DATABASE LAB

POST-MIDSEM PROJECT

Group 1

Mathematics and Computing

Aishwarya Gupta - 10012301
Chirag Maheshwari - 10012315
Parikshit Upadhyaya - 10012324
Abhinandan - 10012339

ABOUT

The Project is titled "***wikiblog***". It is a ***blog and article writing/viewing*** web-application for users.

Its features include:

- Every writer will have a separate user account identified by a *unique username*.
- A user has to authenticate him/her-self using a password set by the user himself.
- Every user will have its own profile (personal details + profile image).
- One user can follow another to get his updates.
- A user can view the blogs and articles without having a user account.
- Every user has the option to write a personal blog or an article for the general community.
- A blog post has only *one* version and it belongs to a particular user.
- An article is anonymous and can be edited or re-written by any user.
- Every version of the article will be stored differentiated from one-another using a version number and its timestamp.
- One can search for other users, articles or even blogs.
- For the search engine to work,
    - Blog/article has a set of tags.
    - Blog/article contains keywords.
    - An article will contain references.
    - User has a set of interests, a username.
- A user can rate an article out of 5.

Salient Features

1. “wikiblog” allows the users to write articles as well as post blogs. The articles correspond to the “wiki” part in “wikiblog” and “blog” is for posts.

2. Users have to register/login themselves before writing and article or posting a blog-post and even rate an article or like/dislike a bog-post. But, one can view articles, user-profiles, and blog-posts without registering and can search for them using the search bar.

3. Articles are open for editing and rating to all the registered users and all the versions of the article are stored and displayed.

4. Blogs are open for likes/dislikes and comments/replies to the comments by all the registered users.

5. Some features of social networking is also there, for e.g., users can follow other users which allows him/her to look for articles and posts of only the people he/she is interested in.

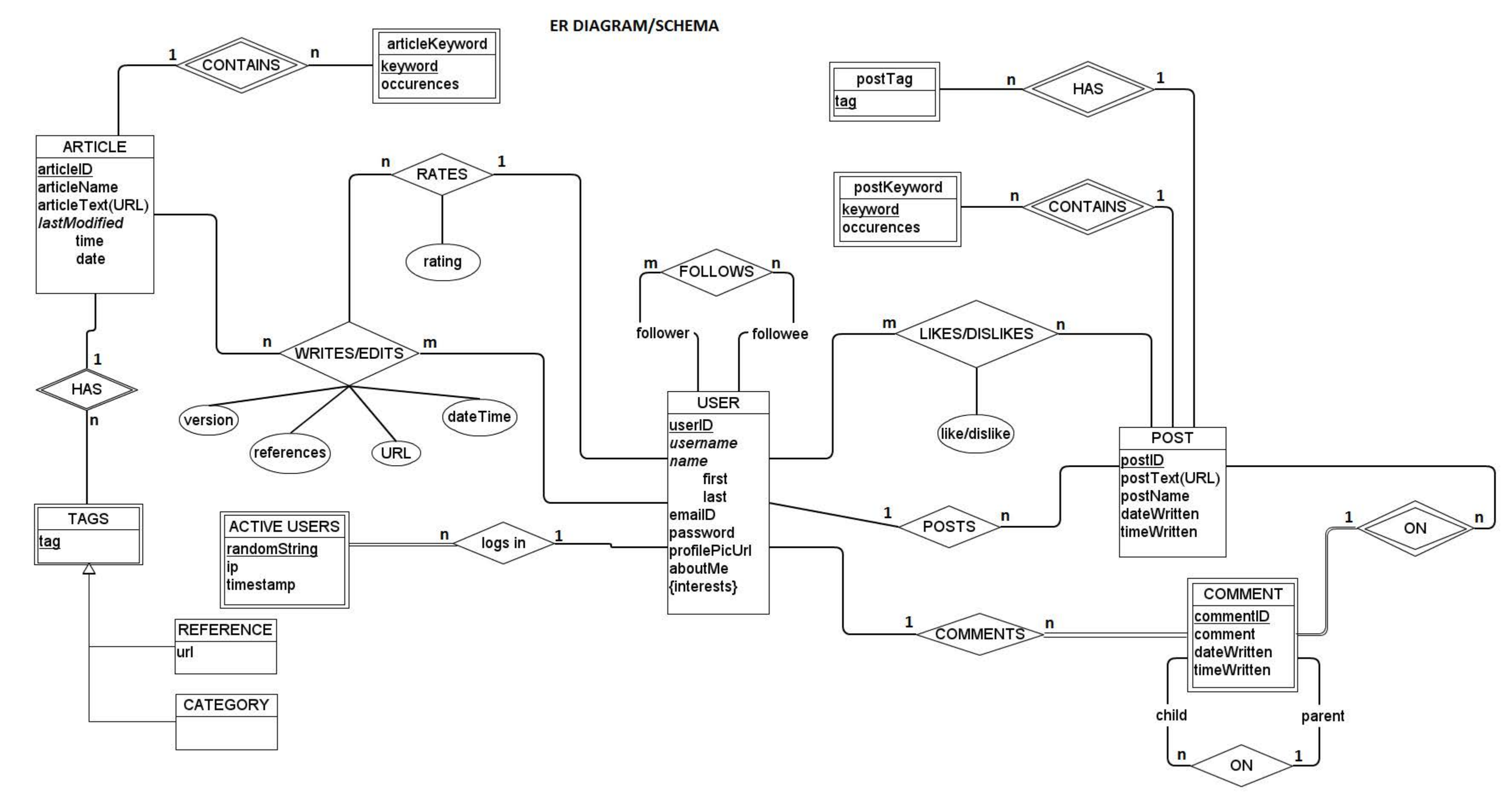
6. You can look up for an article/post/user based on the keywords and tags for article/post and username and interests for a user.

