Project Portfolio 1

RAWDATA 2017

Project Report

Group Name/ Group number 8

Roskilde University
Computer Science Department RAWDATA 2017

Computer Science Department
Roskilde University
http://www.ruc.dk

Title:
Project Portfolio 1

Project Period:
Fall Semester 2017

Project Group:
8

Participant(s):
Eugenio Maria Capuani
Kidane Mahari Tesfai
Kristoffer Carl Schou
Rune Bastian Barrett

Page Numbers: 17

Date of Completion:
October 9, 2017

Abstract:

In this report we go over the design of a database to support a Stack Overflow viewing application (for later implementation). We will also go through the functionality of that application and the normalization of the database.

Contents

Chapter 1

Design

The initial schema contained two tables, namely Posts 1.1 and Comments. It was clear that the tables were badly designed, and as a result in order to avoid redundancy, this schema had to be normalized.This first section contains a break-down of our normalized database schema, followed by some design considerations.

1.1 The Post table

As it can be clearly seen from the figure, the post table 1.1 contains information about the the post and the user: this is not an appropriate way of storing data in the Database, as it lead to unnecessary data redundancy. As a result, the table has been decomposed in to the following two smaller tables: The *Post* table, which stores information only about the post, and the *User* tables stores information only about the user. In addition to that the two tables are linked using a foreign key, namely *owner_ user_ id*. The *Comments* table has been implemented in a similar way to how it looks in the original schema, but it now references our super-table *Post* through *post_ id* and *user_ id*.

post_id	creation_date	score	body	title	owner_user_id	type_id
19	2008-08-01 05:21:22	164	Solut...	13	1	
71	2008-08-01 13:38:00	43	<pHer...		49	2
531	2008-08-02 18:22:53	110	<p>The .		157	2
709	2008-08-03 14:53:54	28	<p>Il...	.NET Testi	111	1
712	2008-08-03 14:57:45	14	<p>Scrw.		91	2
713	2008-08-03 14:59:21	32	<p>Iht..		34	2
718	2008-08-03 15:07:20	5	<p><n...		27	2
798	2008-08-03 19:36:48	3	<code...		230	2
1053	2008-08-04 06:21:38	7	<part...	13	1	
1054	2008-08-04 06:22:19	5	<Sche...		13	2

Figure 1.1: The Posts table

user_id	user_name	user_creation_date	user_location	user_age
1	Jeff Atwood	2008-07-31 14:22:31	El Cerrito, CA	45
3	Jarrod Dixon	2008-07-31 14:22:31	New York, NY	36
4	Joel Spolsky	2008-07-31 14:22:31	New York, NY	
5	Jon Galloway	2008-07-31 14:22:31	San Diego, CA	45
13	Chris Jester-Young	2008-08-01 04:18:05	Raleigh, NC	35
17	Nick Berardi	2008-08-01 12:02:22	Collegeville, PA	35
20	Tom	2008-08-01 12:09:11	Atlanta, GA	
25	CodingWithoutComments	2008-08-01 12:15:23	Seattle, WA	35
26	Shawn	2008-08-01 12:18:15	San Francisco, CA	31

Figure 1.2: The User table

1.2 Schema structure

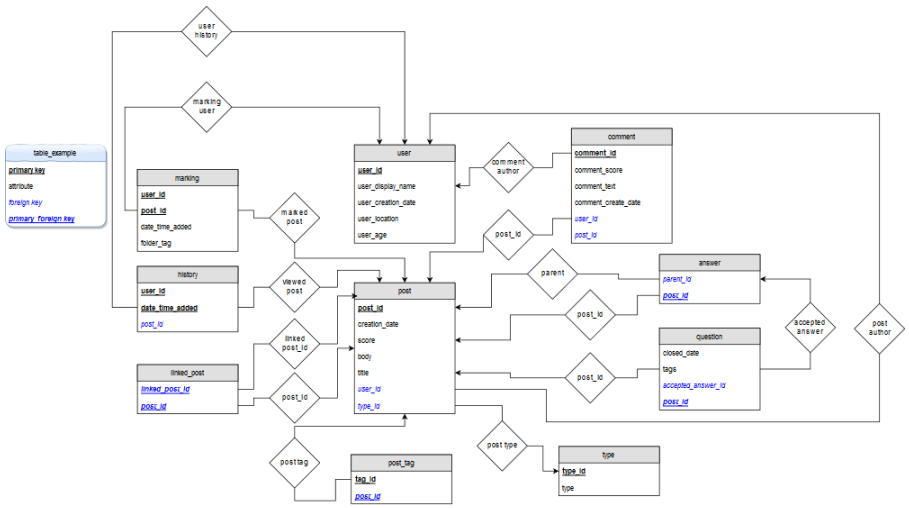

Figure 1.3: The ER model of our normalized schema

- *Post*

  The *Post* table is a super-table: it contains many of the elements common to Questions and Answers such as a *post_ id*, *body* and *title*. The *post_ id* attribute is used to link information from other tables to a given post.

- *User*

  The *User* table handles user data, It is referenced by other tables by the attribute *user_ id*, that identifies the author of a post.

- *Question*

The *Question* table is a specialization of the *Post* table, with the notable characteristic that it can have an accepted answer, in the form of a foreign key referencing a *post_ id*.

- *Answer*

  The *Answer* table is also derivate from the *Post* table, and is only being referenced by *post_ id*. The one distinctive attribute of the *Answer* Table is the *parent_ id* attribute, which is a post_id that defines what question any given answer is attached to.

- *Type*

  The *Type* table will be discussed in more detail in the Design section1.3, as it ties into one of the main design decision we did when refactoring the database from the original schema. For now, it should suffice to say that this class defines what kind of post there can be, either a question or an answer; every type has an associated *type_ id*.

- *comment*

  The comments table has been implemented such that it now references the super-table *Post* through *post_ id* and *user_ id*.

- *post_ tags*

  The *Post_ tag* table contains tags, which can be attached to a post; this table is very simple in its structure, but its important because in the original schema of the Stack Overflow database, tags were stored as a string containing multiple tags separated by "::", as shown in the following picture.

