DataBases Report Project

Blog Management

Group: 8

University of Science and Technology of Hanoi

December,2020

TABLE OF CONTENT

1 Introduction

1.1 Overview

A blog is a private information website or online diary, with the presentation of the latest articles brought to the fore. Bloggers can be individuals or a small group, express their subjective view of a certain topic, write about things they like. As a result, a blog needs to have an efficient database to manage information and help users in the browsing and reading process.

In this project, our team developed a simple database design for a blog. We have also written functions so that stakeholders can interact with the site, such as administrators who can upload and edit information, etc. Details of the design and functionality of the database will be presented in the following sections.

1.2 Group members

Đoàn Văn Chương	BA9-008
Nguyễn Văn Cường	BA9-011
Đào Hải Long	BA9-041
Đoàn Đình Nam	BA9-045
Đặng Hoàng Phúc	BA9-050

2 User requirements

To create an effective database, the first step is to collect user requests.For a blog, there are two main types of clients that are administrators and readers.

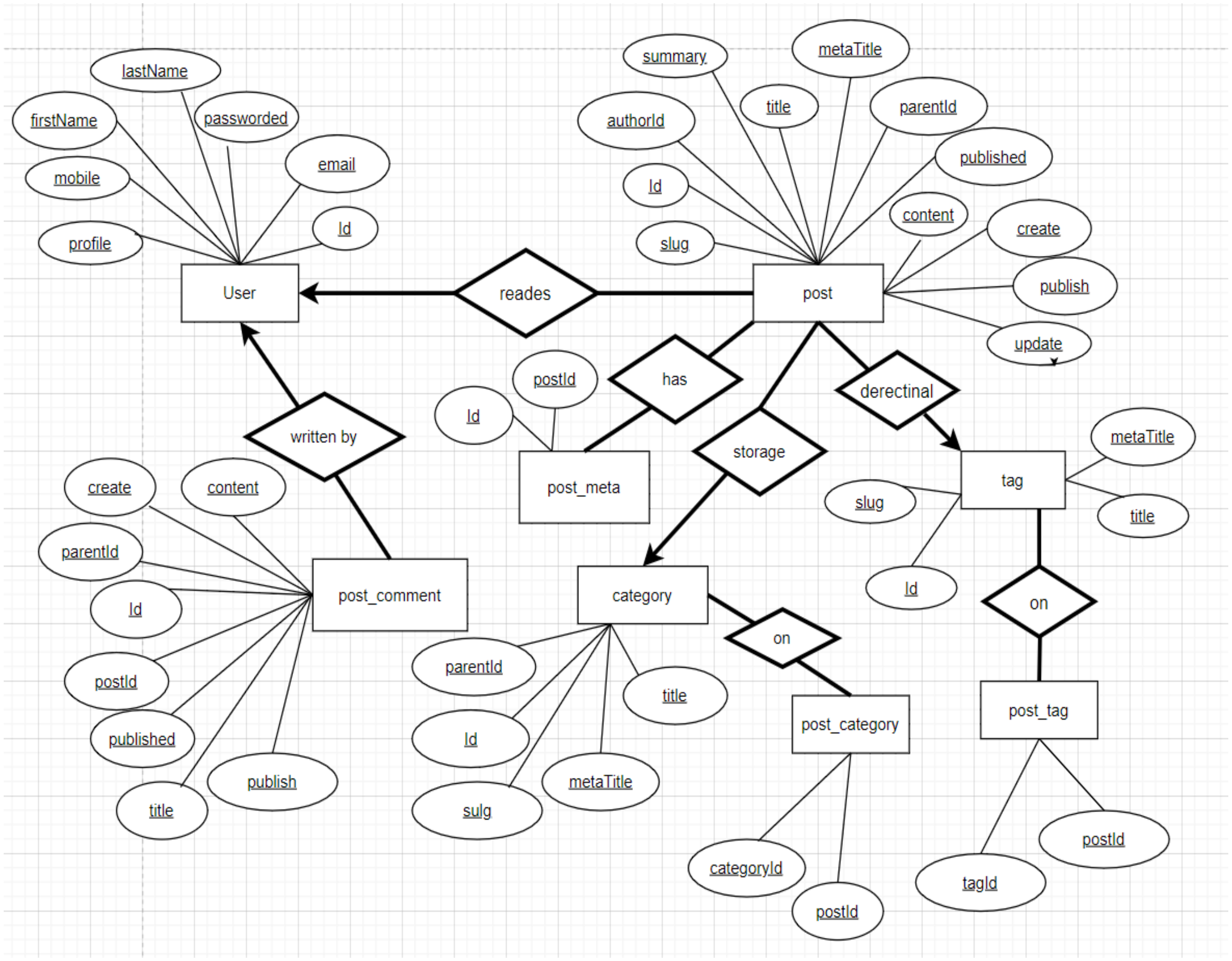
2.1 The administrators

- Upload new information to the blog and its information including name, content (what to write about), status (ongoing or completed).
- Upload a new content, information of the blog.
- Track views and reader comments about the blog.
- Delete a blog or a blog's information.

2.2 Users account

- Basic information about the reader such as email account.
- Readers can rate the blog they have read on a scale of 1 star to 5 stars.
- Readers can create or delete personal comments.
- Readers can change or delete any information about personal accounts.