7. One can see all the users who have liked/disliked a blog-post.

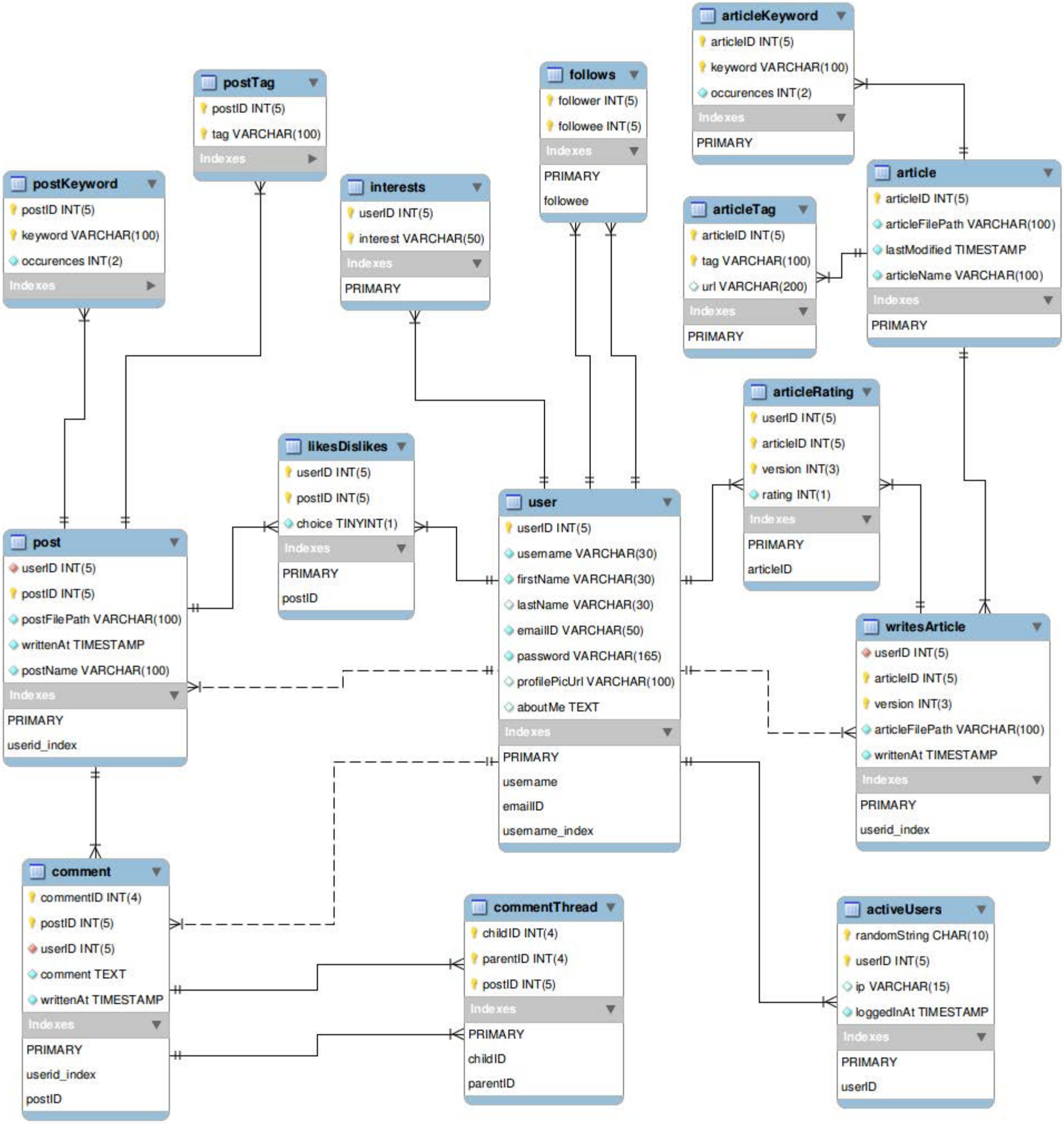
8. On the front page, some of the newly added articles and blogposts are shown.

