DATABASE FINAL REPORTS

**A database for an Internet forum**

Class: 2425I\_INT2211\_37

Group 727

|  |  |  |
| --- | --- | --- |
| Members | Works | Contributing point |
| Lê Vũ Hiếu 23020365 | **Leader**  View.sql  Testbench.sql  Write reports | 9/10 |
| Ngạc Anh Kiệt 21020690 | Planing and propose the system. ER model, Relational model Main.sql  InsertData.sql  Write reports | 10/10 |
| Phạm Quốc Hùng 23020373 | Trigger.sql  Procedure.sql  Write reports | 9/10 |

Contents

[1. Overview: 4](#_Toc184805978)

[1.1. Terminology 4](#_Toc184805979)

[1.2. Proposed model structure: 4](#_Toc184805980)

[2. Database stucture: 5](#_Toc184805981)

[2.1. Entity Relational Schema: 5](#_Toc184805982)

[2.2. Relational model: 6](#_Toc184805983)

[2.3. The Enhanced Entity Relational Schema: 7](#_Toc184805984)

[3. Trigger: 10](#_Toc184805985)

[3.1. Ensuring Date and Time: 10](#_Toc184805986)

[3.2. Ensuring no duplicate username: 10](#_Toc184805987)

[3.3. Trigger for soft-deletion: 11](#_Toc184805988)

[3.4. Trigger for tracking Old username: 12](#_Toc184805989)

[3.5. Prevent Admin of a box from deleting account: 12](#_Toc184805990)

[3.6. Image source validity: 12](#_Toc184805991)

[4. View: 13](#_Toc184805992)

[5. Procedure: 14](#_Toc184805993)

[5.1. Users’s general info: 14](#_Toc184805994)

[5.2. Handling many-to-many relationships: 15](#_Toc184805995)

[6. Conclusion: 15](#_Toc184805996)

# Overview:

## Terminology

In this report, we design a database for an online “forum”. A forum is a website where people gather to discuss, share, and exchange ideas about specific topics or interests. Prominent examples of forums include Reddit, Stack Exchange…

A forum is structured as discussion boards for people to create new posts and response to existing posts. A post may have various media such as video, images, in the end they usually convey a specific topic. Besides, various features can be embedded to enhance user experience like emoji reactions, post’s password.

When developers build a forum, several key aspects must be addressed. An intuitive, visually appealing, and logically structured user interface is crucial, as it shapes the first impression of the platform for new visitors. A secure, efficient, and error-free backend is also essential. This backend is responsible for storing users’ data, such as posts, media, passwords, and personal information, while also retrieving and serving this data when requested.

## Proposed model structure:

In this reports, we will focus on design and build a database for a forum. Our project consists of six files: FinalPJMain.sql, FinalPJTrigger.sql, FinalPJView.sql, FinalPJProcedure.sql, FinalPJ\_InsertData.sql and FinalPJTestbench.sql. Where as:

|  |  |
| --- | --- |
| FinalPJMain.sql | The very basis of the model, containing the database initializing, the entities initializing with coresponding relationship deployed. |
| FinalPJTrigger.sql | This file contains trigger command to ensure that the database works properly. It apply rules on specific entities, and attributes to avoid any abnormality. |
| FinalPJView.sql | This file contain views – virtual tables providing appropriate datas. |
| FinalPJProcedure.sql | Procedures also helps enhancing users’s experiences. It provides some function to help updating the database and showing various group of data. |
| FinalPJ\_InsertData.sql | This file inserts data into the model. |
| FinalPJTestbench.sql | This files store some sample commands such as SELECT statements, Procedures calling. Some statements may violate the rule to proof the efficiency of the Triggers. |

# Database stucture:

## Entity Relational Schema:

A computer generated diagram of a computer program

Description automatically generated with medium confidence

*Figure 1: The entity relationship diagram*

Figure 1 provides an overview of the tables and their relationships, illustrating the core structure of the forum's database. The schema contains nine main entities, each representing a core component of a forum. The entities’ names and their attributes are all detailed in the given diagram.