3 Entity relationship diagram

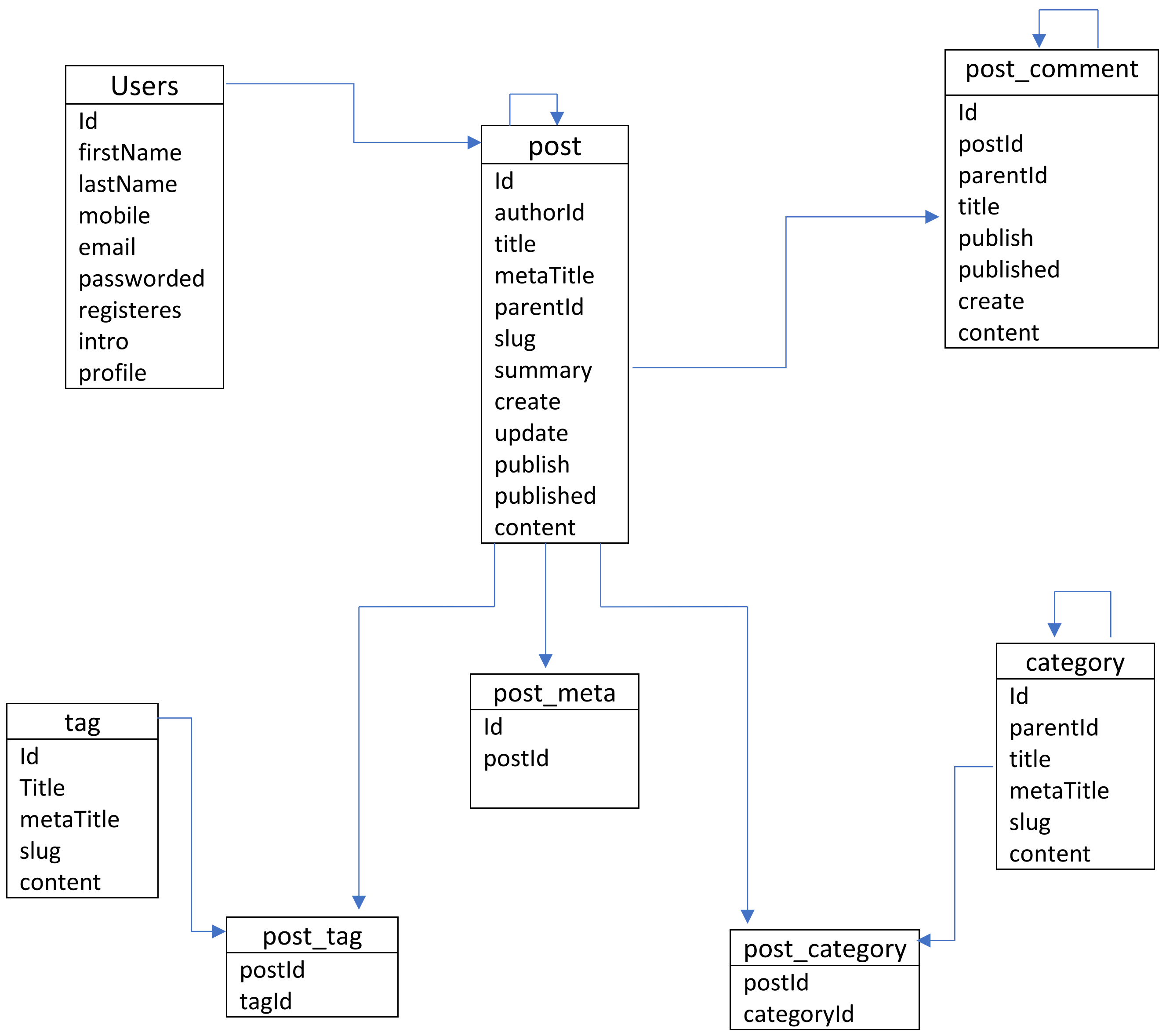


4 Schema

Based on the ERD, our group has developed a schema including 8 tables, all of which are in 3NF (third normal form). The function of each table is:

- **Users Table:** To store user information of all the post authors.
- **Post Table:** To store the post data.
- **Post Meta Table:** Can be used to store additional information of a post including the post banner URL etc.
- **Post Comment Table:** To store the post comments.
- **Post Category Table and Category Table:** To store the post categories and their mappings.
- **Post Tag Table and Tag Table:** Similar to the category and post category tables.

This schema has built so that it can be scaled and maintained easily in the future, when user requirements change or other functionalities are added to the website.


Users
Id
firstName
lastName
mobile
email
passworded
registeres
intro
profile
post
Id
authorId
title
metaTitle
parentId
slug
summary
create
update
publish
published
content
post_comment
Id
postId
parentId
title
publish
published
create
content
tag
Id
Title
metaTitle
slug
content
post_meta
Id
postId
category
Id
parentId
title
metaTitle
slug
content
post_tag
postId
tagId
post_category
postId
categoryId

5 Implementation of functions

5.1 Create table queries

```
CREATE TABLE Users (

    Id   INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    firstName   VARCHAR (100)   NULL,
    lastName   VARCHAR (100) NULL,
    mobile   BIGINT (100) NULL,
    email   VARCHAR (50) NULL,
