Final Research Project for Udacity :

Restructuring a Relational Database Schema (RDBMS) for a live Social Media / Social News Aggregator named ‘Udiddit’.

Research was conducted by: Frederick Zoreta

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!
RESPONSES & OBSERVATIONS: Issues regarding the existing {Schema}

1. There appears to be vital information that is missing. The actual, specific user that started a specific subject matter / topic is not given. I think it should be given. I've decided that I should be adding a {NULL_Value} for the Foreign_Key (FK) in the upcoming table. This is the "topics" table and would be acting as the FK for "Users" table. This simply means that 'topic's would be referencing 'users'.

2. Two tables are simply not enough to have a good schema with the vast amount of incoming data, not to mention a live, ongoing source of tweets.

3. The structure is disorganized and could lead to overloading the database.

4. Related to number 3, there seems to be no limit/restraint/constraint on permitting a user to keep on voting on unlimited basis. This may lead to the entire storage space being over-utilized.

5. Related to number 4, this may pose as a Cyber-Security risk. Although I am not a specialist on cyber security, if there is almost an unlimited amount of permission regarding posts , an intruder could do a 'SQL-injection' type of attack. This may even lead to other forms of data inconsistencies.

6. In the 'bad_posts' table, the 'URL' coulmn has a maximum of 4000 characters. This length is definitely not viable nor makes nay sense for a certain URL length. This definitely uses a lot of disk space.

7. The 'text_content' field in the 'bad_comments' table could also be a VARCHAR. Although text also makes sense, some contents could have numerical values

8. Lack of Foreign_Key constraints! This is very vital in the over-all functionality of a Relational Database Management System (RDBMS).

9. There are certain numerical types that could have been used to save disk space. BIGINT or even INT could be used to contain negative values. Should there be an instance that an auto increment occurs, this would cause a major problem in the database.

10. Within both tables, the 'TEXT_content' value, it would be much better and more accurate to utilize VARCHAR.

11. Quite noticeable, both the 'DOWNVOTES' and 'UPVOTES' appears to have a wrong data type. It is NOT recommended to have a text data type.

12. I would highly suggest a 'user registration' option. Any database should have such very important option, especially for a social media database. This is not just strategically important on a data and technology level, but also on a marketing and PR aspects.

13. I would limit the user_name to maximum 25. Although most DBAs/ analysts would claim that 50 is still reasonable, in reality

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.

    I have decided that it is better not to add any specific index for this initial query. It may potentially/eventually slow down the querying process. SQL indexes with a condition that contains an expression would add overhead to the entire RDBMS.

    b. List all users who haven't created any post.
    Within the 'posts' table, there's a Foreign_Key (FK) named "author_id". I've added an index for this FK.

    c. Find a user by their username.
    A very simple query like this:

```
SELECT *
FROM users
WHERE username = 'Abbey.Gaylord5'
```

    Would simply give you the desired results.
    **the unique index being placed on the 'username' field easily handles this.

    d. List all topics that don't have any posts.
    Another simple query like this:

```
SELECT  T.name
FROM    topics T
        LEFT JOIN
        posts P
        ON T.id = P.topic_id
WHERE   P.topic_id IS NULL;
```

**I've placed an index on the foreign_key (FK) 'topic_id' within the 'posts' table. This specific FK does references the 'topics' table.

e. Find a topic by its name.

```
SELECT *
FROM topics
WHERE name = 'client-server';

OR

SELECT *
FROM topics
WHERE name = 'engineer';
```

**I've used the 'varchar_pattern_ops' for the 'name' field in the 'topics' table.

f. List the latest 20 posts for a given topic.
I've built a unique index on the 'username' field within the 'users' table. Also, pls take note that the foreign_key(FK) in the 'posts' table does reference the 'users' table.

Below is the query that handles this issue:

```
SELECT  T.name,
        P.url,
        P.title,
        u.username AS Author,

        P.text_content
FROM    topics T
        INNER JOIN
        posts p
        ON t.id = p.topic_id
        INNER JOIN
        users U
        ON U.id = P.author_id
WHERE   T.name = 'engineer'
ORDER BY  p.date_created DESC
LIMIT 20;
```

g. List the latest 20 posts made by a given user.
In the 'users' table, there is a 'username' field that has a Unique Index.

There is also an index on the foreign_key in the 'posts' table, and this references the 'users' table.

The query below handles this issue:

```
SELECT  p.title,
    u.username AS author,
    p.url,
    p.text_content
FROM   posts p
    INNER JOIN
    users u
    ON u.id = p.author_id AND u.username = 'Abbey.Gaylord5'
ORDER BY p.date_created DESC
LIMIT 20;
```

h. Find all posts that link to a specific URL, for moderation purposes.
I wrote a specific index on the 'url' field of the 'posts' table.
Here's the query:

```
SELECT p.id,
    p.title,
    u.username AS author,
    u.id AS user_id,
    p.url
FROM   posts p
    INNER JOIN
    users u
    ON u.id = p.author_id  AND  p.url = 'http://woodrow.name'
ORDER BY p.id ASC;
```

i. List all the top-level comments (those that don't have a parent comment) for a given post.
I have placed an index on the 'parent_id' of the 'comment's table.

j. List all the direct children of a parent comment.
The Primary_Key within the 'comments' table has a unique index.

k. List the latest 20 comments made by a given user.
Within the 'comment's table, I've created an index within the primary key(author_id), which in turn references the 'users' table.

l. Compute the score of a post, defined as the difference between the number of upvotes and the number of downvotes

This is the query I've used:

```
SELECT  P.title,
        SUM(UV.vote) AS Post_Scores
FROM    posts P
        INNER JOIN
        user_votes UV
        ON P.id = UV.post_id
GROUP BY P.title
ORDER BY 2 DESC, 1 ASC;
```

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.