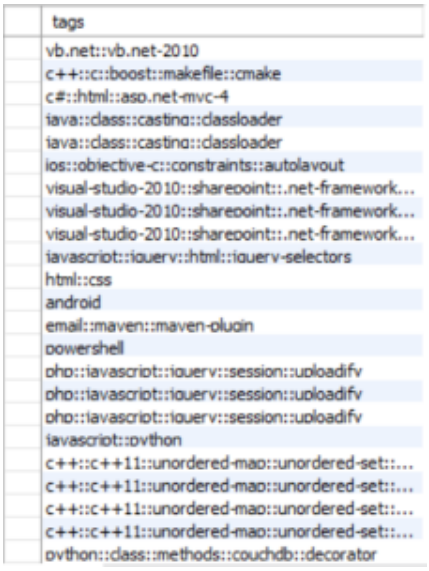

Figure 1.4: Figure of how tags are stored in the sample database schema, form SQL workbench output

.

Since we needed to be able to include specific tags in our search, this data structure didn't suit our needs. In order to obviate this problem, our database file contains a procedure that splits each row in the *tag* column of the original schema into substrings, divided at the "::", that are subsequently inserted into our table.This procedure is described in more detail in chapter 2.

- *History*

  The *History* table is the first of two tables that are needed to provide certain features in the software, in this case a history of the posts the user has viewed; This will most likely be a fixed-length list, and it will probably work as a stack: when the set limit is reached, the oldest post is deleted and the new one is added at the top of the list. The specific implementation of this feature will be done in software later on, but it will be supported by the database either via a special procedure or a trigger.

- *Marking*

  The *Marking* table supports marking functionality in the software, similar to bookmarks. The idea is for the user to be able to bookmark certain questions, and furthermore he should be able organize his bookmarks and group them

through the use of labels. To that end, the *label_tag* attribute stores the label of a given bookmark. As it can be seen from the table and schema, this attribute can only have one value for every post, we consider this a constraint of our design, the reason behind which is explained in 1.3

- *Linked_post*

  The *Linked_post* table contains linked posts, which are feature of the Stack Overflow website and dataset. In brief, users can put links to other posts in a question or an answer. The *Linked_post* table contains the *post_id* of the post the link is posted in, and it thus has a similar function to a parent_id, while the attribute *linked_post_id* contains the id of the post the link is pointing to.

1.3 Design

Our database design is centered around the super-table *Post*: As it can be seen from the diagram above, all other tables reference the *Post* table by *post_id*. During our design process we took an object oriented approach, and implemented questions, answers and comments as different kinds of posts.

Our schema is made up of 10 tables: two of them (*History* and *Marking*) support features to be implemented in software, while the others try to adapt the existing data-set from Stack Overflow. We tried to ensure compatibility with the existing data and the schema of the Stack Overflow database wherever possible, even if that meant adding complexity to our schema. An example of this can be seen in our implementation of the *Type* table and the associated *type_id*.

A *type_id* is an attribute that identifies a post as an either a question or an answer: this was necessary in the original database schema because there was no distinction between the two; Such a distinction is present in our schema however, and thus type_id attributes can be seen as redundant data: type could be reasonably be inferred by parent_id, since answers have a parent questions while questions do not.

Our schema could have been simplified by removing them, but we decided against it because the removal of type may have led to unintended consequences or loss of data, especially when one takes bigger and more complex data samples into account. we wanted to ensure that our schema was compatible with the data, as well as make our transition process easier. Another question that came up during the design process was how to handle labels for the bookmarking functionality: as explained in the previous section, we wanted to give the user the possibility to customize and

organize his bookmarks; in our initial design, this feature would have taken the form of a hierarchical structure of labels or "folders". However in the end we decided against this kind of hierarchical data-structure, because it would have made the table non-atomic. As a result, at present there can only be a single label associated with any given bookmark.

Chapter 2

Functionality

This chapter looks at the functions and procedures implemented in the database.

2.1 Creation of the database

Due to the way the original posts table was structured, the field *tags* would violate the first normal form by having multiple values together. This led us to write a procedure that splits the tag field into atomic values, and inserts these into a table called *post_tags*. (The structure is described in chapter 1).

The functions and procedures used for this are described in section 2.1.1.

2.1.1 Splitting up non atomic tags

num_values_in_delimited_str(tag)

This is a helper function that takes a string delimited by *'::'* and returns the number of values in it. Eg. *"value_a::value_b::value_c"* returns 3. This is used by the procedure seen in listing 2.3.

Listing 2.1: num_values_in_delimited_str(tag)

```
RETURN ROUND((LENGTH(tag) - LENGTH(REPLACE(tag, '::', ''))) /
   LENGTH('::')) + 1;
```

string_at_delimited_pos()

Another helper function is needed for extracting the value in a delimited string. This is used by the procedure seen in listing 2.3.

Listing 2.2: string_at_delimited_pos

```
RETURN REPLACE(SUBSTRING(SUBSTRING_INDEX(str, delim, pos),
       LENGTH(SUBSTRING_INDEX(str, delim, pos -1)) + 1),
       delim, '');
```

split_insert_into_tags()

This is the procedure that splits up tags. A cursor containing *post_id* and *tag* is created and looped over using a read loop. Each value in the currently read tag is then extracted by looping over the tag using the helper functions 2.1 and 2.2, and inserting the returned value into the table *post_tag*.

Listing 2.3: split_insert_into_tags()

```
    open tags_cur;
        read_loop: loop
            fetch tags_cur into post_id, tag;
            set i_max = num_values_in_delimited_str(tag);
            set i = 1;
            if done then
                leave read_loop;
            end if;
            start transaction;
                while i <= i_max do
                    insert ignore into post_tags(post_id, tag_name
                       ) values (post_id, (string_at_delimited_pos
                       (tag, "::", i)));
                    set i=i+1;
                end while;
            commit;
        end loop;
    close tags_cur;
```

2.2 API for querying the database

We have introduced various functions that we expect will be useful as an API when developing an application using this database. Mainly these consists of search queries, and queries for listing data from different tables.

2.2.1 Searching

We currently have a few ways of searching through the posts, whose purpose and implementation will be described below.

search_questions_by_tag

This procedure simply queries all questions that has the given tag, and orders them by score. It seems likely that such a procedure could be useful when developing an API.