    passworded   VARCHAR (16)   NULL,
    registeres   DATETIME NOT NULL,
    intro   TINYTEXT NULL,
    proflie   TEXT NULL
    );

    ALTER TABLE   Users
    ADD CONSTRAINT unique_mobile UNIQUE (mobile);

    ALTER TABLE   Users
    ADD CONSTRAINT unique_email UNIQUE (email);
```

Id	The unique id to identify the user.
First Name	The first name of the user.
Last Name	The last name of the user.
Mobile	The mobile number of the user. It can be used for login and registration purposes.
Email	The email of the user. It can be used for login and registration purposes.
Passworded	The passworded generate by the appropriate algorithm.
Registered	This column can be used to calculate the life of the user with the blog.
Intro	The brief introduction of the Author to be displayed on each post.
Profile	The author details to be displayed on the Author Page.

```
CREATE TABLE post (
        Id   INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
        authorId   INT NOT NULL REFERENCES Users (Id),
        title   VARCHAR (100)   NOT NULL,
        metaTitle   VARCHAR (100) NOT NULL,
        parentId   INT NOT NULL,
        sulg   VARCHAR (128) NOT NULL,
        summary TINYTEXT NULL,
        create   DATETIME NOT NULL,
        update   DATETIME NOT NULL,
        publish   TINYINT (10) NOT NULL DEFAULT 0,
        published    DATETIME NOT NULL,
        content   TEXT   NOT NULL
        );

 ALTER TABLE post
 ADD CONSTRAINT   fk_post_parentId
         FOREIGN KEY (parentId)
         REFERENCES post (Id)
         ON DELETE NO ACTION
         ON UPDATE NO ACTION;
```