Below is the Normalized Schema of the new relational database:

Table_Name: Users
@id (PK),
@username,
@date_created,
@created_by,
@last_login,
@date_updated,
@updated_by

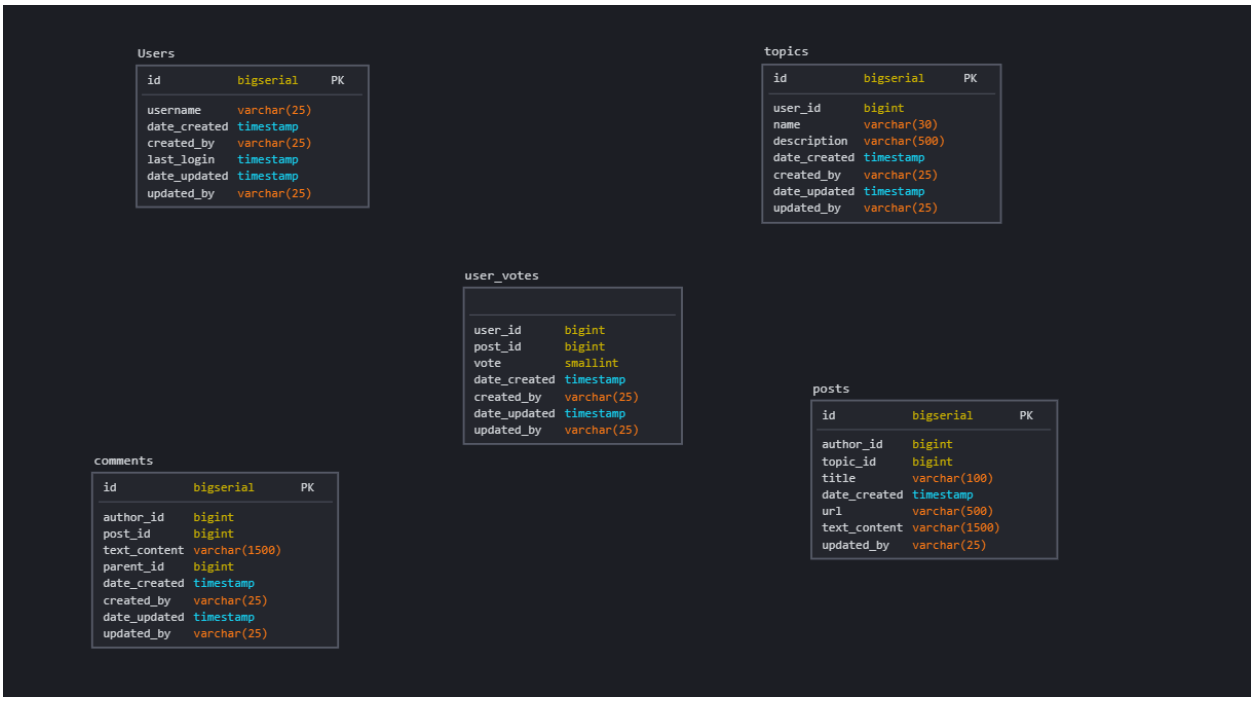
Table_Name: Topics
@id (PK),
@user_id,
@name,
@desciption,
@date_created,
@created_by,
@date_updated,
@updated_by

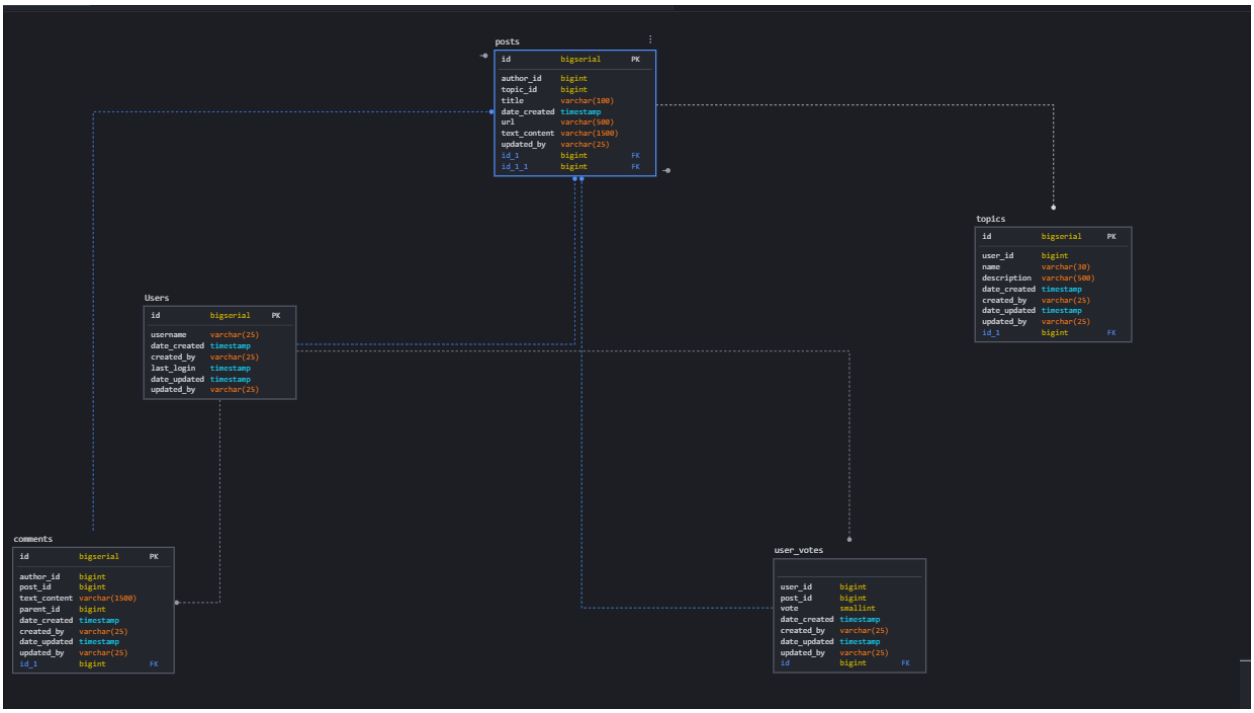
Table_Name: Posts
@id (PK),
@author_id,
@topic_id,
@title,
@date_created,
@created_by,
@date_updated,
@url,
@text_content,
@updated_by