Some Technical Information

- **Database Management System:**
  - MySQL (version: 5.5.29)
  - NAMES utf8
  - TIME_ZONE='+00:00'
  - DEFAULT CHARACTER latin1
  - DEFAULT STORAGE ENGINE = InnoDB
- **Apache HTTP Server**
  - Version: 2.4.4
  - Installed PhP and MySQL access modules
- **phpMyAdmin**
  - Web-based GUI for database management
  - Used it for easy viewing of database changes
- **MySQL WorkBench**
  - Desktop based GUI for database management
- Backend written in **PhP 5**
- Used prepared statement for database access to prevent first degree of SQL injection.


ER DIAGRAM/SCHEMA
articleKeyword
keyword
occurences
CONTAINS
1
n
ARTICLE
articleID
articleName
articleText(URL)
lastModified
time
date
RATES
rating
WRITES/EDITS
version
references
URL
dateTime
HAS
TAGS
tag
REFERENCE
url
CATEGORY
ACTIVE USERS
randomString
ip
timestamp
logs in
postTag
tag
HAS
postKeyword
keyword
occurences
CONTAINS
FOLLOWS
follower
followee
USER
userID
username
name
first
last
emailID
password
profilePicUrl
aboutMe
{interests}
LIKES/DISLIKES
like/dislike
POST
postID
postText(URL)
postName
dateWritten
timeWritten
POSTS
ON
COMMENT
commentID
comment
dateWritten
timeWritten
COMMENTS
child
parent
ON
m
n
1

RELATIONAL SCHEMA



TABLES & CONSTRAINTS

user

1. userID :int(5), not null, auto increment
2. username : varchar(30), alternate indexing (B-Tree), not null
3. first name: varchar(30), not null
4. last name: varchar(30), not null
5. emailID: varchar(50), not null
6. password: varchar(165), not null, (because of the SHA-1 algorithm size)
7. profilePicUrl: varchar(100)
8. aboutMe: text
9. PRIMARY KEY(userID)
10. UNIQUE (username)
11. UNIQUE (emailID)
12. INDEX index_name USING Btree (username)

interests

1. userID: int(5), not null
2. interest: varchar(100), not null
3. PRIMARY KEY(userID, interest)
4. FOREIGN KEY userID REFERENCES user ON DELETE CASCADE ON UPDATE CASCADE