Listing 2.4: search_questions_by_tag

```
BEGIN
    select post.post_id, title, body, score, creation_date,
       closed_date
    from post, question, post_tags
    where post.post_id = question.post_id and post.post_id =
       post_tags.post_id and post_tags.tag_name = tag
    order by post.score desc limit lim;
END //
```

fulltext_search

This functions performs a fulltext search in question.title and question.body with the given string. It makes use of mysql's implementation of natural language fulltext search.

Listing 2.5: fulltext_search

```
BEGIN
    SELECT post_id, title, body, match (title,body) AGAINST
    (search_str IN NATURAL LANGUAGE MODE)  AS score
    FROM post WHERE post.type_id = post_type and MATCH (title,body
       ) AGAINST
    (search_str IN NATURAL LANGUAGE MODE);
END //
```

Searching_Question

Finds questions that matches the last meaningful word of the input string, where that word must be in the title.

Listing 2.6: search_question

```
begin
    declare done int default false;
    declare a char(200);

    declare cur1 cursor for (
        select tag_name
        from post_tags
        where match(tag_name) against(inpute IN BOOLEAN MODE));
    declare continue handler for not found set done = true;

    open cur1;
        read_loop: loop
            fetch cur1 into a;
            leave read_loop;
        end loop;
    close cur1;

    select post_id, type_id, title, body, score from
    (select * from post_tags where match(tag_name) against(a IN
       BOOLEAN MODE)) as t
    natural join post
    where match(post.title) against(+a IN BOOLEAN MODE)
    and match(post.body) against(inpute in natural language mode)
    order by post.score desc;

end;//
```

2.2.2 History, Marking and Listing Data

add_marking

Inserts a marking to a given post for a given user. We have included a 'marking_label here as well, with the purpose of using that to organize markings using labels.

Listing 2.7: add_marking

```
BEGIN
    insert into marking values (user_id, post_id, now(),
       marking_label);
END //
```

retrieve_answers

Retrieves answers to a given question.

Listing 2.8: retrieve_answers

```
BEGIN
    select */*post.post_id, title, body, score, creation_date*/
       from post, answer
    where post.post_id = answer.post_id
    order by post.score desc limit lim;
END //
```

Chapter 3

Normalization of the database

In this chapter we will go through the normalization of the data from the Stack Overflow sample. This will include going trough first, second, third and Boyce-Codd normal form, as well as showing the Entity-relationship model 3.2 of the normalized dataset.

3.1 The data sample

The original structure of the Stack Overflow sample stored all the data in to tables (as seen below) with a lot of redundant entries, like how all posts and comments have all data about the user that made them.

- *Comments(commentid, postid, commentscore, commenttext, commentcreate-date, userid, userdisplayname, usercreationdate, userlocation, userage)*
- *Posts(id, posttypeid, parentid, acceptedanswerid, creationdate, score, body, closed-date, title, tags, owneruserid, owneruserdisplayname, ownerusercreationdate, owneruserlocation, owneruserage, linkpostid)* [1]

In order to facilitate the history and marking functionality of the Stack Overflow Viewer application we added two more tables (shown below).

- *History(user_ id, date_ time_ added, post_ id)*
- *Marking(user_ id, post_ id, date_ time_ added, folder_ tag)*

To avoid having redundant entries in the data we normalized the structure of the database, to fit first, second, third and Boyce-Codd normal form.

3.2 Normal forms

3.2.1 First normal form

A table is in first normal form if it's atomic. Meaning that it doesn't contain any repeating attributes or groups of attributes.

The data set was already close to bing complaint with first normal form and only needed to have the tags split into a separate table.

3.2.2 Second normal form

A table is in second normal form when there's no attributes in it that depend on part of a concatenated key. Meaning that if the table has a primary key that contains two attributes no attribute in the table depends on only one attribute of the key.

The data was already compliant with the second normal form, without furtherer work, as there were no partial dependencies.

3.2.3 Third normal form

A table is in third normal form when no attributes depend on a non-key attribute.

To be compliant with third normal form we removed the transitive dependencies, like how the user attributes are dependent on *user_ id* witch in turn is dependent on *post_ id*. This was done by extracting the user information from *Posts* and *Comments* and storing this information in a new table called *User* and splitting the *Posts* table into three separate tables: *Post*, as a super-table, and *Answer* and *Question*, as sub-tables.

- *User(user_ id, user_ name, user_ creation_ date, user_ location, user_ age)*
- *Post(post_ id, creation_ date, score, body, title, owner_ user_ id, type_ id)*
- *Question(accepted_ answer_ id, closed_ date, post_ id)*
- *Answer(parent_ id, post_ id)*
- *Comment(comment_ id, comment_ score, comment_ text, comment_ create_ date, user_ id, post_ id)*
- *Post_ tags((post_ id, tag)*
- *Linked_ posts(link_ post_ id, post_ id)*
- *History(user_ id, date_ time_ added, post_ id)*
- *Marking(user_ id, post_ id, date_ time_ added, folder_ tag)*

3.2.4 Boyce-Codd normal form

Boyce-Codd normal form is a stricter version of third normal form, that ensures that all redundancy from functional dependencies is removed. This is done by requiring that all dependences are either trivial functional dependencies, meaning that one side of the equation is a subset of the other side, or that the dependency is on the primary key.

We used the Boyce-Codd Decomposition Algorithm (seen below) to ensure that there was no redundancy left from the functional dependencies.