Table_Name: Comments
@id (PK),
@author_id,
@post_id,
@text_content,
@parent_id,
@created_by,
@date_created,
@date_updated

Table_Name: User_Votes
@user_id,
@post_id,
@vote,
@created_by,
@date_updated,
@updated_by

Below are 2 images using the Normalized Schema. First are just the entities/tables, and the 2nd involves the entity connections/relationships.



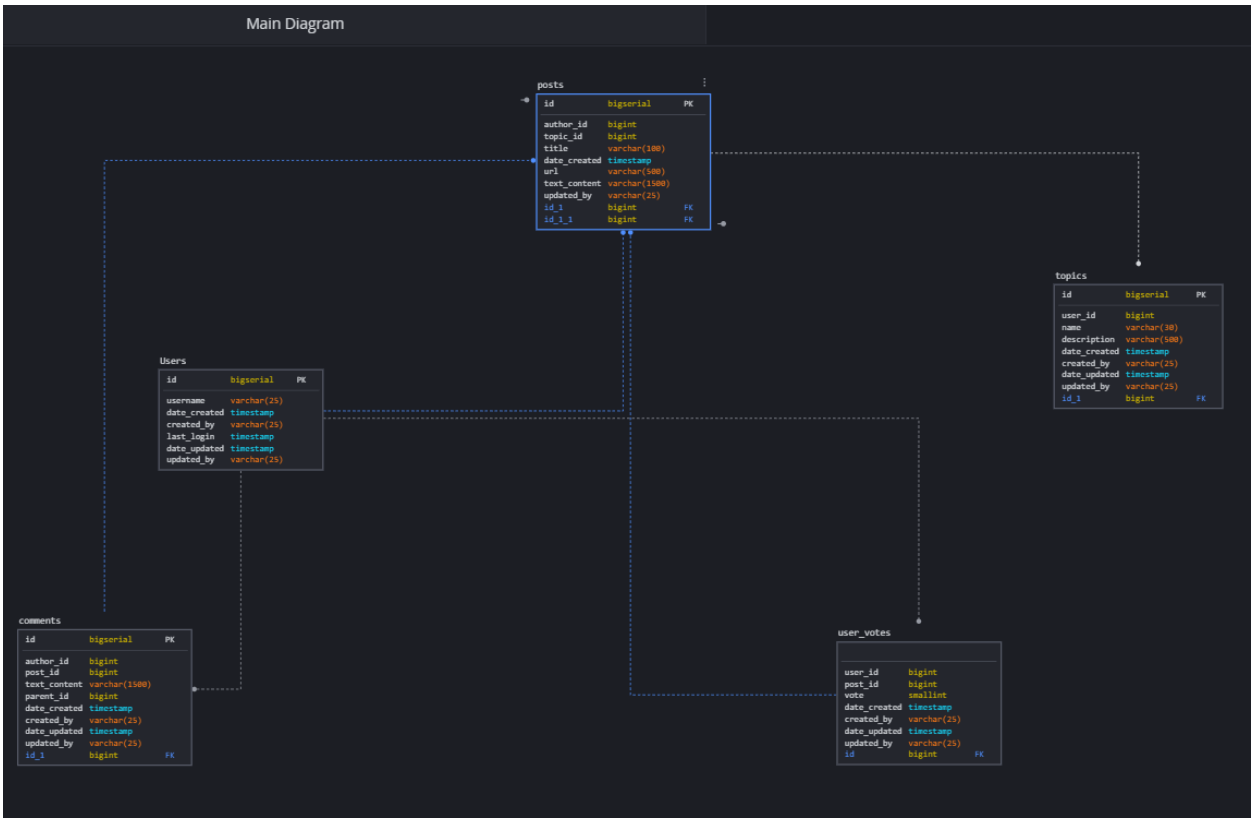


4. Guideline #4: your new database schema will be composed of five (5) tables that should have an auto-incrementing id as their primary key.

Once you've taken the time to think about your new schema, write the DDL for it in the space provided here:

[redacted]

[redacted]



**I purposely set the size of "text_content" and "url" fields solely based on the maximum lengths of data that was given on the specific research. Since the maximum length for the 'url' field was only 24. I decided to make it with a maximum of 500 characters.