follows

1. follower: int(5), not null
2. followee: int(5), not null
3. PRIMARY KEY(follower, followee)
4. FOREIGN KEY follower REFERENCES user ON DELETE CASCADE ON UPDATE CASCADE
5. FOREIGN KEY followee REFERENCES user ON DELETE CASCADE ON UPDATE CASCADE

article

1. articleID: int(5), not null, auto increment
2. articleText: text, not null
3. articleName: varchar(100), not null, default 'new article'
4. lastModified: timestamp, not null, default CURRENT_TIMESTAMP
5. PRIMARY KEY(articleID)

writesArticle

1. userID: int(5), not null
2. articleID: int(5), not null
3. version: int(3), not null
4. articleText: text, not null

5. writtenAt: timestamp, not null
6. PRIMARY KEY(articleID, version)
7. INDEX index_name USING Btree (userID)
8. FOREIGN KEY userID REFERENCES user ON DELETE CASCADE ON UPDATE CASCADE
9. FOREIGN KEY articleID REFERENCES article ON DELETE CASCADE ON UPDATE CASCADE

articleRating

1. userID: int(5), not null
2. articleID: int(5), not null
3. version: int(3), not null
4. rating: int(1), not null
5. PRIMARY KEY(userID, articleID, version)
6. FOREIGN KEY userID REFERENCES user ON DELETE CASCADE ON UPDATE CASCADE
7. FOREIGN KEY (articleID, version) REFERENCES writesArticle ON DELETE CASCADE ON UPDATE CASCADE

activeUsers

1. randomString: char(10), not null
2. userID: int(5), not null
3. ip: char(15)
4. loggedInAt: timestamp, default CURRENT_TIMESTAMP
5. PRIMARY KEY(randomString, userID)
6. FOREIGN KEY userID REFERENCES user ON DELETE CASCADE ON UPDATE CASCADE

post

1. userID: int(5), not null, auto increment
2. postID: int(5), not null
3. postName: varchar(100), not null, default 'new post'
4. postText: text, not null
5. writtenAt: timestamp, not null, default CURRENT_TIMESTAMP
6. PRIMARY KEY(postID)
7. INDEX index_name USING Btree (userID)
8. FOREIGN KEY userID REFERENCES user ON DELETE CASCADE ON UPDATE CASCADE

comment

1. commentID: int(4), not null
2. postID: int(5), not null
3. userID: int(5), not null
4. comment: text, not null
5. writtenAt: timestamp, not null, default CURRENT_TIMESTAMP
6. PRIMARY KEY(commentID, postID)

7. INDEX index_name USING Btree (userID)
8. FOREIGN KEY userID REFERENCES user ON DELETE CASCADE ON UPDATE CASCADE
9. FOREIGN KEY postID REFERENCES post ON DELETE CASCADE ON UPDATE CASCADE

Comment_thread

1. childID: int(4), not null
2. parentID: int(4), not null
3. postID: int(5), not null
4. PRIMARY KEY(childID, parentID, postID)
5. FOREIGN KEY(childID, postID) REFERENCES comment ON DELETE CASCADE ON UPDATE CASCADE
6. FOREIGN KEY(parentID, postID) REFERENCES comment ON DELETE CASCADE ON UPDATE CASCADE

likesDislikes

1. userID: int(5), not null
2. postID: int(5), not null
3. choice: boolean, not null
4. PRIMARY KEY (userID, postID)
5. FOREIGN KEY userID REFERENCES user ON DELETE CASCADE ON UPDATE CASCADE
6. FOREIGN KEY postID REFERENCES post ON DELETE CASCADE ON UPDATE CASCADE

articleTag

1. articleID: int(5), not null
2. tag: varchar(100), not null
3. url: varchar(200)
4. PRIMARY KEY (articleID, tag)
5. FOREIGN KEY articleID REFERENCES article ON DELETE CASCADE ON UPDATE CASCADE

postTag

1. postID: int(5), not null
2. tag: varchar(100), not null
3. PRIMARY KEY (postID, tag)
4. FOREIGN KEY postID REFERENCES post ON DELETE CASCADE ON UPDATE CASCADE

