Udiddit, a social news aggregator

Rayner Mendez

## 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!

|  |
| --- |
| 1st: Normalization is a urgent requirement here. I will create a post, user, topic, vote, and comment tables. This can change later on.  2nd: I will inspect the tables and create proper relations between the tables.  3rd: I will index most frequent accessed data from the tables. |

## 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:

|  |
| --- |
| DROP TABLE IF EXISTS  users,  topic,  post,  comment,  vote;  CREATE TABLE users (  id BIGSERIAL PRIMARY KEY,  username VARCHAR(25) UNIQUE NOT NULL,  last\_login TIMESTAMP NOT NULL DEFAULT *CURRENT\_TIMESTAMP*,  CONSTRAINT username\_length  CHECK (*char\_length*(username) > 0) );  CREATE TABLE topic (  id SERIAL PRIMARY KEY,  topic VARCHAR(30) UNIQUE NOT NULL,  description VARCHAR(500),  CONSTRAINT topic\_length  CHECK (*char\_length*(topic) > 0) );  CREATE TABLE post (  id BIGSERIAL PRIMARY KEY,  title VARCHAR(100) NOT NULL,  url VARCHAR(2048),  content TEXT,  created\_on TIMESTAMP NOT NULL DEFAULT *CURRENT\_TIMESTAMP*,  user\_id BIGINT NOT NULL,  topic\_id INTEGER,  CONSTRAINT fk\_user\_id  FOREIGN KEY (user\_id)  REFERENCES users ON DELETE SET NULL,  CONSTRAINT fk\_topic\_id  FOREIGN KEY (topic\_id)  REFERENCES topic ON DELETE CASCADE,  CONSTRAINT url\_or\_content  CHECK ((post.url IS NOT NULL AND post.content IS NULL)  OR (post.content IS NOT NULL AND post.url IS NULL)),  CONSTRAINT title\_length  CHECK (*char\_length*(title) > 0) );  CREATE INDEX ON post (created\_on);  CREATE TABLE vote (  id BIGSERIAL PRIMARY KEY,  upvote BOOLEAN,  downvote BOOLEAN,  created\_on TIMESTAMP NOT NULL DEFAULT *CURRENT\_TIMESTAMP*,  user\_id BIGINT,  post\_id BIGINT NOT NULL,  CONSTRAINT fk\_user\_id  FOREIGN KEY (user\_id)  REFERENCES users ON DELETE SET NULL,  CONSTRAINT fk\_post\_id  FOREIGN KEY (post\_id)  REFERENCES post ON DELETE CASCADE,  CONSTRAINT upvote\_or\_downvote  CHECK ((vote.upvote IS NOT NULL AND vote.downvote IS NULL)  OR (vote.downvote IS NOT NULL AND vote.upvote IS NULL)) );  CREATE INDEX ON vote (created\_on);  CREATE TABLE comment (  id BIGSERIAL PRIMARY KEY,  content TEXT NOT NULL,  created\_on TIMESTAMP NOT NULL DEFAULT *CURRENT\_TIMESTAMP*,  user\_id BIGINT,  post\_id BIGINT NOT NULL,  comment\_id BIGINT,  CONSTRAINT fk\_user\_id  FOREIGN KEY (user\_id)  REFERENCES users ON DELETE SET NULL,  CONSTRAINT fk\_post\_id  FOREIGN KEY (post\_id)  REFERENCES post ON DELETE CASCADE,  CONSTRAINT fk\_comment\_id  FOREIGN KEY (comment\_id)  REFERENCES comment ON DELETE CASCADE,  CONSTRAINT content\_length  CHECK (*char\_length*(content) > 0) );  CREATE INDEX ON comment (created\_on); |

## 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:

|  |
| --- |
| -- INSERT all usernames INTO users table INSERT INTO users (username) SELECT DISTINCT(username) FROM bad\_posts ORDER BY username ON CONFLICT DO NOTHING;  INSERT INTO users (username) SELECT DISTINCT(username) FROM bad\_comments ORDER BY username ON CONFLICT DO NOTHING;  INSERT INTO users (username) SELECT DISTINCT(*REGEXP\_SPLIT\_TO\_TABLE*(upvotes, ',')) AS username FROM bad\_posts ORDER BY username ON CONFLICT DO NOTHING;  INSERT INTO users (username) SELECT DISTINCT(*REGEXP\_SPLIT\_TO\_TABLE*(downvotes, ',')) AS username FROM bad\_posts ORDER BY username ON CONFLICT DO NOTHING;   -- INSERT all topics INTO topic table INSERT INTO topic (topic) SELECT DISTINCT(topic) FROM bad\_posts ORDER BY topic;   -- INSERT title, user\_id, topic\_id, url, content INTO posts INSERT INTO post (id, title, user\_id, topic\_id, url, content) SELECT bp.id,  *SUBSTR*(bp.title, 1, 100) AS title,  u.id AS user\_id,  t.id AS topic\_id,  bp.url AS url,  bp.text\_content AS content FROM bad\_posts bp JOIN users u  ON bp.username = u.username JOIN topic t  ON bp.topic = t.topic ON CONFLICT DO NOTHING;  -- INSERT comments INTO comments table INSERT INTO comment (id, content, user\_id, post\_id) SELECT bc.id, bc.text\_content, u.id, bc.post\_id FROM bad\_comments bc JOIN users u ON bc.username = u.username ON CONFLICT DO NOTHING;  -- INSERT upvotes INTO vote table INSERT INTO vote (upvote, user\_id, post\_id) WITH upvote AS (  SELECT id as post\_id, *REGEXP\_SPLIT\_TO\_TABLE*(upvotes, ',') AS voter FROM bad\_posts ) SELECT true, u.id, uv.post\_id FROM upvote uv JOIN users u ON uv.voter = u.username ON CONFLICT DO NOTHING;  -- INSERT downvote INTO vote table INSERT INTO vote (downvote, user\_id, post\_id) WITH downvote AS (  SELECT id as post\_id, *REGEXP\_SPLIT\_TO\_TABLE*(downvotes, ',') AS voter FROM bad\_posts ) SELECT true, u.id, dv.post\_id FROM downvote dv JOIN users u ON dv.voter = u.username ON CONFLICT DO NOTHING; |