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!

|  |
| --- |
| A few things that could be improved about the schema.   1. Normalize table “bad\_comments” by migrating username column out of the table and create another table called users that has fields “userid” primary key and user name. Therefore insert a new column called userid within “bad\_comments” that is a foreign key referencing table “users”. 2. post\_id within “bad\_comments” should be a foreign key reference to table “bad\_posts”. 3. Normalize “bad\_posts” by removing username column, and adding userid foreignkey referencing “users” table. 4. Within table “bad\_posts” the field topic could also be a foreignkey field referencing a separate new table called “topics” with primary key id and “topic name” 5. Both tables are not normalized. 6. There are no constraints to safeguard against compromising integrity of data. |

## 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:

|  |
| --- |
| *--TABLE#1*  CREATE TABLE "users"(  id SERIAL,  name VARCHAR(50),  PRIMARY KEY (id),   CONSTRAINT usernamelen CHECK(length("name") > 0 AND length("name") <=25),   CONSTRAINT uniqname UNIQUE("name")  );  CREATE INDEX "userindx" ON "users" ("id");  *--TABLE#2*  CREATE TABLE "topics"(  topicID SERIAL,  name VARCHAR(30),  description VARCHAR(500),  PRIMARY KEY(topicID),  CONSTRAINT uniqtopic UNIQUE("name"),  CONSTRAINT topiclen CHECK(length("name") > 0)  );  CREATE INDEX "topicsname" ON "topics" ("name");  *--TABLE#3*  CREATE TABLE "posts"(  id INTEGER,  title VARCHAR(150),  topicid INTEGER,  userid INTEGER,  text\_content TEXT,  url VARCHAR(4000),  posttime  DATE,  CONSTRAINT topicnotnull CHECK(topicid  <> NULL),  CONSTRAINT postsprime PRIMARY KEY(id),  CONSTRAINT fk\_topic FOREIGN KEY(topicid) REFERENCES "topics" ON DELETE CASCADE,  CONSTRAINT fk\_user  FOREIGN KEY(userid) REFERENCES "users" ON DELETE SET NULL,  CONSTRAINT nonemptytopic CHECK(length(title) > 0),  CONSTRAINT urlortxt  CHECK( (length(url) > 0  AND  length(text\_content) = 0) OR (length(url) = 0  AND  length(text\_content) > 0) )  );  CREATE INDEX "postindx" ON "posts" ("title");  *--TABLE#4*  CREATE TABLE "comments"(  id SERIAL,  textContent TEXT,  userid INTEGER,  topicid INTEGER,  postid INTEGER,  parentid INTEGER,  commenttime  DATE,  CONSTRAINT  commentsnonempty CHECK(LENGTH(TRIM(textContent)) > 0),  CONSTRAINT commentsprime PRIMARY KEY(ID),  CONSTRAINT fktable\_user FOREIGN KEY(userid) REFERENCES "users" ON DELETE SET NULL,  CONSTRAINT fktable\_topic FOREIGN KEY(topicid) REFERENCES "topics"  );  CREATE INDEX "commentindx" ON "comments" ("id");    *--TABLE#5*  CREATE TABLE "votes"(  userid integer,  postid  integer,  vote smallint,  votetime DATE,  CONSTRAINT postnotnull CHECK(postid <> NULL,  CONSTRAINT votesprimary PRIMARY KEY (userid, postid),  CONSTRAINT fkvotesuser FOREIGN KEY(userid) REFERENCES "users" ON DELETE SET NULL,  CONSTRAINT fkvotespost FOREIGN KEY(postid) REFERENCES "posts" ON DELETE CASCADE,  CONSTRAINT limitvoteval CHECK( vote = 1 OR vote = -1)  );  CREATE INDEX "votesindx" ON "votes" ("userid"); |

## 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. The order of your migrations matter! For example, since posts depend on **users** and **topics**, you’ll have to migrate the latter first.✔
3. 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
4. You can use the Postgres string function **regexp\_split\_to\_table** to unwind the comma-separated votes values into separate rows
5. Don’t forget that some users only vote or comment, and haven’t created any posts. You’ll have to create those users too.
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:

|  |
| --- |
| *--TABLE "users" 1*  INSERT INTO "users"("name") SELECT DISTINCT(bad\_comments.username) FROM "bad\_comments";  INSERT INTO "users"("name") SELECT DISTINCT(bad\_posts.username)  FROM "bad\_posts"   WHERE username NOT IN (SELECT "username" FROM bad\_comments);  CREATE VIEW upusers AS SELECT REGEXP\_SPLIT\_TO\_TABLE(upvotes, ',') AS username FROM bad\_posts;  CREATE VIEW downusers AS SELECT REGEXP\_SPLIT\_TO\_TABLE(downvotes, ',') AS username FROM bad\_posts;    *--testing only*  *--SELECT DISTINCT("username") FROM downusers  WHERE "username" NOT IN (SELECT upusers.username  FROM upusers) limit 3;*  INSERT INTO "users"("name") SELECT  DISTINCT("username") FROM upusers  WHERE "username" NOT IN (SELECT "name" FROM users);  INSERT INTO "users"("name") SELECT DISTINCT("username") FROM downusers  WHERE "username" NOT IN (SELECT "name" FROM users);   DROP VIEW  upusers;   DROP VIEW downusers;        *--TABLE "topics" 2*  INSERT INTO "topics" ("name") SELECT DISTINCT("topic") FROM "bad\_posts";    *--TABLE "posts" 3*  INSERT INTO "posts" ("id","title", "text\_content", "url", "userid", topicid)  SELECT bad\_posts.id,"title", "text\_content", "url", users.id, topics.topicid  FROM "bad\_posts"  JOIN users ON users.name = bad\_posts.username  JOIN topics ON topics.name = bad\_posts.topic;    *--TABLE "comments" 4*   INSERT INTO "comments" ("id", "textcontent", "postid", "parentid", "userid","topicid")   SELECT  bad\_comments.id, bad\_comments.text\_content, bad\_comments.post\_id, -1, users.id, posts.topicid   FROM  bad\_comments JOIN "users" ON bad\_comments.username = users.name                      JOIN "posts" ON  bad\_comments.post\_id = posts.id;    *--TABLE "votes" 5*  CREATE VIEW DOWNVOTERS AS SELECT bad\_posts.id as postid, users.id as authorID,  REGEXP\_SPLIT\_TO\_TABLE(downvotes, ',') as DOWNVoter  FROM bad\_posts JOIN users ON users.name = bad\_posts.username   ;  CREATE VIEW UPVOTERS  AS SELECT bad\_posts.id as postid, users.id as authorID,   REGEXP\_SPLIT\_TO\_TABLE(upvotes, ',') as UPVoter  FROM bad\_posts JOIN users ON users.name = bad\_posts.username   ;    INSERT INTO "votes" ("postid", "userid", "vote")  SELECT downvoters.postid, users.id , -1  FROM  downvoters  join users on downvoters.downvoter = users.name;  INSERT INTO "votes" ("postid", "userid", "vote")  SELECT upvoters.postid, users.id , 1  FROM  upvoters  join users on upvoters.upvoter = users.name;  DROP VIEW DOWNVOTERS;  DROP VIEW UPVOTERS; |
|  |

## Requires Changes

### 3 specifications require changes

Dear Student,

Congratulations on completing all the requirements of the Udiddit, A Social News Aggregator (Part I) project, and earning a new milestone.

Amazing work in this submission! ✌ I was truly happy to see the results of all your learning come so clearly into your final project. You were really good at showcasing your SQL skills.✌

Below mentioned are the only minimal change that is required in your project. Fix for the below-mentioned one will drive you towards the success path.

1. Add the CHECK constraints CHECK(LENGTH(TRIM(**<column name>**))>0)✔
   * A comment’s text content can’t be empty.✔
2. Import missing users from upvotes and downvotes columns. )✔
3. But you missed to import users from upvotes and downvotes column. Kindly import the same to match the counts as expected. )✔
4. Add the indexes as mentioned in the rubrics. ✔
   * The data model provided should have appropriate indexes in order for the queries in guidelines 1 and 2 to run quickly and efficiently. Note as some constraints automatically add indexes, it’s important to make sure that you don’t duplicate any indexes. ✔
   * You missed creating the INDEXES as mentioned in the rubrics✔

Remember to check out these, for some tips on database design:

* [10 best database design practices](https://medium.com/quick-code/10-best-database-design-practices-1f10f3441730)
* [Database Design Patterns](https://stackoverflow.com/questions/44281/database-patterns) - **A StackOverFlow Post**
* [Primary Key vs Unique Key](https://www.geeksforgeeks.org/difference-between-primary-key-and-unique-key/)

I wish you all the best for the upcoming challenges.✌

## Requires Changes - 2nd time:

Apply syntax highlighting to DDL and DML queries ✔ ✔

Add important auditing fields (like a timestamp field) ✔✔

which are required to provide answer to some of the queries

Important auditing fields are missing in some tables where are required to answer

all the features and queries outlined in the document.

For example, you need to add an additional field

with some TIMESTAMP type so as to answer the following queries : ✔✔

2.a - List all users who haven’t logged-in in the last year. ✔✔

2.b - List all users who haven’t created any post. ✔✔

DATETIME

Set NOT NULL constraint for all the foreign keys that don't have the constraint ON DELETE SET NULL ✔✔

posts

CONSTRAINT fk\_topic FOREIGN KEY(topicid) REFERENCES "topics" ON DELETE CASCADE, ✔

votes

CONSTRAINT fkvotespost FOREIGN KEY(postid) REFERENCES "posts" ON DELETE CASCADE, ✔