The \_User stores users’ data, while violation, DeletedUser, \_Role and Oldname entities are considered additional entities to provide more information about users. Posts created by users are managed in the Post table, a post may have medias and comments, they also need their own entities, which are \_Comment and Image table. In the forum's structure, users organize content into groups, commonly referred to as "boxes". These boxes are the foundational home of communities within a forum, every posts must be posted in a specific box.

## Relational model:

A screenshot of a computer code

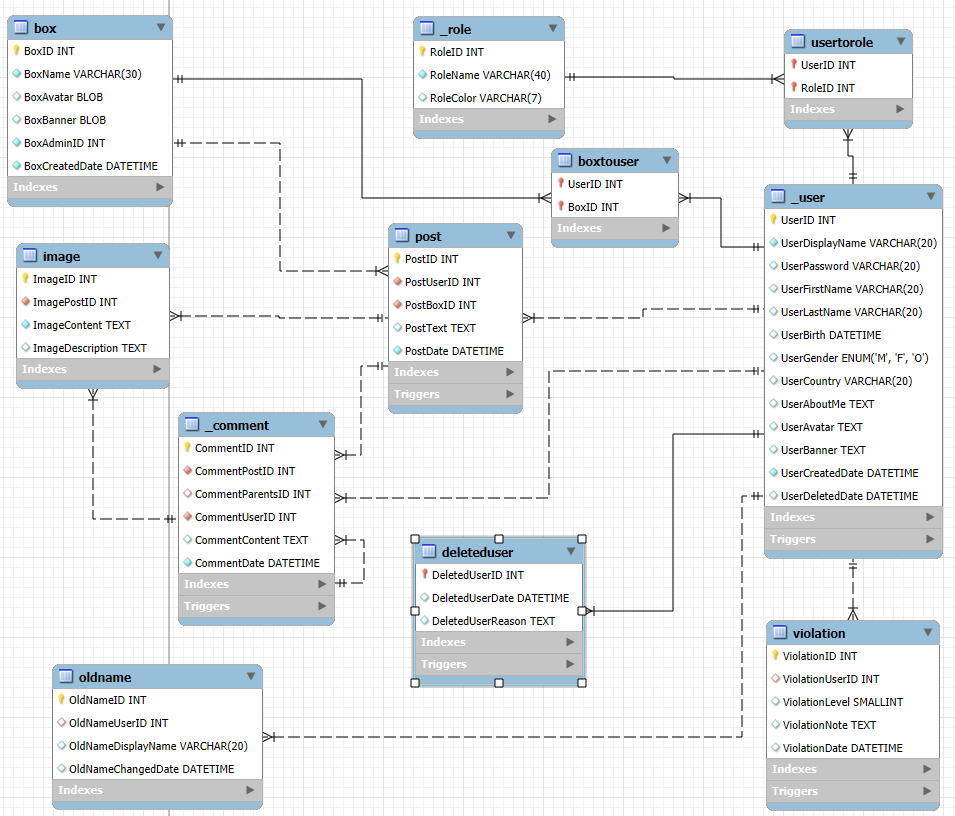
Description automatically generated

Figure 2: The relational model

Figure 2 shows the relational model of the proposed database, this relational model is a physical representation that can be directly implemented using a Relational Database Management System (RDBMS), in this reports we use MySQL.

## The Enhanced Entity Relational Schema:

The schema in Figure 1 only offer a basic understanding of the database, it is not yet able to depict the relationships between entities. To address this difficulty, we update it into an Enhanced Entity Relational (ERR) Schema, as illustrated in Figure 2.