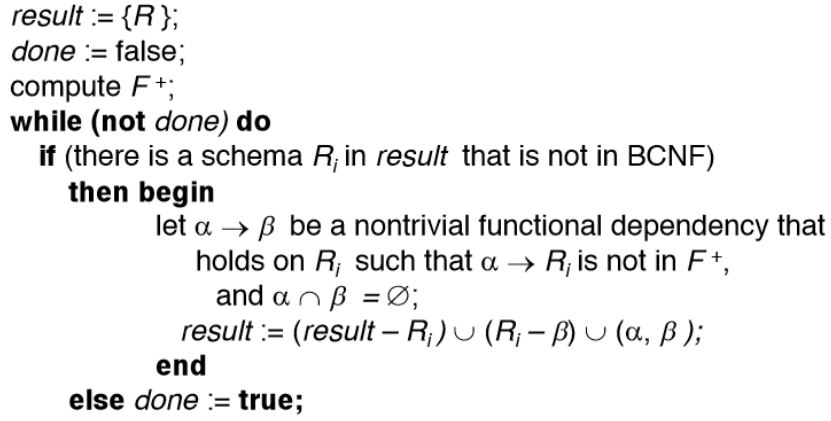

Figure 3.1: The Boyce-Codd Decomposition Algorithm

This resulted in a database structure like this:

- *User(user_id, user_name, user_creation_date, user_location, user_age)*
- *Post(post_id, creation_date, score, body, title, owner_user_id, type_id)*
- *Question(accepted_answer_id, closed_date, post_id)*
- *Answer(parent_id, post_id)*
- *Comment(comment_id, comment_score, comment_text, comment_create_date, user_id, post_id)*
- *Post_tags(post_id, tag)*
- *Linked_posts(link_post_id, post_id)*
- *History(user_id, date_time_added, post_id)*
- *Marking(user_id, post_id, date_time_added, folder_tag)*

3.3 Entity-relationship model

After normalization of the dataset we visualized it with an ER-model 3.2, both for later reference and to help keep an overview of the dataset.

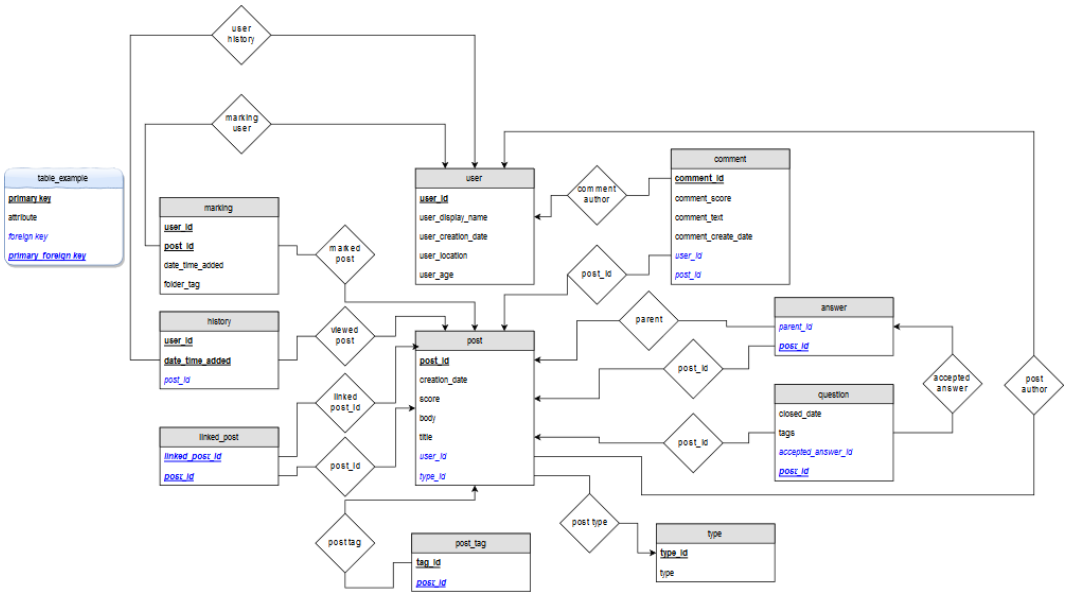

Figure 3.2: Entity-relationship model

Bibliography

[1] Troels Andreasen & Henrik Bulskov. *RAWDATA Portfolio Subproject 1 Requirements*. `https://moodle.ruc.dk/`. 2017.

Appendix A

Initial design document

During design and normalization we worked on the tables in the following text document.

```
user (user_id(PK), user_name, user_creation_date, user_location, user_age)

post(post_id(PK), creation_date, score, body, title, owner_user_id(FK), type_id) /* type id
    somewhat redundant */

question(accepted_answer_id(FK), closed_date, tags, _post_id(PK,FK))
answer(parent_id, post_id(PK,FK))

comment(comment_id(PK), comment_score, comment_text, comment_create_date, user_id(FK), post_id(FK)
    )

post_tags((post_id(FK), tag)(PK))//tag_id
//tags(TAG_ID, tag)  <split tags on '::'>
```

```
//post_type(type_id, type); /* redundant? a post with a parentid is an answer, other posts are
   questions */ /* could remove and store type directly in the post table*/

history((USER_ID, DATE_TIME_ADDED)(PK), post_id(FK)); /*Index by users for search*/

marking(USER_ID, POST_ID, date_time_added, folder_tag);

linked_posts(LINK_POST_ID, POST_ID)

-------------------------

//In posts(), index questions aside from answers
```