Below are all the queries being utilized for this project:

Part 1: Data Definition Language (DDL) for a newer and better NORMALIZED SCHEMA
Users Table

```
CREATE TABLE Users
(
  id BIGSERIAL CONSTRAINT "PK_Users" PRIMARY KEY,
  username VARCHAR(25) CONSTRAINT "Usernames_Not_Null" NOT NULL,
  date_created TIMESTAMP,
  created_by VARCHAR(25),
  last_login TIMESTAMP,
  date_updated TIMESTAMP,
  updated_by VARCHAR(25),
  CONSTRAINT "Not_Permitted_Empty_Usernames" CHECK (LENGTH(TRIM("username")) > 0)
);

CREATE UNIQUE INDEX "Unique_Or_Distinct_Username" ON "users"(TRIM("username"));
```

Topics Table

```
CREATE TABLE Topics
(
  id BIGSERIAL CONSTRAINT "PK_Topics" PRIMARY KEY,
  user_id BIGINT DEFAULT NULL,
  name VARCHAR(30) CONSTRAINT "topic_name_not_null" NOT NULL,
  description VARCHAR(500) DEFAULT NULL,
  date_created TIMESTAMP,
  created_by VARCHAR(25),
  date_updated TIMESTAMP,
  updated_by VARCHAR(25),
  CONSTRAINT "Not_Permitted_Empty_Names" CHECK (LENGTH(TRIM("name")) > 0)
);

CREATE UNIQUE INDEX "Unique_Or_Distinct_Topics" ON "topics"(TRIM("name"));
CREATE INDEX ON "topics"(LOWER("name") VARCHAR_PATTERN_OPS);
```

Posts Table

```sql
CREATE TABLE Posts
(
  id BIGSERIAL CONSTRAINT "PK_Posts" PRIMARY KEY,
  author_id BIGINT,
  topic_id BIGINT CONSTRAINT "Topic_Required" NOT NULL,
  title VARCHAR(100) CONSTRAINT "Title_Not_Null" NOT NULL,
  date_created TIMESTAMP,
  created_by VARCHAR(25),
  date_updated TIMESTAMP,
  url VARCHAR(500) DEFAULT NULL,
  text_content VARCHAR(1500) DEFAULT NULL,
  updated_by VARCHAR(25),
  CONSTRAINT "Not_Permitted_Empty_Titles" CHECK (LENGTH(TRIM("title")) > 0),
  CONSTRAINT "TxtContent_or_Url" CHECK (
    (NULLIF(url,'') IS NULL OR NULLIF(text_content,'') IS NULL)
    AND NOT
    (NULLIF(url,'') IS NULL AND NULLIF(text_content,'') IS NULL)
  ),
  CONSTRAINT "FK_Posts_to_Users" FOREIGN KEY ("author_id") REFERENCES "users" ON DELETE SET
NULL,
  CONSTRAINT "FK_Posts_to_Topics" FOREIGN KEY ("topic_id") REFERENCES "topics" ON DELETE
CASCADE

);

CREATE INDEX "Posts_By_Users" ON "posts"("author_id");
CREATE INDEX "Locate_Posts_With_Url" ON "posts"("url");
CREATE INDEX "Topics_Post_Matching_URL" ON "posts"("topic_id");
```

Comments Table

```
CREATE TABLE Comments
(
  id BIGSERIAL CONSTRAINT "PK_Comments" PRIMARY KEY,
  author_id BIGINT,
  post_id BIGINT,
  text_content VARCHAR(1500) CONSTRAINT "Comment_Not_Null" NOT NULL,
  parent_id BIGINT DEFAULT NULL,
  date_created TIMESTAMP,
  created_by VARCHAR(25),
  date_updated TIMESTAMP,
  updated_by VARCHAR(25),
  CONSTRAINT "Not_Permitted_Empty_Comments" CHECK (LENGTH(TRIM("text_content"))>0),
  CONSTRAINT "FK_Parent_Child_Comments" FOREIGN KEY ("parent_id") REFERENCES "comments" ON
DELETE CASCADE,
  CONSTRAINT "FK_Comment_to_Post" FOREIGN KEY ("post_id") REFERENCES "posts" ON DELETE CASCADE,
  CONSTRAINT "FK_Comment_to_Users" FOREIGN KEY ("author_id") REFERENCES "users" ON DELETE SET
NULL
);

CREATE INDEX "Locate_Parent_Comments" ON "comments"("parent_id");
CREATE INDEX "Locate_Comments_By_Users" ON "comments"("author_id");
```

User_Votes Table

```
CREATE TABLE User_Votes
(
  user_id BIGINT,
  post_id BIGINT,
  vote SMALLINT CONSTRAINT "Valid_Votes" CHECK ("vote" IN (-1, 1)),
  date_created TIMESTAMP,
  created_by VARCHAR(25),
  date_updated TIMESTAMP,
  updated_by VARCHAR(25),
  CONSTRAINT "PK_User_Votes" PRIMARY KEY ("post_id","user_id"),
  CONSTRAINT "FK_Votes_to_Users" FOREIGN KEY ("user_id") REFERENCES "users" ON DELETE SET NULL,
  CONSTRAINT "FK_Votes_to_Posts" FOREIGN KEY ("post_id") REFERENCES "posts" ON DELETE CASCADE
);

CREATE INDEX "Votes_Computed" ON "user_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:

Step 2 – Migration of Data

Migrate Data into Users Table