Id	The unique id to identify the post.
Author Id	The author id to identify the post author.
Parent Id	The parent id to identify the parent post. It can be used to form the table of content of the parent post of series.
Title	The post title to be displayed on the Post Page and the lists.
Meta Title	The meta title to be used for browser title and SEO.
Slug	The post slug to form the URL.
Summary	The summary of the post to mention the key highlights.
Publish	It can be used to identify whether the post is publicly available.
Create	It stores the date and time at which the post is created.
Update	It stores the date and time at which the post is updated.
Published	It stores the date and time at which the post is published.
Content	The column used to store the post data.

```
CREATE TABLE post_meta (
        Id   INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
        postId   INT NOT NULL REFERENCES post (Id),
        keyPost   VARCHAR (100) NOT NULL,
        content   TEXT   NOT NULL
);
```

Id	The unique id to identify the post meta.
Post Id	The post id to identify the parent post.
KeyPost	The key identifying the meta.
Content	The column used to store the post data.

```
CREATE TABLE   post_comment (
            Id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
            postId   INT NOT NULL REFERENCES post (Id),
            parentId INT NOT NULL,
            title   VARCHAR (100) NOT NULL,
            create   DATETIME NOT NULL,
            publish   TINYINT (10) NOT NULL DEFAULT 0,
            published   DATETIME NOT NULL,
            content   TEXT   NOT NULL
);

ALTER TABLE   post_comment
ADD CONSTRAINT   fk_comment_parentId
       FOREIGN KEY (parentId)
       REFERENCES post_comment (Id)
       ON DELETE NO ACTION
       ON UPDATE NO ACTION;
```

Id	The unique id to identify the post comment.
Post Id	The post id to identify the parent post.
Parent Id	The parent id to identify the parent comment.
Title	The comment title.
Publish	It can be used to identify whether the comment is publicly available.
Create	It stores the date and time at which the comment is submitted.
Published	It stores the date and time at which the comment is published.
Content	The column used to store the comment data.

```
CREATE TABLE category (
            Id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
            parentId INT NOT NULL,
            title   VARCHAR (100) NOT NULL,
            metaTitle   VARCHAR (100) NOT NULL,
            sulg   VARCHAR (128) NOT NULL,
            content   TEXT   NOT NULL
);

ALTER TABLE category
ADD CONSTRAINT fk_category_parentId
    FOREIGN KEY (parentId)
    REFERENCES category (Id)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION;
```

Id	The unique id to identify the category.
Parent Id	The parent id to identify the parent category.
Title	The category titles.
Meta Title	The meta title to be used for browser title and SEO.
Slug	The category slug to form the URL.
Content	The column used to store the category data.

```
CREATE TABLE post_category (
            categoryId   INT NOT NULL,
            postId   INT NOT NULL,
            PRIMARY KEY (postId),
            FOREIGN KEY (postId)
            REFERENCES post (Id),
            PRIMARY KEY (categoryId),
            FOREIGN KEY (categoryId)
            REFERENCES category (Id)
);
```

Post Id	The post id to identify the post.
Category Id	The category id to identify the category.

```
CREATE TABLE tag (
            Id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
            title   VARCHAR (100) NOT NULL,
            metaTitle   VARCHAR (100) NOT NULL,
            sulg   VARCHAR (128) NOT NULL,
            content    TEXT   NOT NULL
);
```

Id	The unique id to identify the tag.
Title	The tag titles.
Meta Title	The meta title to be used for browser title and SEO.
Slug	The tag slug to form the URL.
Content	The column used to store the tag data.

```
CREATE TABLE post_tag (
        tagId   INT NOT NULL,
        postId   INT NOT NULL,
        PRIMARY KEY (postId),
        FOREIGN KEY (postId)
        REFERENCES post (Id),
        PRIMARY KEY (tagId),
        FOREIGN KEY (tagId)
        REFERENCES tag (Id)
);
```

PostId	The post id to identify the post.
TagId	The category id to identify the category.

5.2 Insert and update commands

For this type of command, multiple inputs are usually required. Moreover, a simple query sometimes cannot perform the whole operation since one insertion or update can affect a value in another table, where constraints such as UPDATE ON CASCADE or UPDATE CON DELETE are not suitable.