Appendix B

stack_overflow_normalized.sql

During design and normalization we worked on the tables in the following text document.

```
DROP DATABASE if exists stack_overflow_normalized;
CREATE DATABASE stack_overflow_normalized;
USE stack_overflow_normalized;

-- ---------------- DATABASE CREATION AND DATA INSERTION -------------------

-- user (user_id(PK), user_name, user_creation_date, user_location, user_age)
CREATE TABLE user (
    user_id INT UNSIGNED PRIMARY KEY,
    user_name VARCHAR(30) NOT NULL,
    user_creation_date DATETIME,
    user_location VARCHAR(200),
    user_age INT UNSIGNED
);
```

```sql
insert into user (user_id, user_name, user_creation_date, user_location, user_age)
select distinct owneruserid, owneruserdisplayname, ownerusercreationdate, owneruserlocation,
   owneruserage
from stackoverflow_sample_universal.posts
union
select distinct userid, userdisplayname, usercreationdate, userlocation, userage
from stackoverflow_sample_universal.comments;

-- insert ignore into user (user_id, user_name, user_creation_date, user_location, user_age)
-- select userid, userdisplayname, usercreationdate, userlocation, userage
-- from stackoverflow_sample_universal.comments;

-- post(post_id(PK), creation_date, score, body, title, owner_user_id(FK), type_id) /* type id
   somewhat redundant */
drop table if exists post;
CREATE TABLE post (
    post_id INT UNSIGNED PRIMARY KEY,
    creation_date DATETIME,
    score BIGINT,
    body TEXT,
    title VARCHAR(300),
    owner_user_id INT UNSIGNED NOT NULL REFERENCES user (user_id),
    type_id INT UNSIGNED,
    FULLTEXT (title,body), -- used by: fulltext_search
    FULLTEXT (body), -- used by: Searching_Questions
    FULLTEXT (title)-- used by: Searching_Questions
);

insert into post (post_id, creation_date, score, body, title, owner_user_id, type_id)
select distinct id, creationdate, score, body, title, owneruserid, posttypeid
from stackoverflow_sample_universal.posts;
```

```
-- question(post_id(PK,FK), accepted_answer_id(FK), closed_date, tags) /* no need for tags here
   right? */
CREATE TABLE question (
    post_id INT PRIMARY KEY REFERENCES post (post_id),
    accepted_answer_id INT REFERENCES post (post_id),
    closed_date DATETIME
);

insert into question (post_id, accepted_answer_id, closed_date)
select distinct id, acceptedanswerid, closeddate
from stackoverflow_sample_universal.posts as p where p.posttypeid = 1;


-- answer(parent_id, post_id(PK,FK))
CREATE TABLE answer (
    post_id INT PRIMARY KEY REFERENCES post (post_id),
    parent_id INT
);

insert into answer (post_id, parent_id)
select distinct id, parentid
from stackoverflow_sample_universal.posts as p where p.posttypeid = 2;

-- comment(comment_id(PK), comment_score, comment_text, comment_create_date, user_id(FK), post_id(
   FK))
CREATE TABLE comment (
    comment_id INT PRIMARY KEY,
    comment_score INT,
    comment_text MEDIUMTEXT,
    comment_create_date DATETIME,
```

```
    user_id INT REFERENCES user(user_id),
    post_id INT references post(post_id)
);
insert into comment (comment_id, comment_score, comment_text, comment_create_date, user_id,
   post_id)
select commentid, commentscore, commenttext, commentcreatedate, userid, postid
from stackoverflow_sample_universal.comments;

-- post_tags((post_id(FK), tag)(PK))//tag_id
drop table if exists post_tags;
CREATE TABLE post_tags (
    post_id INT REFERENCES post (post_id),
    tag_name VARCHAR(50),
    primary key(post_id, tag_name),
    fulltext(tag_name) -- used in Searching_Questions
);

-- tags(TAG_ID, tag)  <split tags on '::'>
/*drop table if exists tags;
CREATE TABLE tags (
    tag_id INT AUTO_INCREMENT PRIMARY KEY,
    tag_name VARCHAR(50)
);*/

-- string_at_delimited_pos(str, delim, pos) /* returns the value at n'th position eg.
   string_at_delimited_pos("a::b::c::d", "::", 3) returns "c" */
-- drop function if exists string_at_delimited_pos;
 CREATE FUNCTION string_at_delimited_pos(str VARCHAR(255), delim VARCHAR(12), pos INT)
 RETURNS VARCHAR(255)
 RETURN REPLACE(SUBSTRING(SUBSTRING_INDEX(str, delim, pos),
        LENGTH(SUBSTRING_INDEX(str, delim, pos -1)) + 1),
```

```
        delim, '');
-- select string_at_delimited_pos("qwer::rtyu::khhj","::",2);

-- num_values_in_delimited_str(str) /* Returns the number of values such that a::b::c returns 3*/
DELIMITER //
CREATE FUNCTION num_values_in_delimited_str ( tag varchar(200) )
RETURNS INT
BEGIN
   RETURN ROUND((LENGTH(tag) - LENGTH(REPLACE(tag, '::', ''))) / LENGTH('::')) + 1;
END; //
DELIMITER ;

/* This procedure creates a cursor containing id, tag from the original 'posts' table,
 iterates over that splitting up each tag inside the original 'tags' string, and inserts them
 into the post_tag table */
-- drop procedure if exists split_insert_into_tags;
delimiter //
create procedure split_insert_into_tags()
begin
    declare done int default false;
    declare tag varchar(90);
    declare post_id int;
    declare i_max int;
    declare i int;

    declare tags_cur cursor for select id, tags from stackoverflow_sample_universal.posts;
    declare continue handler for not found set done = true;

    open tags_cur;
        read_loop: loop
            fetch tags_cur into post_id, tag;
```

```
            set i_max = num_values_in_delimited_str(tag);
            set i = 1;
            if done then
                leave read_loop;
            end if;
            start transaction;
                while i <= i_max do
                    insert ignore into post_tags(post_id, tag_name) values (post_id, (
                       string_at_delimited_pos(tag, "::", i)));
                    set i=i+1;
                end while;
            commit;
        end loop;
    close tags_cur;
end;//
delimiter ;

call split_insert_into_tags();

-- linked_posts(LINK_POST_ID, POST_ID)
CREATE TABLE linked_posts (
    post_id INT REFERENCES post (post_id),
    link_post_id INT REFERENCES post (post_id),
    PRIMARY KEY (link_post_id , post_id)
);

insert into linked_posts (post_id, link_post_id)
select id, linkpostid
from stackoverflow_sample_universal.posts where linkpostid > "";

-- history((USER_ID, DATE_TIME_ADDED)(PK), post_id(FK)); /*Index by users for search*/
```

```
CREATE TABLE history (
    user_id INT REFERENCES user(user_id),
    datetime_added DATETIME,
    link_post_id INT REFERENCES post (post_id),
    PRIMARY KEY (user_id, datetime_added)
);
-- marking(USER_ID, POST_ID, date_time_added, folder_tag);
CREATE TABLE marking (
    user_id INT REFERENCES user(user_id),
    post_id INT REFERENCES post(post_id),
    datetime_added DATETIME,
    folder_tag varchar(200),
    PRIMARY KEY (user_id, post_id) /* added post_id to the PK instead of datetime_added. no need
        for multiple marks to one post */
);
-- //post_type(type_id, type); /* somewhat redundant. a post with a parentid is an answer, posts
   without are questions */ /* could also remove and store type directly in the post table*/

-- ---------------- FUNCTIONALITY / API -------------------

-- search_questions_by_tag(tag, lim) /* Basic tag search query. Returns relevant data about
   questions that contains the <tag>, ordered by score, limited to <lim> */
-- drop procedure if exists search_questions_by_tag;
DELIMITER //
CREATE PROCEDURE search_questions_by_tag (IN tag varchar(200), lim int)
BEGIN
    select post.post_id, title, body, score, creation_date, closed_date
    from post, question, post_tags
    where post.post_id = question.post_id and post.post_id = post_tags.post_id and post_tags.
        tag_name = tag
    order by post.score desc limit lim;
```

```
END //
DELIMITER ;
-- call search_questions_by_tag("c#", 50);

-- retrieve_answers(question_id) /* Retrieves the answers to a given question */
-- drop procedure if exists retrieve_answers;
DELIMITER //
CREATE PROCEDURE retrieve_answers (IN question_id int, lim int)
BEGIN
    select */*post.post_id, title, body, score, creation_date*/ from post, answer
    where post.post_id = answer.post_id
    order by post.score desc limit lim;
END //
DELIMITER ;
-- call retrieve_answers(9033, 50);

-- fulltext_search(search_str) /* Procedure that finds questions using mysql's built in fulltext
   search (ignoring useless words, using multiword strings), searching in both title and body */
drop procedure if exists fulltext_search;
DELIMITER //
CREATE PROCEDURE fulltext_search (in search_str varchar(400), post_type int)
BEGIN
    SELECT post_id, title, body, match (title,body) AGAINST
    (search_str IN NATURAL LANGUAGE MODE)  AS score
    FROM post WHERE post.type_id = post_type and MATCH (title,body) AGAINST
    (search_str IN NATURAL LANGUAGE MODE);
END //
DELIMITER ;
-- call fulltext_search('mysql', 1);
-- call fulltext_search('mysql', 2);
-- call fulltext_search('javascript tutorial',1);
```

```
-- call fulltext_search('machine learning',1);
-- call fulltext_search('how to python good',1);
-- call fulltext_search('Hi database teach me to be the bestest at searching thank you bye bye',1)
   ;

-- add_marking(user_id, post_id, marking_label) /* Inserts a marking to at given post <post_id>,
   for a given user <user_id>, with a given folder name <marking_label> and a <now()> timestamp*/
-- drop procedure if exists add_marking;
DELIMITER //
CREATE PROCEDURE add_marking (IN user_id int, post_id int, marking_label varchar(200))
BEGIN
    insert into marking values (user_id, post_id, now(), marking_label);
END //
DELIMITER ; /* todo: handle the user inserting multiple identical marks. gives duplicate error now
   . insert ignore would do it but its probably bad design*/
-- call add_marking(1185, 9033, 'MyFolder');
-- select * from marking;

-- Searching_Questions /* Finds questions that matches the last meaningful word of the input
   string, where that word must be in the title. */
drop procedure if exists Searching_Questions;
delimiter //
create procedure Searching_Questions( in inpute char (200))
begin
    declare done int default false;
    declare a char(200);

    declare cur1 cursor for (
        select tag_name
        from post_tags
        where match(tag_name) against(inpute IN BOOLEAN MODE));
```

```
    declare continue handler for not found set done = true;

    open cur1;
        read_loop: loop
            fetch cur1 into a;
            leave read_loop;
        end loop;
    close cur1;

    select post_id, type_id, title, body, score from
    (select * from post_tags where match(tag_name) against(a IN BOOLEAN MODE)) as t
    natural join post
    where match(post.title) against(+a IN BOOLEAN MODE)
    and match(post.body) against(inpute in natural language mode)
    order by post.score desc;

end;//
delimiter ;


-- Search by %word% in title
-- Search by tag in body
```

