**DataBases Report Project**

**Blog Management**

**  
 Group: 8**

**University of Science and Technology of Hanoi**

December,2020

**TABLE OF CONTENT**

[**1 Introduction** 3](#_Toc58508974)

[**1.1 Overview** 3](#_Toc58508975)

[**1.2 Group members** 3](#_Toc58508976)

[**2 User requirements** 4](#_Toc58508977)

[**2.1 The administrators** 4](#_Toc58508978)

[**2.2 Users account** 4](#_Toc58508979)

[**3 Entity relationship diagram** 5](#_Toc58508980)

[**4 Schema** 6](#_Toc58508981)

[**5 Implementation of functions** 8](#_Toc58508982)

[**5.1 Create table queries** 8](#_Toc58508983)

[**5.2 Insert and update commands** 14](#_Toc58508984)

[**5.2.1 For the administrator** 14](#_Toc58508985)

[**5.2.2 For the reader** 16](#_Toc58508986)

[**5.3 Delete commands** 18](#_Toc58508987)

## **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**

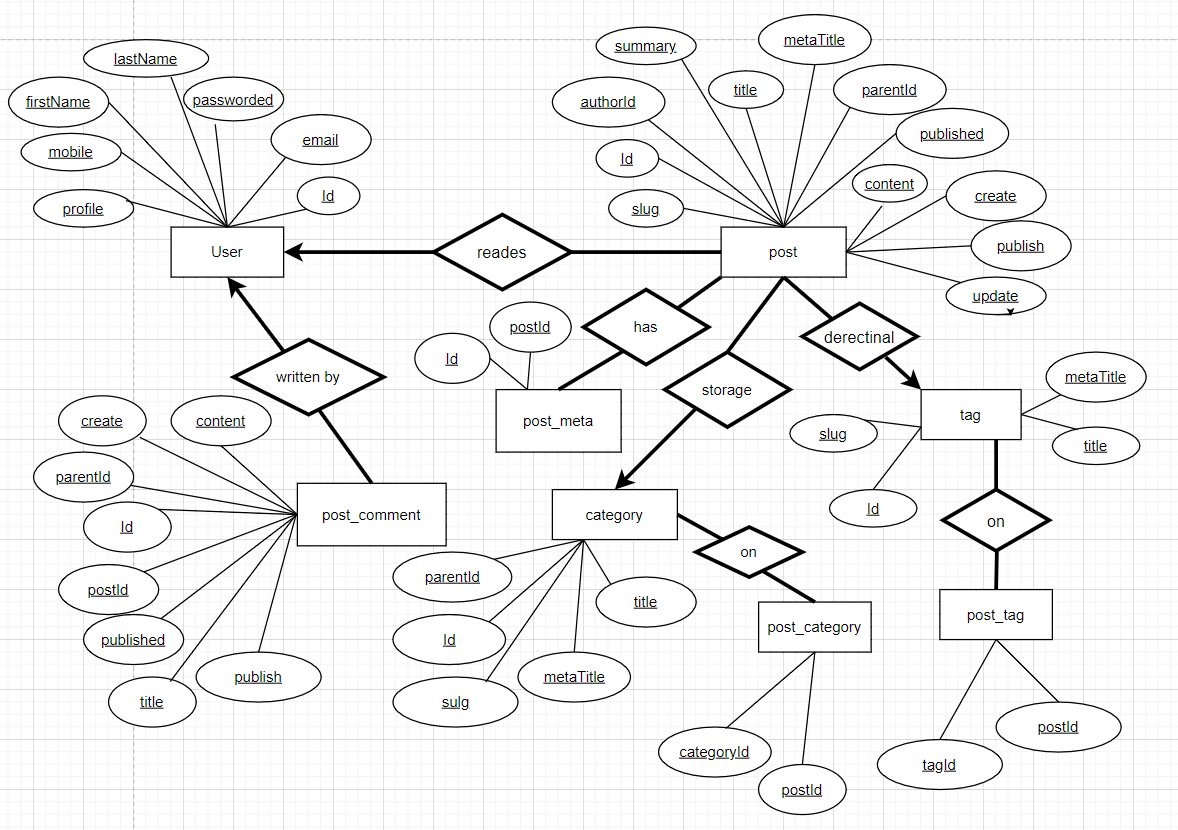
* Upload new information to the blog and its information including name, content (what to write about), status (ongoing or completed).
* Upload a new content, information of the blog.
* Track views and reader comments about the blog.
* Delete a blog or a blog's information.

### 

### **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.

|  |
| --- |
| post\_comment |
| Id  postId  parentId  title  publish  published  create  content |

|  |
| --- |
| Users |
| Id  firstName  lastName  mobile  email  passworded  registeres  intro  profile |

|  |
| --- |
| post |
| Id  authorId  title  metaTitle  parentId  slug  summary  create  update  publish  published  content |

|  |
| --- |
| category |
| Id  parentId  title  metaTitle  slug  content |

|  |
| --- |
| tag |
| Id  Title  metaTitle  slug  content |

|  |
| --- |
| post\_meta |
| Id  postId |

|  |
| --- |
| 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

|  |  |
| --- | --- |
| **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. |

ON UPDATE NO ACTION;

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','new\_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;