```sql
INSERT INTO "users"("username")
  SELECT username
  FROM   bad_posts

  UNION

  SELECT regexp_split_to_table(upvotes, ',')::VARCHAR AS Username
  FROM   bad_posts

  UNION

  SELECT regexp_split_to_table(downvotes, ',')::VARCHAR AS Username
  FROM   bad_posts

  UNION

  SELECT (username::VARCHAR) AS username
  FROM   bad_comments;
```

Migrate Data to Topics

```sql
INSERT INTO "topics"("name")
  SELECT DISTINCT topic
  FROM bad_posts;
```

Migrate Data to Posts

```
INSERT INTO "posts"("id","author_id","topic_id","title","url","text_content")
  SELECT  BP.id::BIGINT AS PK_Posts,

          U.id AS Author_ID,
          T.id AS Topic_ID,
          BP.Title::VARCHAR(100),
          BP.url::VARCHAR(500),
          BP.Text_Content::VARCHAR


  FROM topics T
       INNER JOIN
       bad_posts BP
       ON
       T.name = BP.topic
       INNER JOIN
       users U
       ON
       BP.username = U.username;
```

Migrate Data to Comments

```
INSERT INTO "comments"("author_id","post_id","text_content")
  SELECT
          U.id AS author_id,
          BC.post_id,
          BC.text_content::VARCHAR(1500)
  FROM users U
       INNER JOIN
       bad_comments BC
       ON U.username = BC.username;
```

Migrate data to User_Votes Table:

```
INSERT INTO "user_votes"("user_id","post_id","vote")
 SELECT  Query2.user_id,
      Query1.post_id,
      Query1.vote
 FROM (
      (
       SELECT  BP.id::BIGINT AS post_id,
            regexp_split_to_table(bp.upvotes,',')::VARCHAR AS username,
            1::SMALLINT AS vote
       FROM    bad_posts BP
      ) AS Query1
      INNER JOIN
      (
       SELECT  U.id AS user_id,
            U.username
```

```
        FROM users U
      ) AS Query2
      ON Query1.username = Query2.username
    )

 UNION ALL

 SELECT  Query4.user_id,
      Query3.post_id,
      Query3.vote
 FROM (
      (
       SELECT  BP.id::BIGINT AS post_id,
            regexp_split_to_table(bp.downvotes,',')::VARCHAR AS username,
            -1::SMALLINT AS vote
       FROM    bad_posts bp
      ) AS Query3
      INNER JOIN
      (
       SELECT  u.id AS user_id,
            u.username
       FROM users U
      )AS Query4
      ON Query3.username = Query4.username
);
```

/*
A. List all users who haven't logged in the last year.
*/

```
SELECT  Username
FROM    Users
WHERE   last_login < NOW() - '1 year'::INTERVAL;
```

/*
B. List all users who haven't created any post.
*/

```
SELECT  U.username
FROM    users U
        LEFT JOIN
        posts p
        ON U.id = P.author_id
WHERE   P.author_id IS NULL;
```

/*
C. Find a user by their username.
*/

```sql
SELECT *
FROM users
WHERE username = 'Abbey.Gaylord5';
```

/*
D. List all topics that don't have any posts.
*/

```sql
SELECT  T.name
FROM    topics T
        LEFT JOIN
        posts P
        ON T.id = P.topic_id
WHERE   P.topic_id IS NULL;
```

/*
E. Find a topic by its name.
*/ note: These are 4 different queries being separated by 'OR' (means 4 options)

```sql
SELECT *
FROM topics
WHERE name = 'client-server';

OR

SELECT *
FROM topics
WHERE name = 'engineer';

OR

SELECT *
FROM topics
WHERE name = 'flexibility';

SELECT  *
FROM    topics
WHERE   name = 'Beauty';
```

/*
F. List the latest 20 posts for a given topic
*/

```sql
SELECT  T.name,
        P.url,
        P.title,
        u.username AS Author,

        P.text_content
FROM    topics T
        INNER JOIN
        posts p
        ON t.id = p.topic_id
        INNER JOIN
        users U
        ON U.id = P.author_id
WHERE   T.name = 'engineer'
ORDER BY  p.date_created DESC
LIMIT 20;
```

/*
G. List the latest 20 posts for a given user
*/

```sql
SELECT  p.title,
        u.username AS author,
        p.url,
        p.text_content
FROM    posts p
        INNER JOIN
        users u
        ON u.id = p.author_id AND u.username = 'Abbey.Gaylord5'
ORDER BY p.date_created DESC
LIMIT 20;
```

/*
H. Find all posts that link to a specific URL, for moderation purposes.
*/
Note: The ‘OR’ means these are 2 queries, being separated. I am giving options.