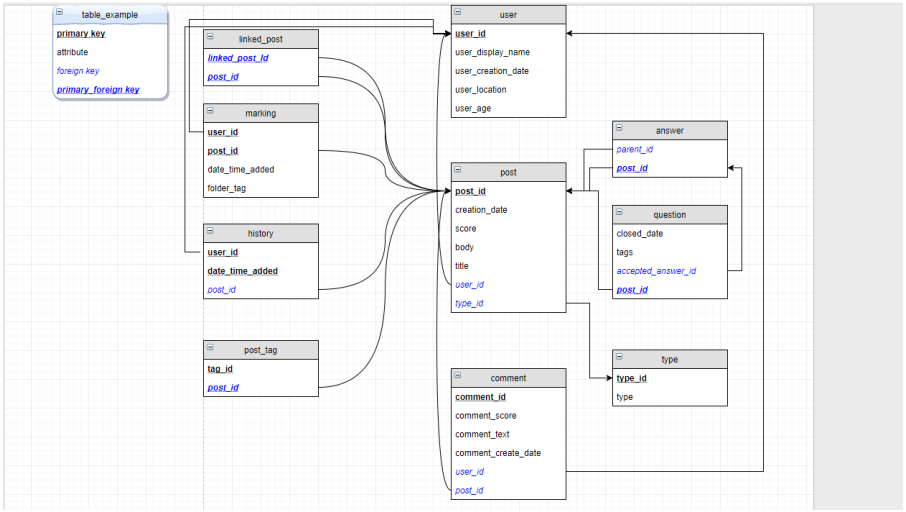

Figure B.1: ER model

id	posttypeid	parentid	acceptedanswerid	creationdate	score	body	closeddate	title	tags	owneruserid	owneruserdisplayna	ownerusercreati	owneruserloca	owneruserage	linkpostid
1364...	1	NULL	15831991	2012-11-30 1...	2	<p>I'm setting my console full scree...	NULL	Hide ...	vb.ne...	1365365	Jamie H	2012-04-30 0...	NULL	19	1165879
1932...	1	NULL	19329766	2013-10-12 0...	0	<p>My project uses boost while the...	NULL	How t...	c++;...	641235	xunzhang	2010-10-11 1...	Beijing, China	NULL	7855174
1739...	1	NULL	NULL	2013-06-30 2...	-6	<p>I was just wondering if it is pos...	2013-07-0...	Using ...	c#;h...	2514148	kost	2013-06-23 1...	United States	NULL	372865
2658...	1	NULL	NULL	2014-10-27 0...	2	<p>I wrote my own classloader. wh...	NULL	Retur...	java;...	3204194	tenius	2014-01-16 2...	Russia	NULL	234600
2658...	1	NULL	NULL	2014-10-27 0...	2	<p>I wrote my own classloader. wh...	NULL	Retur...	java;...	3204194	tenius	2014-01-16 2...	Russia	NULL	2215843
1802...	1	NULL	18060672	2013-08-02 1...	0	<p>Edition:

<p>... | NULL | warni... | ios;o... | 1314522 | Sudha Tiwari | 2012-04-05 0... | Ludhiana, I... | 25 | 15874806 |
2067...	1	NULL	20674639	2013-12-18 2...	0	<p>I want to make an event reciev...	NULL	Event...	visual...	1223297	Mahmoud Favez	2012-02-21 1...	Cairo, Egypt	47	2923210
2067...	1	NULL	20674639	2013-12-18 2...	0	<p>I want to make an event reciev...	NULL	Event...	visual...	1223297	Mahmoud Favez	2012-02-21 1...	Cairo, Egypt	47	8161183
2067...	1	NULL	20674639	2013-12-18 2...	0	<p>I want to make an event reciev...	NULL	Event...	visual...	1223297	Mahmoud Favez	2012-02-21 1...	Cairo, Egypt	47	18588803
1582...	1	NULL	15821264	2013-04-04 2...	5	<p>I get the following console mes...	NULL	tOuer...	lavas...	171461	lildupont	2009-09-10 1...	Montreal, C...	44	7815951
2347...	1	NULL	NULL	2014-05-05 1...	1	<p>I came upon an <a href="http:...	NULL	Are t...	html;...	78739	JMS10	2009-03-16 1...	NULL	NULL	83073
1833...	1	NULL	NULL	2013-08-20 1...	-1	<p>in my code image1 is show on le...	NULL	how t...	androd	2686011	user2686011	2013-08-15 1...	NULL	NULL	18330788
9599...	1	NULL	NULL	2012-03-07 1...	1	<p>After performing a '<code>mv...	NULL	move...	email;...	691197	user691197	2011-04-04 1...	NULL	NULL	1243326
2514...	1	NULL	25146501	2014-08-05 1...	0	<p>I want to search the content of...	NULL	Powe...	powe...	3911596	user3911596	2014-08-05 1...	NULL	NULL	17144355
5158...	1	NULL	5158681	2011-03-01 1...	2	<p>Ok, so I've just setting started ...	NULL	tOuer...	php;...	621855	Henrikz	2011-02-17 1...	NULL	NULL	1284666
5158...	1	NULL	5158681	2011-03-01 1...	2	<p>Ok, so I've just setting started ...	NULL	tOuer...	php;...	621855	Henrikz	2011-02-17 1...	NULL	NULL	3499387
5158...	1	NULL	5158681	2011-03-01 1...	2	<p>Ok, so I've just setting started ...	NULL	tOuer...	php;...	621855	Henrikz	2011-02-17 1...	NULL	NULL	4291095
1192...	1	NULL	11925753	2012-08-12 2...	1	<blockquote>
<p><strong...	2012-08-1...	Is it p...	lavas...	975097	Anderson Green	2011-09-02 0...	NULL	NULL	1562759
2436...	1	NULL	24362641	2014-06-23 0...	9	<p>The Question</stron...	NULL	How t...	c++;...	1128289	Praxeolitic	2012-01-03 1...	United States	26	2590677
2436...	1	NULL	24362641	2014-06-23 0...	9	<p>The Question</stron...	NULL	How t...	c++;...	1128289	Praxeolitic	2012-01-03 1...	United States	26	7110301
2436...	1	NULL	24362641	2014-06-23 0...	9	<p>The Question</stron...	NULL	How t...	c++;...	1128289	Praxeolitic	2012-01-03 1...	United States	26	8157937
2436...	1	NULL	24362641	2014-06-23 0...	9	<p>The Question</stron...	NULL	How t...	c++;...	1128289	Praxeolitic	2012-01-03 1...	United States	26	14813965
1435...	1	NULL	14353140	2013-01-16 0...	2	<p>As i mentioned in <a href="htt...	NULL	Pytho...	pytho...	1833591	Thomas Sovcher	2012-11-18 1...	Switzerland	31	14333553
1308...	1	NULL	NULL	2012-10-26 0...	5	<p>I am a vouno developer on And...	NULL	What...	andro...	1776447	Julien Renard	2012-10-26 0...	NULL	NULL	10231323
2673...	1	NULL	NULL	2014-11-04 1...	1	<p>I have a class that implements ...	NULL	Parcel...	andro...	1439522	prom85	2012-06-06 1...	NULL	NULL	4853952
1326...	1	NULL	1326242	2009-08-25 0...	10	<p>I'm working on a new WPF appli...	NULL	Openi...	mvvm...	156789	JasonD	2009-08-15 0...	Denver, CO	43	1321346
8172...	1	NULL	8173415	2011-11-17 1...	1	<p>I'm trying to create a SOL state...	NULL	How t...	sql;s...	935476	rishimaharaj	2011-09-08 1...	Pittsburgh, PA	31	5067684
2819...	1	NULL	28194950	2015-01-28 1...	1	<p>Im trying to write code to recur...	NULL	How t...	pytho...	4464936	user4464936	2015-01-17 1...	NULL	NULL	13591970
2819...	1	NULL	28194950	2015-01-28 1...	1	<p>Im trying to write code to recur...	NULL	How t...	pytho...	4464936	user4464936	2015-01-17 1...	NULL	NULL	23087820
1094...	1	NULL	10940647	2012-06-07 2...	1	<p>If I have array. for example : ...	NULL	How t...	lavas...	508377	Adham	2010-11-15 1...	Gaza City, ...	28	3954438
2312...	1	NULL	NULL	2014-04-17 0...	0	<p>I have read several articles abo...	NULL	Detail...	java;...	3544305	user3544305	2014-04-17 0...	NULL	NULL	19147235
2818...	1	NULL	28187450	2015-01-28 0...	-2	<p>I have the following code that s...	NULL	Check...	c#;d...	4273361	Tal	2014-11-20 0...	NULL	NULL	531384
2818...	1	NULL	28187450	2015-01-28 0...	-2	<p>I have the following code that s...	NULL	Check...	c#;d...	4273361	Tal	2014-11-20 0...	NULL	NULL	22683040
2180...	1	NULL	2180507	2010-02-01 2...	3	<p>Anyone know how to get Sorin...	NULL	Sorin...	java;...	243494	Kyle	2010-01-04 2...	Cupertino, CA	29	1620823
1262...	1	NULL	12622363	2012-09-27 1...	16	<pre><code>foreach (var row in t...	NULL	Use o...	c#;d...	638064	christofr	2011-02-28 1...	NULL	NULL	2325777
1262...	1	NULL	12622363	2012-09-27 1...	16	<pre><code>foreach (var row in t...	NULL	Use o...	c#;d...	638064	christofr	2011-02-28 1...	NULL	NULL	2786727
2030...	1	NULL	20306708	2013-11-30 2...	3	<p>I'm trying to get the last modifi...	NULL	Get L...	c#;d...	1116831	Wilson	2011-12-26 2...	NULL	NULL	NULL
1571...	1	NULL	NULL	2013-03-29 2...	2	<p>I tried to build <a href="http://...	NULL	Compi...	compi...	987846	kakyo	2011-10-10 1...	NULL	NULL	NULL

