
 - iii. The topic's name can't 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 can't be empty.
 - iii. Posts should contain either a URL or a text content, **but not both**.
 - iv. If a topic 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 comment's text content can't 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 don't 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 haven't logged in in the last year.
 - b. List all users who haven't created any post.
 - c. Find a user by their username.
 - d. List all topics that don't have any posts.
 - e. Find a topic by its name.
 - f. List the latest 20 posts for a given topic.
 - 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 don't 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: 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 snuser table ===
-- i.   Each username has to be unique
-- ii.  Usernames can be composed of at most 25 characters
-- iii. Usernames can't be empty

CREATE TABLE "snuser"
(
    id SERIAL PRIMARY KEY,
    user_name VARCHAR(25) NOT NULL,
    last_login TIMESTAMP
    CONSTRAINT "unique_user_name" UNIQUE ("user_name"),
    CONSTRAINT "no_empty_user_name" CHECK (LENGTH(TRIM("user_name")) > 0)
);

-- === Create topic table ===
-- i.   Topic names have to be unique.
-- ii.  The topic's name is at most 30 characters
-- iii. The topic's name can't be empty
-- iv.  Topics can have an optional description of at most 500 characters.

CREATE TABLE "topic"
(
    id SERIAL PRIMARY KEY,
    topic_name VARCHAR(30) NOT NULL,
    description VARCHAR(500),
    CONSTRAINT "unique_topic" UNIQUE ("topic_name"),
    CONSTRAINT "no_empty_topic_name" CHECK (LENGTH(TRIM("topic_name")) > 0)
);

-- === Create post table ===
-- i.   Posts have a required title of at most 100 characters
-- ii.  The title of a post can't be empty.
-- iii. Posts should contain either a URL or a text content, but not both.
-- iv.  If a topic 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.

CREATE TABLE "post"
(
    id SERIAL PRIMARY KEY,

```

```

title VARCHAR(100) NOT NULL,
content_url VARCHAR(400),
content_text TEXT,
created_on TIMESTAMPTZ,
topic_id INTEGER NOT NULL REFERENCES "topic" ON DELETE CASCADE,
user_id INTEGER REFERENCES "snuser" ON DELETE SET NULL,
CONSTRAINT "no_empty_title" CHECK (LENGTH(TRIM("title")) > 0),
CONSTRAINT "exclusive_url_text" CHECK (
    (LENGTH(TRIM("content_url")) > 0 AND LENGTH(TRIM("content_text")) = 0) OR
    (LENGTH(TRIM("content_url")) = 0 AND LENGTH(TRIM("content_text")) > 0)
);

-- === Create comment table ===
-- i. A comment's text content can't 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.

CREATE TABLE "comment"
(
    id SERIAL PRIMARY KEY,
    content TEXT NOT NULL,
    post_id INTEGER NOT NULL REFERENCES "post" ON DELETE CASCADE,
    user_id INTEGER REFERENCES "snuser" ON DELETE SET NULL,
    parent_comment_id INTEGER REFERENCES "comment" ON DELETE CASCADE,
    CONSTRAINT "no_empty_content" CHECK(LENGTH(TRIM("content")) > 0)
);

-- === Create vote table ===
-- 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.
-- A user can only cast one vote on a given post
CREATE TABLE "vote"
(

```

```
id SERIAL PRIMARY KEY,  
user_id INTEGER REFERENCES "snuser" ON DELETE SET NULL,  
post_id INTEGER NOT NULL REFERENCES "post" ON DELETE CASCADE,  
vote INTEGER NOT NULL,  
CONSTRAINT "vote_up_down" CHECK("vote" = 1 OR "vote" = -1),  
CONSTRAINT "one_vote_per_user_per_post" 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. 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.

Write the DML to migrate the current data in bad_posts and bad_comments to your new database schema:

```

-- Mitigate bad_db to new_db
-- 1. mitigate username from 'bad_posts' and 'bad_comments' to 'snuser' table --
'user_name' column

INSERT INTO "snuser" ("user_name")
  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;

-- 2. mitigate 'topic' from 'bad_posts' to 'topic' table -- 'topic_name' column

INSERT INTO "topic" ("topic_name")
SELECT DISTINCT topic FROM bad_posts;

-- 3. mitigate 5 columns from 'bad_posts' to 'post'

INSERT INTO "post" ("user_id", "topic_id", "title", "content_url", "content_text")
SELECT s.id, tp.id, LEFT(b.title, 100), b.url, b.text_content
FROM bad_posts b
JOIN snuser s ON b.username = s.user_name
JOIN topic tp ON b.topic = tp.topic_name;

-- 4. mitigate 3 columns from 'bad_comments' to 'post'
INSERT INTO "comment" ("post_id", "user_id", "content")
SELECT po.id, s.id, b.text_content
FROM bad_comments b
JOIN snuser s ON b.username = s.user_name
JOIN post po ON po.id = b.post_id;

-- 5. mitigate 3 columns into 'vote'

WITH upvote AS (SELECT id, REGEXP_SPLIT_TO_TABLE(upvotes, ',') upvoters
                FROM bad_posts)

INSERT INTO "vote" ("post_id", "user_id", "vote")
SELECT up.id, s.id, 1 vote_up
FROM upvote up