```sql
SELECT p.id,
       p.title,
       u.username AS author,
       u.id AS user_id,
       p.url
FROM   posts p
       INNER JOIN
       users u
       ON u.id = p.author_id  AND  p.url = 'http://woodrow.name'
ORDER BY p.id ASC;

OR

SELECT p.id,
       p.title,
       u.username AS author,
       u.id AS user_id,
       p.url
FROM   posts p
       INNER JOIN
       users u
       ON u.id = p.author_id  AND  p.url = 'http://emilio.net'
ORDER BY p.id ASC;
```

/*
I. List all the top-level comments (those that don't have a parent comment) for a given post.
*/

```sql
SELECT  p.title post,
        u.username AS author,
        c.text_content
FROM    posts p
        INNER JOIN
        comments c
        ON p.id = c.post_id AND c.parent_id IS NULL AND p.id = 10000
        INNER JOIN
        users u
        ON u.id = c.author_id;
```

/*
J. List all the direct children of a parent comment.
*/

```sql
SELECT  u.username AS author,
        parent.id AS parent_comment_id,
        child.id AS child_comment_id,
        child.date_created AS comment_date,
        child.text_content
FROM    comments parent
        INNER JOIN
        comments child
        ON child.parent_id = parent.id AND parent.id = 1
        INNER JOIN
        users u
        ON child.author_id = u.id
ORDER BY 4 ASC;
```

/*
K. List the latest 20 comments made by a given user.
*/

```sql
SELECT  u.username AS author,
        c.text_content
FROM    comments C
        INNER JOIN
        users u
        ON c.author_id = u.id AND u.username = 'Audrey.Hickle12'
ORDER BY c.date_created DESC
LIMIT 20;
```

/*
L. Compute the score of a post, defined as the difference between the number of upvotes and the number of downvotes.
*/

```sql
SELECT  P.title,
        SUM(UV.vote) AS Post_Scores
FROM    posts P
        INNER JOIN
        user_votes UV
        ON P.id = UV.post_id
GROUP BY P.title
ORDER BY 2 DESC, 1 ASC;
```

**These same queries are also written on a separate file and also down below

```
/*
    Part 1 - DDL for newer and much BETTER Normalized Schema
*/

CREATE TABLE Users
(
  id BIGSERIAL CONSTRAINT "PK_Users" PRIMARY KEY,
  username VARCHAR(25) CONSTRAINT "Usernames_Not_Null" NOT NULL,
  date_created TIMESTAMP,
  created_by VARCHAR(25),
  last_login TIMESTAMP,
  date_updated TIMESTAMP,
  updated_by VARCHAR(25),
  CONSTRAINT "Not_Permitted_Empty_Usernames" CHECK (LENGTH(TRIM("username")) > 0)
);

CREATE UNIQUE INDEX "Unique_Or_Distinct_Username" ON "users"(TRIM("username"));

CREATE TABLE Topics
(
  id BIGSERIAL CONSTRAINT "PK_Topics" PRIMARY KEY,
  user_id BIGINT DEFAULT NULL,
  name VARCHAR(30) CONSTRAINT "topic_name_not_null" NOT NULL,
  description VARCHAR(500) DEFAULT NULL,
  date_created TIMESTAMP,
  created_by VARCHAR(25),
  date_updated TIMESTAMP,
  updated_by VARCHAR(25),
  CONSTRAINT "Not_Permitted_Empty_Names" CHECK (LENGTH(TRIM("name")) > 0)
);

CREATE UNIQUE INDEX "Unique_Or_Distinct_Topics" ON "topics"(TRIM("name"));
CREATE INDEX ON "topics"(LOWER("name") VARCHAR_PATTERN_OPS);


CREATE TABLE Posts
(
  id BIGSERIAL CONSTRAINT "PK_Posts" PRIMARY KEY,
  author_id BIGINT,
  topic_id BIGINT CONSTRAINT "Topic_Required" NOT NULL,
  title VARCHAR(100) CONSTRAINT "Title_Not_Null" NOT NULL,
  date_created TIMESTAMP,
  created_by VARCHAR(25),
  date_updated TIMESTAMP,
  url VARCHAR(500) DEFAULT NULL,
  text_content VARCHAR(1500) DEFAULT NULL,
  updated_by VARCHAR(25),
  CONSTRAINT "Not_Permitted_Empty_Titles" CHECK (LENGTH(TRIM("title")) > 0),
  CONSTRAINT "TxtContent_or_Url" CHECK (
    (NULLIF(url,'') IS NULL OR NULLIF(text_content,'') IS NULL)
    AND NOT
    (NULLIF(url,'') IS NULL AND NULLIF(text_content,'') IS NULL)
  ),
```

```
  CONSTRAINT "FK_Posts_to_Users" FOREIGN KEY ("author_id") REFERENCES "users" ON DELETE SET
NULL,
  CONSTRAINT "FK_Posts_to_Topics" FOREIGN KEY ("topic_id") REFERENCES "topics" ON DELETE
CASCADE

);

CREATE INDEX "Posts_By_Users" ON "posts"("author_id");
CREATE INDEX "Locate_Posts_With_Url" ON "posts"("url");
CREATE INDEX "Topics_Post_Matching_URL" ON "posts"("topic_id");


CREATE TABLE Comments
(
  id BIGSERIAL CONSTRAINT "PK_Comments" PRIMARY KEY,
  author_id BIGINT,
  post_id BIGINT,
  text_content VARCHAR(1500) CONSTRAINT "Comment_Not_Null" NOT NULL,
  parent_id BIGINT DEFAULT NULL,
  date_created TIMESTAMP,
  created_by VARCHAR(25),
  date_updated TIMESTAMP,
  updated_by VARCHAR(25),
  CONSTRAINT "Not_Permitted_Empty_Comments" CHECK (LENGTH(TRIM("text_content"))>0),
  CONSTRAINT "FK_Parent_Child_Comments" FOREIGN KEY ("parent_id") REFERENCES "comments" ON
DELETE CASCADE,
  CONSTRAINT "FK_Comment_to_Post" FOREIGN KEY ("post_id") REFERENCES "posts" ON DELETE
CASCADE,
  CONSTRAINT "FK_Comment_to_Users" FOREIGN KEY ("author_id") REFERENCES "users" ON DELETE
SET NULL
);