articleKeyword

1. articleID: int(5), not null
2. keyword: varchar(100), not null

3. occurences: int(2), not null, deafult 1
4. PRIMARY KEY (articleID, keyword)
5. FOREIGN KEY articleID REFERENCES article ON DELETE CASCADE ON UPDATE CASCADE

postKeyword

1. postID: int(5), not null
2. keyword: varchar(100), not null
3. occurences: int(2), not null, deafult 1
4. PRIMARY KEY (postID, keyword)
5. FOREIGN KEY postID REFERENCES post ON DELETE CASCADE ON UPDATE CASCADE

Final QUERIES

```
-- QUERIES ON ARTICLE
    -- count the number of articles without versioning taking into account
        SELECT count(DISTINCT(articleID)) FROM writesArticle WHERE userID = ?
    -- count the number of articles
        SELECT count(articleID) FROM writesArticle WHERE userID = ?
    -- get the timing of an article using its article-ID
        SELECT lastModified FROM article WHERE articleID = ?
    -- get the user information for an article with a fixed version
        SELECT userID FROM writesArticle WHERE articleID = ? AND version = ?
    -- insert a new article
        INSERT INTO article(articleName, articleText) VALUES (?, ?)
    -- insert a new version of article in the writesArticle
        INSERT INTO writesArticle(userID, articleID, version, articleText,
writtenAt) VALUES (?, ?, ?, ?, ?)
    -- insert a new article tag
        INSERT INTO articleTag(articleID, tag, url) VALUES (?, ?, ?)
    -- insert a new keyword in an article with its number of occurences
        INSERT INTO articleKeyword(articleID, keyword, occurences) VALUES (?,
?, ?)
    -- insert a new rating to an article
        INSERT INTO articleRating(userID, articleID, version, rating) VALUES
(?, ?, ?, ?)
    -- get the latest article given an article-ID
        SELECT articleText, lastModified, articleName FROM article WHERE
articleID = ?
    -- get all the articles written by a particular user taking into account
the versioning.
        SELECT articleID, version, articleText, writtenAt FROM writesArticle
WHERE userID = ? ORDER BY writtenAt DESC
    -- get all the tags and references for an article.
        SELECT tag, url FROM articleTag WHERE articleID = ?
    -- get the information for an article for a particular version
        SELECT userID, articleText, writtenAt FROM writesArticle WHERE
articleID = ? AND version = ?
    -- get the last written/edited version for an article
        SELECT max(version) FROM writesArticle WHERE articleID = ?
    -- get all the information for an article including its version ordered
on the basis of version
        SELECT userID, version, writtenAt FROM writesArticle WHERE articleID
= ? ORDER BY version
    -- get the average rating for an article
        SELECT avg(rating) FROM articleRating WHERE articleID = ? AND version
= ?
    -- get the rating of article for an article by a particular user.
        SELECT avg(rating) FROM articleRating WHERE userID = ? AND articleID
= ? AND version = ?
    -- update the details for an article when a new version is written
        UPDATE article SET articleText = ?, lastModified = ? WHERE articleID
= ?
    -- this is a part of the search query.
        -- get the all the articles with the given keyword matching its tags.
            SELECT articleID FROM articleTag WHERE tag LIKE ?
```

```
        -- get all the articles containing the keyword between its body
ordered by the number of occurences for basic ranking.
            SELECT articleID FROM articleKeyword WHERE keyword LIKE ? ORDER
BY occurences DESC
    -- get the latest written articles limited by a number
        SELECT articleID, articleText, articleName, lastModified FROM article
ORDER BY lastModified DESC LIMIT ?;

-- Queries related to a user
    -- add a new user when a new registration happens
        INSERT INTO user(username, firstName, lastName, emailID, password,
profilePicUrl, aboutMe) VALUES (?, ?, ?, ?, ?, ?, ?)
    -- insert interests for the user
        INSERT INTO interests(userID, interest) VALUES (?, ?)
    -- insert an user login details, i.e., when it becomes an active user
        INSERT INTO activeUsers(randomString, userID, ip) VALUES (?, ?, ?)
    -- get the userid with a particular username
        SELECT userID from user WHERE username = ?
    -- get an userid with a particular email address
        SELECT userID from user WHERE emailID = ?
    -- check for a logged in user comparing its unique random string and
userid combination
        SELECT loggedInAt from activeUsers WHERE userID = ? AND randomString
= ?
    -- get the password hash for a given username
        SELECT userID, password FROM user WHERE username = ?
    -- get all the details for an user using its userid
        SELECT username, firstname, lastname, emailID, profilePicUrl, aboutme
from user WHERE userID = ?
    -- get all the interests for an user
        SELECT interest from interests WHERE userID = ?
    -- get all the details for an user using its username
        SELECT userID, firstname, lastname, emailID, profilePicUrl, aboutme