Figure B.2: Original post table

commentid	postid	commentscore	commenttext	commentcreatedate	userid	userdisplayname	usercreationdate	userlocation	userage
18728068	13649012	3	This mav help: http://stackoverflow.com/a/116...	2012-11-30 16:53:35	62576	Ken White	2009-02-04 19:31:58	United States	NULL
29025325	19329707	0	possible duplicate of fSet minimum version of bo...	2013-10-24 07:46:52	641235	xunzhang	2010-10-11 16:35:48	Beijing, China	NULL
2782235	2751704	0	Wouldn't that interfere if there was a user calle...	2010-05-01 22:10:09	20862	Ignacio Vazquez-Abrams	2008-09-23 02:06:20	NULL	NULL
2782240	2751704	0	If either system was in place I would use the oo...	2010-05-01 22:11:21	234631	Stefan	2009-12-18 15:29:10	Edinburgh, United Kingdom	26
25255117	17394734	1	What do you mean, is that possible? Of course i...	2013-06-30 20:58:44	218589	Klaus Byskov Pedersen	2009-11-25 13:19:42	Denmark	39
25255630	17394734	0	I'm pretty new to this.. I tried doing it using a f...	2013-06-30 21:33:44	2514148	kost	2013-06-23 18:10:19	United States	NULL
31044642	20722398	0	The `task.oet()` call is blocking. Anyway, you d...	2013-12-21 19:17:12	1321716	Mattias Buelens	2012-04-09 11:01:42	Belgium	23
41782591	26583319	0	whats the exception thrown	2014-10-27 08:13:03	2717954	user2717954	2013-08-26 11:49:45	NULL	NULL
41782719	26583319	0	loadClass just loads the class. You then need to...	2014-10-27 08:18:26	126769	nos	2009-06-22 08:06:23	Norway	NULL
9158912	7556427	0	What type of application is this? Web or deskto...	2011-09-26 14:17:33	203479	Mike Mooney	2009-11-05 14:07:48	Chester Springs, PA	37
9158957	7556427	0	C# and Web App.	2011-09-26 14:19:42	426733	gabrielVa	2010-08-20 20:15:53	NULL	NULL
26358921	18021406	0	It would be more useful to post the code that s...	2013-08-02 17:00:59	1285133	rdelmar	2012-03-22 06:04:43	Sebastopol, CA	NULL
26425871	18021406	0	@rdelmar all the views and constraints are in xib.	2013-08-05 14:05:25	1314522	Sudha Tiwari	2012-04-05 06:29:17	Ludhiana, India	25
32450822	21497037	0	Yes, it is, since chunking breaks uploads in inde...	2014-02-01 10:46:36	1248114	arkascha	2012-03-04 13:18:31	NULL	NULL
33534974	22106846	4	It's C#, don't worry at all about "deletino" your ...	2014-02-28 22:14:17	2316200	Pierre-Luc Pineault	2013-04-24 15:08:34	Quebec, Canada	26
33535006	22106846	0	"Resources" refers to unmanaged resources, i....	2014-02-28 22:15:49	50079	Jon	2008-12-30 05:05:53	Berlin	36
33535038	22106846	0	You can manually call GC.collect() but it's not re...	2014-02-28 22:16:53	2636891	KRUKUSA	2013-07-31 06:48:59	NULL	NULL
33535113	22106846	0	Rather, I want to remove any reference to the ...	2014-02-28 22:19:49	2808956	FatalSleep	2013-09-23 22:27:50	NULL	NULL
33535393	22106846	1	Probably a more relevant question - why do yo...	2014-02-28 22:30:12	2453802	Rick Daley	2013-06-05 00:06:22	NULL	NULL
33535505	22106846	0	E.g. let's say an instance of a class represents ...	2014-02-28 22:34:23	2808956	FatalSleep	2013-09-23 22:27:50	NULL	NULL
33535659	22106846	0	C# was designed to avoid manual management...	2014-02-28 22:40:30	1974021	DasKrümelmonster	2013-01-13 11:47:50	Germany	NULL
33535744	22106846	0	In case you desperately crave such a mechanis...	2014-02-28 22:44:07	1974021	DasKrümelmonster	2013-01-13 11:47:50	Germany	NULL
33535790	22106846	0	Well, it's not removing it myself. It's removing r...	2014-02-28 22:46:19	2808956	FatalSleep	2013-09-23 22:27:50	NULL	NULL
33536370	22106846	1	Check out reachability section of the Wikipedia ...	2014-02-28 23:12:20	477420	Alexei Levenkov	2010-10-15 20:09:54	NULL	NULL
10135016	8239282	0	Im not sure. but if you think about it: letters ar...	2011-11-23 08:49:52	1037066	Grrbrr404	2011-11-09 06:50:24	NULL	NULL

Figure B.3: Original comments table