CREATE INDEX "Locate_Parent_Comments" ON "comments"("parent_id");
CREATE INDEX "Locate_Comments_By_Users" ON "comments"("author_id");

CREATE TABLE User_Votes
(
  user_id BIGINT,
  post_id BIGINT,
  vote SMALLINT CONSTRAINT "Valid_Votes" CHECK ("vote" IN (-1, 1)),
  date_created TIMESTAMP,
  created_by VARCHAR(25),
  date_updated TIMESTAMP,
  updated_by VARCHAR(25),
  CONSTRAINT "PK_User_Votes" PRIMARY KEY ("post_id","user_id"),
  CONSTRAINT "FK_Votes_to_Users" FOREIGN KEY ("user_id") REFERENCES "users" ON DELETE SET
NULL,
  CONSTRAINT "FK_Votes_to_Posts" FOREIGN KEY ("post_id") REFERENCES "posts" ON DELETE
CASCADE
);

CREATE INDEX "Votes_Computed" ON "user_votes"("vote");

/*
```

```
    Part 2 - Data Migration
*/

/* Step 1 - Migrate data to "users" table
   I used "UNION" to remove all duplicates. */

INSERT INTO "users"("username")
  SELECT username
  FROM   bad_posts

  UNION

  SELECT regexp_split_to_table(upvotes, ',')::VARCHAR AS Username
  FROM   bad_posts

  UNION

  SELECT regexp_split_to_table(downvotes, ',')::VARCHAR AS Username
  FROM   bad_posts

  UNION

  SELECT (username::VARCHAR) AS username
  FROM   bad_comments;

/* Step 2 - Migrate data to "topics" table
   "DISTINCT" used to remove duplicates */

INSERT INTO "topics"("name")
  SELECT DISTINCT topic
  FROM bad_posts;

/* Step 3 - Migrate data to "posts" table */

INSERT INTO "posts"("id","author_id","topic_id","title","url","text_content")
  SELECT  BP.id::BIGINT AS PK_Posts,

        U.id AS Author_ID,
        T.id AS Topic_ID,
        BP.Title::VARCHAR(100),
        BP.url::VARCHAR(500),
        BP.Text_Content::VARCHAR



  FROM topics T
      INNER JOIN
      bad_posts BP
      ON
      T.name = BP.topic
      INNER JOIN
      users U
      ON
      BP.username = U.username;
```

```
/* Step 4 - Migrate data to "comments" table */

INSERT INTO "comments"("author_id","post_id","text_content")
  SELECT
        U.id AS author_id,
        BC.post_id,
        BC.text_content::VARCHAR(1500)
  FROM users U
      INNER JOIN
      bad_comments BC
      ON U.username = BC.username;

/* Step 5 - Migrate data to "user_votes" table */

INSERT INTO "user_votes"("user_id","post_id","vote")
  SELECT  Query2.user_id,
        Query1.post_id,
        Query1.vote
  FROM (
        (
          SELECT  BP.id::BIGINT AS post_id,
                regexp_split_to_table(bp.upvotes,',')::VARCHAR AS username,
                1::SMALLINT AS vote
          FROM    bad_posts BP
        ) AS Query1
        INNER JOIN
        (
          SELECT  U.id AS user_id,
                U.username
          FROM users U
        ) AS Query2
        ON Query1.username = Query2.username
      )

  UNION ALL

  SELECT  Query4.user_id,
        Query3.post_id,
        Query3.vote
  FROM (
        (
          SELECT  BP.id::BIGINT AS post_id,
                regexp_split_to_table(bp.downvotes,',')::VARCHAR AS username,
                -1::SMALLINT AS vote
          FROM    bad_posts bp
        ) AS Query3
        INNER JOIN
        (
          SELECT  u.id AS user_id,
                u.username
          FROM users U
        )AS Query4
        ON Query3.username = Query4.username
```

```
);


/*
  A. List all users who haven't logged in the last year.
*/

SELECT  Username
FROM    Users
WHERE   last_login < NOW() - '1 year'::INTERVAL;

/*
  B. List all users who haven't created any post.
*/

SELECT  U.username
FROM    users U
        LEFT JOIN
        posts p
        ON U.id = P.author_id
WHERE   P.author_id IS NULL;

/*
  C. Find a user by their username.
*/


SELECT *
FROM users
WHERE username = 'Abbey.Gaylord5';

/*
  D. List all topics that don't have any posts.
*/

SELECT  T.name
FROM    topics T
        LEFT JOIN
        posts P
        ON T.id = P.topic_id
WHERE   P.topic_id IS NULL;

/*
  E. Find a topic by its name.
*/

SELECT *
FROM topics
WHERE name = 'client-server';

OR

SELECT *
FROM topics
```

```
WHERE name = 'engineer';

OR

SELECT *
FROM topics
WHERE name = 'flexibility';

SELECT  *
FROM    topics
WHERE   name = 'Beauty';



/*
  F. List the latest 20 posts for a given topic
*/

SELECT  T.name,
        P.url,
        P.title,
        u.username AS Author,

        P.text_content
FROM    topics T
        INNER JOIN
        posts p
        ON t.id = p.topic_id
        INNER JOIN
        users U
        ON U.id = P.author_id
WHERE   T.name = 'engineer'
ORDER BY  p.date_created DESC
LIMIT 20;

