Udiddit, a social news aggregator

-Manish Taori

## 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!

|  |
| --- |
| 1. The table might make better use of constraints, such as the UNIQUE constraint to prevent repeated entries within a column, the CHECK constraint to apply any business rules, and, most crucially, the FOREIGN KEY constraint to enable relationships between related tables. 2. We could improve the schema by changing the columns "upvotes" and "downvotes" to NUMERIC, which can have a value of 1 and 0 respectively. 3. No indexing is done on queries which will slow the query read operation. 4. Username column in both tables should have constraint “Not Null” 5. Foreign key can be added in post\_id(bad\_comment) column from bad\_post table. |

## 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:

|  |
| --- |
| ---------------------------------------- --user table craeted ---------------------------------------- CREATE TABLE users(  user\_id SERIAL PRIMARY KEY,  username VARCHAR(25) UNIQUE NOT NULL,  login\_timestamp TIMESTAMP  CONSTRAINT "username\_lenght\_not\_zero" CHECK (Length(TRIM("username")) > 0) );  --Index for a list of all users who haven't logged in in the last year. CREATE INDEX log\_in\_index ON users (login\_timestamp);  -- Index for finding a user by their username. CREATE INDEX username\_index ON users (username);  ---------------------------------------- --Topic table created ---------------------------------------- CREATE TABLE topics(  topic\_id SERIAL PRIMARY KEY,  topic\_name VARCHAR(30) CONSTRAINT unique\_topic\_required UNIQUE NOT NULL,  topic\_description VARCHAR(500)  CONSTRAINT "topics\_length\_not\_zero" CHECK (Length(Trim("topic\_name")) > 0) ); --Index on topic\_name CREATE INDEX topic\_index ON topics(topic\_name);  ---------------------------------------- --posts table created ---------------------------------------- CREATE TABLE posts (  post\_id SERIAL PRIMARY KEY,  topic\_id INTEGER REFERENCES topics ON DELETE CASCADE NOT NULL,  user\_id INTEGER REFERENCES users ON DELETE SET NULL,  title VARCHAR(100) NOT NULL,  post\_timestamp TIMESTAMP,  url TEXT DEFAULT NULL,  text\_content TEXT DEFAULT NULL,  CONSTRAINT either\_url\_or\_text   CHECK(url IS NOT NULL AND text\_content IS NULL OR  url IS NULL AND text\_content IS NOT NULL)  CONSTRAINT "posts\_length\_not\_zero" CHECK (Length(Trim("title")) > 0), );   -- Index for a list of latest posts for a given user CREATE INDEX latest\_posts\_per\_user ON posts (topic\_id,user\_id);  --Index for a list of latest posts for a given topic CREATE INDEX latest\_posts\_per\_topic ON posts (topic\_id,post\_timestamp);  -- Index to find posts with URL CREATE INDEX post\_with\_url ON posts (url);  ---------------------------------------- --comments table created ---------------------------------------- CREATE TABLE comments (  comment\_id SERIAL PRIMARY KEY,  topic\_id INTEGER REFERENCES topics ON DELETE CASCADE NOT NULL,  post\_id INTEGER REFERENCES posts ON DELETE CASCADE NOT NULL,  user\_id INTEGER REFERENCES users ON DELETE SET NULL,  comment\_timestamp TIMESTAMP,  text\_content TEXT CONSTRAINT text\_content\_required NOT NULL,  level INTEGER REFERENCES comments ON DELETE CASCADE  CONSTRAINT "posts\_length\_not\_zero" CHECK (Length(Trim("text\_content")) > 0 ), );   --index to list the latest 20 comments made by a given user CREATE INDEX comments\_by\_user ON comments (user\_id,comment\_timestamp);  --index to list all the top-level comments for a given post CREATE INDEX level\_index ON comments (level);  --index to list all the direct children of a parent comment CREATE INDEX parent\_id\_index ON comments (post\_id);   ---------------------------------------- --Votes table created ---------------------------------------- CREATE TABLE votes (  vote\_id SERIAL PRIMARY KEY,  vote INTEGER CONSTRAINT vote\_up\_down CHECK(vote=1 OR vote=-1),  user\_id INTEGER REFERENCES users ON DELETE SET NULL,  post\_id INTEGER REFERENCES posts ON DELETE CASCADE NOT NULL ); -- Index to find score of post. CREATE INDEX score\_of\_post ON votes (vote); |

## 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:

|  |
| --- |
| --------------------------------- --inserting data in USERS --------------------------------- -- users who have only commented INSERT INTO users (username) with union\_users AS  (SELECT b.username  FROM users u  RIGHT JOIN bad\_comments b  ON b.username = u.username  WHERE u.username IS NULL  union --users who have post  SELECT username  FROM bad\_posts) SELECT DISTINCT username FROM union\_users   -- users who have voted up INSERT INTO users (username)  WITH upvote\_table AS   (SELECT REGEXP\_SPLIT\_TO\_TABLE(upvotes, ',')   AS up  FROM bad\_posts)  SELECT DISTINCT up  FROM users u  RIGHT JOIN upvote\_table t  ON t.up = u.username  WHERE u.username IS NULL;  -- users who have voted down INSERT INTO users (username)  WITH downvote\_table AS   (SELECT REGEXP\_SPLIT\_TO\_TABLE(downvotes, ',')  AS down  FROM bad\_posts)  SELECT down  FROM users u  RIGHT JOIN downvote\_table t  ON t.down = u.username  WHERE u.username IS NULL;  --------------------------------- --inserting data in TOPICS --------------------------------- INSERT INTO topics (topic\_name)  SELECT DISTINCT topic FROM bad\_posts;  --------------------------------- --inserting data in POSTS --------------------------------- INSERT INTO posts (post\_id,   topic\_id,  user\_id,  title,   url,   text\_content)  SELECT b.id, t.topic\_id, u.user\_id,   LEFT(b.title,100), b.url, b.text\_content  FROM bad\_posts b  JOIN topics t  ON t.topic\_name = b.topic  JOIN users u  ON u.username = b.username;  --------------------------------- --inserting data in COMMENTS --------------------------------- INSERT INTO comments (user\_id,   post\_id,   text\_content,   level)  SELECT u.user\_id, p.post\_id, b.text\_content,   ROW\_NUMBER() OVER(PARTITION BY p.post\_id)  FROM bad\_comments b  JOIN posts p  ON p.post\_id = b.post\_id   JOIN users u  ON u.username = b.username ;  --------------------------------- --inserting data in VOTES --------------------------------- --upvotes INSERT INTO votes (user\_id,   post\_id,   vote)  WITH upvotes\_table AS   (SELECT id, REGEXP\_SPLIT\_TO\_TABLE(downvotes, ',')   AS downvote  FROM bad\_posts)  SELECT u.user\_id, t.id, -1 AS vote  FROM upvotes\_table t  JOIN users u  ON t.downvote = u.username;  --downvotes INSERT INTO votes (user\_id,  post\_id,   vote)  WITH downvote\_table AS   (SELECT id, REGEXP\_SPLIT\_TO\_TABLE(upvotes, ',')   AS upvote  FROM bad\_posts)  SELECT u.user\_id, t.id, 1 AS vote  FROM downvote\_table t  JOIN users u  ON t.upvote = u.username ; |