from user WHERE username = ?
    -- update an user details when the user edits its profile
        UPDATE user set firstname = ?, lastname = ?, profilePicUrl = ?,
aboutme = ? WHERE userID = ?
    -- update an user details with its password
        UPDATE user set firstname = ?, lastname = ?, profilePicUrl = ?,
aboutme = ?, password = ? WHERE userID = ?
    -- update an user details withs its password and email address
        UPDATE user set firstname = ?, lastname = ?, profilePicUrl = ?,
aboutme = ?, password = ?, emailID = ? WHERE userID = ?
    -- update an user details with its email address
        UPDATE user set firstname = ?, lastname = ?, profilePicUrl = ?,
aboutme = ?, emailID = ? WHERE userID = ?
    -- delete all interests for an user
        DELETE from interests WHERE userID = ?
    -- part of the search query
        -- find all the users with keyword as a substring in its username
            SELECT userID FROM user where username LIKE ?
        -- find all the users with keyword as a substring iin one of its
interests
            SELECT userID from interests where interest LIKE ?

-- queries related to a post
    -- get the post count for an user
```

```
        SELECT count(*) FROM post WHERE userID = ?
    -- add a new post
        INSERT INTO post(userID, postText, postName) VALUES (?, ?, ?)
    -- add tags for a blog-post
        INSERT INTO postTag(postID, tag) VALUES (?, ?)
    -- add keywords with its occurences for a post
        INSERT INTO postKeyword(postID, keyword, occurences) VALUES (?, ?, ?)
    -- get all the details for a post
        SELECT userID, postText, writtenAt, postName FROM post WHERE postID =
?
    -- get all the posts written by a particular user
        SELECT postID, postText, writtenAt, postName FROM post WHERE userID =
? ORDER BY writtenAt DESC
    -- get all the tags for a post
        SELECT tag FROM postTag WHERE postID = ?
    -- part of the search query
        -- find all the post containing the query as a substring in one of
its tags
            SELECT postID FROM postTag WHERE tag LIKE ?
        -- find all the post containing the keyword as a part of its body
            SELECT postID FROM postKeyword WHERE keyword LIKE ? ORDER BY
occurences DESC
    -- get all the latest post limited by a number
        SELECT postID, postText, postName, writtenAt FROM post ORDER BY
writtenAt DESC LIMIT ?

-- queries related to followers
    -- count the number of followers
        SELECT count(*) from follows WHERE followee = ?
    -- count the number of followees
        SELECT count(*) from follows WHERE follower = ?
    -- check for a particular following
        SELECT follower, followee from follows WHERE follower = ? AND
followee = ?
    -- add a new following
        INSERT INTO follows(follower, followee) VALUES (?, ?)
    -- delete a following
        DELETE FROM follows WHERE follower = ? AND followee = ?
    -- get all the followers
        SELECT follower from follows WHERE followee = ?
    -- get all the followees
        SELECT followee from follows WHERE follower = ?

-- queries related to comments
    -- count the number of comments
        SELECT count(*) FROM comment WHERE postID = ?
    -- get all the root comments for a particular post
        SELECT userID, comment, writtenAt, commentID FROM comment C WHERE
postID = ? AND NOT EXISTS (SELECT * FROM commentThread CT WHERE CT.childID =
C.commentID) ORDER BY writtenAt
    -- get all the children for a comment (should be used in recursion to get
all the articles)
        SELECT userID, comment, writtenAt, commentID FROM comment C WHERE
commentID IN (SELECT childID FROM commentThread WHERE parentID = ? AND postID
= ?)  ORDER BY writtenAt
    -- get the last entered commentID
        SELECT max(commentID) FROM comment WHERE postID = ?
```

```
    -- add a new comment
        INSERT INTO comment(commentID, postID, userID, comment) VALUES (?, ?,
?, ?)
    -- add a new parent child relationship for replies on comments
        INSERT INTO commentThread(childID, parentID, postID) VALUES (?, ?, ?)

-- queries related to likes
    -- get the number of likes
        SELECT count(*) FROM likesDislikes WHERE postID = ? AND choice = 1
    -- get the number of dislikes
        SELECT count(*) FROM likesDislikes WHERE postID = ? AND choice = 0
    -- get the users who like a post
        SELECT userID FROM likesDislikes WHERE postID = ? AND choice = 1
    -- get the users who dislike a post
        SELECT userID FROM likesDislikes WHERE postID = ? AND choice = 0
    -- get the choice of an user on a post
        SELECT choice FROM likesDislikes WHERE postID = ? AND userID = ?
    -- add a new liking/disliking
        INSERT INTO likesDislikes(userID, postID, choice) VALUES (?, ?, ?)
    -- update the old liking/disliking
        UPDATE likesDislikes SET choice = ? WHERE userID = ? AND postID = ?
    -- remove any liking/disliking for a post by an user
        DELETE FROM likesDislikes WHERE userID = ? AND postID = ?
```