5.2.1 For the administrator

• Upload a new manga with an automatically assigned id. Views, ratings, total number of chapters are 0 by default. Latest update is the current timestamp by default.

```
DELIMITER $$
   CREATE PROCEDURE upload_blog
      (IN new_Id INT, IN new_metaTitle VARCHAR (100), IN new_authorId, IN
new_title VARCHAR (100), IN new_summary TINYTEXT, IN new_create
DATETIME, IN new_content)
  BEGIN
    INSERT INTO post (Id, metaTitle, authorId, title, summary, create, content)
    VALUES ('new_id', 'new_meta_title', 'new_athorID', 'new_title', 'new_summary',
'new_create', 'new_content');
  END $$
DELIMITER;
```

- **Upload a new category helps us has an overarching group of blog posts.**

```
    DELIMITER $$
      CREATE PROCEDURE newcategory
       (IN new_Id   INT, IN new_metaTitle VARCHAR (100), IN new_title
  VARCHAR (100), IN new_slug VARCHAR (128), IN new_content TEXT)
     BEGIN
        INSERT INTO post (Id, metaTitle, title, slug, content)
        VALUES ('new_id', 'new_meta_title', 'new_title', 'new_slug', 'new_content');
   END $$
 DELIMITER;
```

• **Upload a new the tag will help readers easily find the content on the blog.**

```
 DELIMITER $$
   CREATE PROCEDURE newtag (IN new_Id   INT, IN new_slug VARCHAR
(128))
 BEGIN
    INSERT INTO tag (Id, slug)
    VALUES ('new_ID','new_slug');
 END $$
DELIMITER;
```

5.2.2 For the reader

• **Create new account, reader's id is automatically assigned:**

```
 DELIMITER $$
   CREATE PROCEDURE new_reader_account


    INSERT INTO user (Id, firstName, lastName, mobile, email)
    VALUES ('new_Id','new_firstname','new_lastname','new-mobile','new_email');


 DELIMITER;
```

• Create new profile that matches the account.

```
DELIMITER $$
  CREATE PROCEDURE new_reader_personal

    INSERT INTO user (firstname, lastname, mobile, email)

   VALUES('new_id','new_firstName','new_lastName','new_mobile','new_email','n
  ew_profile')

DELIMITER;
```

• Make a new comment under a blog, the id and timestamp of the comment are assigned automatically:

```
DELIMITER $$
    CREATE PROCEDURE add_comment

        INSERT INTO post_comment (Id, title, create, publish, content)
       VALUES ('new_Id','new_title','new_create','new_publish','new_content');

DELIMITER;
```

• Update the comment of someone:

```
DELIMITER $$
      CREATE PROCEDURE update_comment
               (IN post_comment. Id   INT)
      BEGIN
          UPDATE comment SET upvote = upvote + 1
           WHERE Id = post_comment. Id;
    END $$
DELIMITER;
```

5.3 Delete commands

- **Delete a post with all of its comments:**

```
 DELIMITER $$
   CREATE PROCEDURE del_post (IN Id INT)
   BEGIN
        DELETE FROM post WHERE post.Id=Id;
        DELETE FROM post_meta WHERE post_meta. postId=Id;
        DELETE FROM post_comment WHERE post_comment.postId=Id;
        DELETE FROM post_category WHERE post_category.postId = Id;
        DELETE FROM post_tag WHERE post_tag.postId = Id;
 END $$
DELIMITER;
```

- **Delete a comment:**

```
  DELIMITER $$
    CREATE PROCEDURE del_comment (IN comment_id INT)
   BEGIN
      DELETE FROM post_comment
      WHERE Id= post_comment.Id;
END $$
DELIMITER;
```

- **Delete a user account along with their comments:**

```
  DELIMITER $$
     CREATE PROCEDURE del_user (IN id INT)
    BEGIN
       DELETE FROM Users WHERE Users.Id=Id;
       DELETE FROM post_comment WHERE post_comment.Id = Id;
   END $$
  DELIMITER;
```