```



```

JOIN snuser s ON s.user_name=up.upvoters;

WITH downvote AS (SELECT id, REGEXP_SPLIT_TO_TABLE(downvotes,',') downvoters
                  FROM bad_posts)



INSERT INTO "vote" ("post_id","user_id", "vote")
SELECT d.id, s.id, -1 AS vote_down
FROM downvote d
JOIN snuser s ON s.user_name=d.downvoters;

-- Create index for fast queries
(DELETE INDEXING FOR user_name and topic_name due to unique constraint)
CREATE INDEX ON "snuser" ("user_name" VARCHAR_PATTERN_OPS);
CREATE INDEX ON "topic" ("topic_name" VARCHAR_PATTERN_OPS);
CREATE INDEX ON "post" ("title" VARCHAR_PATTERN_OPS);




```

SAMPLE Tables:

snuser_table

	id [PK] integer 	user_name character varying (25) 
1	1	Aaliyah.Kilback
2	2	Aaliyah.Waelchi
3	3	Aaliyah3
4	4	Aaliyah_Spencer25
5	5	Aaron.Mills81
6	6	Aaron18
7	7	Aaron86
8	8	Aaron94
9	9	Aaron_Tremblay
10	10	Abagail20

topic_table

	id [PK] integer 	topic_name character varying (30) 	description character varying (500) 
1	1	navigate	[null]
2	2	Boliviano_Mvdol	[null]
3	3	Vermont	[null]
4	4	Bedfordshire	[null]
5	5	Representative	[null]
6	6	Run	[null]
7	7	attitude-oriented	[null]
8	8	parsing	[null]
9	9	Synergized	[null]
10	10	connecting	[null]





post_table





	id [PK] integer	content text	post_id integer	user_id integer	parent_comment_id integer
1	1	Voluptatem cum nisi maxim...	2615	6881	[null]
2	2	Ab ea ad velit tempore. Cons...	6755	106	[null]
3	3	Atque quaerat et. Omnis con...	9102	2541	[null]
4	4	Ratione facilis et beatae aut ...	9734	6399	[null]
5	5	Veritatis doloribus officiis cu...	890	5224	[null]
6	6	Perspiciatis eum voluptas m...	3345	1818	[null]
7	7	Impedit alias magnam mollit...	6045	10140	[null]
8	8	Id veniam magnam corrupti. ...	4108	10747	[null]
9	9	Cumque autem magnam iste...	5019	3291	[null]
10	10	Sed impedit repellendus fugi...	578	10596	[null]

comment_table

	id [PK] integer	content text	post_id integer	user_id integer	parent_comment_id integer
1	1	Voluptatem cum nisi maxim...	2615	6881	[null]
2	2	Ab ea ad velit tempore. Cons...	6755	106	[null]
3	3	Atque quaerat et. Omnis con...	9102	2541	[null]
4	4	Ratione facilis et beatae aut ...	9734	6399	[null]
5	5	Veritatis doloribus officiis cu...	890	5224	[null]
6	6	Perspiciatis eum voluptas m...	3345	1818	[null]
7	7	Impedit alias magnam mollit...	6045	10140	[null]
8	8	Id veniam magnam corrupti. ...	4108	10747	[null]
9	9	Cumque autem magnam iste...	5019	3291	[null]
10	10	Sed impedit repellendus fugi...	578	10596	[null]

vote_table

	id [PK] integer 	user_id integer 	post_id integer 	vote integer 
1	1	5774	1	1
2	2	2586	1	1
3	3	7709	1	1
4	4	2470	1	1
5	5	1315	1	1
6	6	5298	1	1
7	7	6056	1	1
8	8	6241	1	1
9	9	7425	2	1
10	10	550	2	1

	id [PK] integer 	user_id integer 	post_id integer 	vote integer 
1	249800	6507	1	-1
2	249801	5712	1	-1
3	249802	5497	1	-1
4	249803	704	2	-1
5	249804	2746	2	-1
6	249805	7861	2	-1
7	249806	9462	2	-1
8	249807	7605	2	-1
9	249808	2470	2	-1
10	249809	217	2	-1