Query Optimization

For query optimization, we ran some test on the queries we used with the "EXPLAIN EXTENDED" clause on the select statement. Here are the results:

```
mysql> EXPLAIN EXTENDED
    -> SELECT count(DISTINCT(articleID)) FROM writesArticle WHERE
userID = 6;
+----+-------------+--------------+------+--------------+----------
---+---------+-------+------+----------+-------------+
| id | select_type | table        | type | possible_keys | key
| key_len | ref   | rows | filtered | Extra       |
+----+-------------+--------------+------+--------------+----------
---+---------+-------+------+----------+-------------+
|  1 | SIMPLE      | writesArticle | ref  | userid_index  |
userid_index | 4       | const |    1 |   100.00 | Using index |
+----+-------------+--------------+------+--------------+----------
---+---------+-------+------+----------+-------------+
1 row in set, 1 warning (0.00 sec)

mysql> EXPLAIN EXTENDED
    -> SELECT articleID FROM articleKeyword WHERE keyword LIKE 'this'
ORDER BY occurences DESC;
+----+-------------+----------------+------+--------------+------+---
------+------+------+----------+-----------------------------+
| id | select_type | table          | type | possible_keys | key  |
key_len | ref  | rows | filtered | Extra                       |
+----+-------------+----------------+------+--------------+------+---
------+------+------+----------+-----------------------------+
|  1 | SIMPLE      | articleKeyword | ALL  | NULL          | NULL |
NULL    | NULL |  345 |   100.00 | Using where; Using filesort |
+----+-------------+----------------+------+--------------+------+---
------+------+------+----------+-----------------------------+
1 row in set, 1 warning (0.00 sec)

mysql> EXPLAIN EXTENDED
    -> SELECT userID from user WHERE username = 'chirag'
    -> ;
+----+-------------+-------+-------+-------------------------+--------
--+---------+-------+------+----------+-------------+
| id | select_type | table | type  | possible_keys           | key
| key_len | ref   | rows | filtered | Extra       |
+----+-------------+-------+-------+-------------------------+--------
--+---------+-------+------+----------+-------------+
|  1 | SIMPLE      | user  | const | username,username_index |
username | 32      | const |    1 |   100.00 | Using index |
+----+-------------+-------+-------+-------------------------+--------
--+---------+-------+------+----------+-------------+
1 row in set, 1 warning (0.00 sec)
```

```
mysql> show index from articleKeyword;

+----------------+------------+----------+--------------+-------------
+-----------+-------------+----------+--------+------+------------+---
------+---------------+
| Table          | Non_unique | Key_name | Seq_in_index | Column_name
| Collation | Cardinality | Sub_part | Packed | Null | Index_type |
Comment | Index_comment |
+----------------+------------+----------+--------------+-------------
+-----------+-------------+----------+--------+------+------------+---
------+---------------+
| articleKeyword |          0 | PRIMARY  |            1 | articleID
| A         |           2 |     NULL | NULL   |      | BTREE      |
|               |
| articleKeyword |          0 | PRIMARY  |            2 | keyword
| A         |         345 |     NULL | NULL   |      | BTREE      |
|               |
+----------------+------------+----------+--------------+-------------
+-----------+-------------+----------+--------+------+------------+---
------+---------------+
2 rows in set (0.00 sec)

mysql> EXPLAIN EXTENDED