/*
  G.  List the latest 20 posts for a given user
*/

SELECT  p.title,
        u.username AS author,
        p.url,
        p.text_content
FROM    posts p
        INNER JOIN
        users u
        ON u.id = p.author_id AND u.username = 'Dora55'
ORDER BY p.date_created DESC
LIMIT 20;


OR
```

```
SELECT  p.title,
        u.username AS author,
        p.url,
        p.text_content
FROM    posts p
        INNER JOIN
        users u
        ON u.id = p.author_id AND u.username = 'Abbey.Gaylord5'
ORDER BY p.date_created DESC
LIMIT 20;

/*
  H. Find all posts that link to a specific URL, for moderation purposes.
*/ note: I am giving 2 options here, being separated by 'OR'

SELECT p.id,
       p.title,
       u.username AS author,
       u.id AS user_id,
       p.url
FROM   posts p
       INNER JOIN
       users u
       ON u.id = p.author_id  AND  p.url = 'http://woodrow.name'
ORDER BY p.id ASC;

OR

SELECT p.id,
       p.title,
       u.username AS author,
       u.id AS user_id,
       p.url
FROM   posts p
       INNER JOIN
       users u
       ON u.id = p.author_id  AND  p.url = 'http://emilio.net'
ORDER BY p.id ASC;

/*
  I. List all the top-level comments (those that don't have a parent comment) for a given post.
*/

SELECT  p.title post,
        u.username AS author,
        c.text_content
FROM    posts p
        INNER JOIN
        comments c
        ON p.id = c.post_id AND c.parent_id IS NULL AND p.id = 10000
        INNER JOIN
        users u
        ON u.id = c.author_id;
```

```
/*
  J. List all the direct children of a parent comment.
*/

SELECT  u.username AS author,
        parent.id AS parent_comment_id,
        child.id AS child_comment_id,
        child.date_created AS comment_date,
        child.text_content
FROM    comments parent
        INNER JOIN
        comments child
        ON child.parent_id = parent.id AND parent.id = 1
        INNER JOIN
        users u
        ON child.author_id = u.id
ORDER BY 4 ASC;

/*
  K.  List the latest 20 comments made by a given user.
*/

SELECT  u.username AS author,
        c.text_content
FROM    comments C
        INNER JOIN
        users u
        ON c.author_id = u.id AND u.username = 'Audrey.Hickle12'
ORDER BY c.date_created DESC
LIMIT 20;

/*
  L. Compute the score of a post, defined as the difference between the number of
     upvotes and the number of downvotes.
*/

SELECT  P.title,
        SUM(UV.vote) AS Post_Scores
FROM    posts P
        INNER JOIN
        user_votes UV
        ON P.id = UV.post_id
GROUP BY P.title
ORDER BY 2 DESC, 1 ASC;
```

Sources of information and disclaimer:

I, Frederick Zoreta, have used outside resources (other than Udacity) in learning and practicing SQL. I have listed the exact links and lectures that I have been using for more than 1 year.

1.www.datacamp.com

My academic profile: https://www.datacamp.com/profile/ericzoreta808

Similar courses utilized in research/practice of SQL:

Joining Data in PostgreSQL:
https://learn.datacamp.com/courses/joining-data-in-postgresql

Exploratory Data Analysis in SQL:
https://learn.datacamp.com/courses/exploratory-data-analysis-in-sql
Intermediate SQL:
https://learn.datacamp.com/courses/intermediate-sql
PostGreSQL Summary Stats & Window Functions:
https://learn.datacamp.com/courses/postgresql-summary-stats-and-window-functions
Analyzing Business Data in SQL:
https://learn.datacamp.com/courses/analyzing-business-data-in-sql
Applying SQL to the Real World :
https://learn.datacamp.com/courses/applying-sql-to-real-world-problems
Improving PostGreSQL Query Performance:
https://learn.datacamp.com/courses/improving-query-performance-in-postgresql
Creating PostGreSQL Databases:
https://learn.datacamp.com/courses/creating-postgresql-databases

2.www.TeamTreeHouse.com

My academic profile: https://teamtreehouse.com/frederickzoreta

Similar courses utilized in research/practice of SQL:

Reporting with SQL:
https://teamtreehouse.com/library/reporting-with-sql
Querying Relational Databases:
https://teamtreehouse.com/library/querying-relational-databases
Common Table Expressions (CTE) using WITH() Function:
https://teamtreehouse.com/library/common-table-expressions-using-with
Modifying Data with SQL:
https://teamtreehouse.com/library/modifying-data-with-sql

3.www.udemy.com

My academic profile: https://www.udemy.com/user/fredrickzoreta/

Similar courses utilized in research/practice of SQL:

MySQL for Data Analysis:
https://www.udemy.com/course/mysql-for-data-analysis/learn/lecture/15211486#overview
Advanced MySQL for Data Analytics:
https://www.udemy.com/course/advanced-sql-mysql-for-analytics-business-intelligence/learn/lecture/16450492#overview
The Complete SQL Bootcamp- Zero to Hero :
https://www.udemy.com/course/the-complete-sql-bootcamp/
Mastering SQL for Data Science:
https://www.udemy.com/course/master-sql-for-data-science/
The Complete Oracle Course:
https://www.udemy.com/course/the-complete-oracle-sql-certification-course/