    -> SELECT userID, comment, writtenAt, commentID FROM comment C
WHERE postID = 5 AND NOT EXISTS (SELECT * FROM commentThread CT WHERE
CT.childID = C.commentID) ORDER BY writtenAt;
+----+--------------------+-------+------+-----------------+---------
+---------+-------------------------------+------+----------+----------
-------------------+
| id | select_type        | table | type | possible_keys   | key     |
key_len | ref                           | rows | filtered | Extra
|
+----+--------------------+-------+------+-----------------+---------
+---------+-------------------------------+------+----------+----------
-------------------+
|  1 | PRIMARY            | C     | ref  | postID          | postID  |
4       | const                         |    1 |   100.00 | Using
where; Using filesort |
|  2 | DEPENDENT SUBQUERY | CT    | ref  | PRIMARY,childID | PRIMARY |
4       | mchirag_wikiblog.C.commentID |    1 |   100.00 | Using index
|
+----+--------------------+-------+------+-----------------+---------
+---------+-------------------------------+------+----------+----------
-------------------+
2 rows in set, 2 warnings (0.00 sec)

mysql> explain extended
    -> SELECT userID FROM likesDislikes WHERE postID = 5 AND choice =
1;
```

```
+----+-------------+---------------+------+---------------+------+----
-----+------+------+----------+-------------+
| id | select_type | table         | type | possible_keys | key  |
key_len | ref  | rows | filtered | Extra       |
+----+-------------+---------------+------+---------------+------+----
-----+------+------+----------+-------------+
|  1 | SIMPLE      | likesDislikes | ALL  | postID        | NULL |
NULL    | NULL |    5 |    80.00 | Using where |
+----+-------------+---------------+------+---------------+------+----
-----+------+------+----------+-------------+
1 row in set, 1 warning (0.00 sec)

mysql> explain extended
    -> SELECT count(*) from follows WHERE followee = 6
    -> ;
+----+-------------+---------+------+---------------+----------+------
---+-------+------+----------+-------------+
| id | select_type | table   | type | possible_keys | key      |
key_len | ref   | rows | filtered | Extra       |
+----+-------------+---------+------+---------------+----------+------
---+-------+------+----------+-------------+
|  1 | SIMPLE      | follows | ref  | followee      | followee | 4
| const |    2 |   100.00 | Using index |
+----+-------------+---------+------+---------------+----------+------
---+-------+------+----------+-------------+
1 row in set, 1 warning (0.01 sec)
```

As you can see, *every selection query has been optimized by using the proper index* available in the table. This was *done by adding indexes* to the table during its creation for fast retrieval afterwards.

Also, we have minimally used any joins or sub-queries which vastly reduce the performance of a query. This was done by breaking a complex query into two or more parts which is much faster in performance.

NORMALIZATION

During database formation we kept in mind for reducing any redundant information which crawled into our database design. This has been ensured so there is no overhead in keeping track of inconsistencies in the database. Thus ensuring a normalized database.

As seen in our relational schema attached at the end, you can easily deduce that there is no redundant information in the database.

But, for fast retrieval from database for articles, we have done some de-normalization and consequently have two copies of the latest available version of an article.