This approach makes the Many-to-Many relationship more intuitive and clear. Table 1 lists and explains each entity, along with the relationships involved.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Purpose | Primary Key | Foreign Key and where it reference to | Relationship |
| \_User | This table stores users’s personal information | UserID | None | None |
| Violation | This table stores users’s law violation attempts. | ViolationID | ViolationUserID  -> \_User(UserID) | Many-> One |
| DeletedUser | This table will be automatically filled using a trigger which will be mentioned later in this reports, for the purpose of soft-deletion. Soft-deletion is to updated a person entity (which is UserDeletedDate) instead of actual deletion from the database. Doing so will help prevent uneccessary improper phenomenon from happening. | DeletedUserID | DeletedUserID ->\_User(UserID) | One -> One |
| OldName | This table tracks users’s old display name when they change ones. | OldNameID | OldNameID -> \_User(UserID) | Many -> One |
| UserToRole | A bridge entity describe the many-to-many role between \_User and \_Role | UserID, RoleID | UserID -> \_User(UserID) RoleID -> \_Role(RoleID) | Many -> One Many -> One |
| \_Role | A table for users’ roles. Roles decides users’s right in a specific Box, or even the entire forum. Sometime, it’s just for fun. | RoleID | None | None |
| Post | This table stores users’s posts. | PostID | PostUserID -> \_User(UserID) PostBoxID -> Box(BoxID) | Many -> One  Many -> One |
| Box | A box, as mentioned earlier, serves as a home for a group of users and must have at least one user (who acts as the Admin if they are the sole member). All posts are created within a specific box. | BoxID | None | None |
| BoxToUser | A bridge entity describe the many-to-many role between Box and \_User | BoxID, UserID | UserID -> \_User(UserID) BoxID -> Box(BoxID) | Many -> One  Many -> One |
| Image | This table stores general medias for posts and comments. | ImageID | ImagePostID -> Post(PostID),  ImageCommentID  -> \_Comment (CommentID) | Many -> One  Many -> One |
| \_Comment | This table stores all comments belongs to posts and itself (commonly known as replies). If CommentParentsID is NULL, so that comment is replying directly to the post. | CommentID | CommentUserID -> \_User(UserID),  CommentPostID -> Post(PostID),  CommentParentsID -> \_Comment (CommentID) | Many -> One,  Many -> One,  One -> Many |

The above Database satisfies 3 Normal Form sinces:

* Satisfying First Normal Form: Each attribute contains only a single value.
* Satisfying Second Normal Form: Satisfies the First Normal Form, and the relation must not contain any partial dependency, or in another word, all non-key attributes must depends on the whole candidate key, not only a proper subset of it.

(If A is a proper subset of B, A must contain every element exists in B, and B must contain at least one element that is not existing in A)

* Satisfying Third Normal Form: Satisfies the Second Normal Form, and their is no transitive dependency.

(A transitive dependency is: In a table, if A determines B, B determines C, so A transitively determines C.)

# Trigger:

Trigger are called when an event happens, an event can be updating, deleting, modifying... We implement a couple of triggers to ensure no abnormalities occured in the database when it is modifying in all means.

## Ensuring Date and Time:

We created Triggers to ensure that the time an action taken (commenting, posting, entering a box...) is valid, since some events happens must to be earlier or later than a specific date and time to makes sense. For example, we have CheckDatePostToUser

DELIMITER ?

CREATE TRIGGER CheckDatePostToUser

BEFORE INSERT ON Post

FOR EACH ROW

BEGIN

DECLARE TriggerUserCreatedDate DATETIME;

SELECT UserCreatedDate INTO TriggerUserCreatedDate FROM \_User  
WHERE UserID = NEW.PostUserID;

IF NEW.PostDate <= TriggerUserCreatedDate THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'PostDate can not be earlier than UserCreatedDate.';

END IF;

END ?

CheckDatePostToUser prevent the created date of a post being earlier than users’ account created date, since the other way around does not make sense.

We also have CheckDateCommentToUser, CheckDateCommentToPost, CheckDateViolationToUser trigger to validate the same for other events.

## Ensuring no duplicate username:

Our system does not allow duplicate username, we implement a trigger to solve this:

DELIMITER ?

CREATE TRIGGER CheckDuplicateUserDisplayName

BEFORE INSERT ON \_User

FOR EACH ROW

BEGIN

DECLARE UserExists INT;

SELECT COUNT(\*) INTO UserExists FROM \_User WHERE UserDisplayName = NEW.UserDisplayName;

IF UserExists > 0 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'UserDisplayName already exists, Please choose a different username.';

END IF;

END ?

When table \_User is being inserted, we check the username before hand by perform the SELECT statement to count the username. If any found, the trigger prevent the data from inserting into the table.

## Trigger for soft-deletion:

Our system does not perform the actual deletion, we use the soft deletion technique – to update a value indicating that the user is deleted. In our database, we use DeletedUser table for that purpose. We provide AutoInsertUser function:

DELIMITER ?

CREATE TRIGGER AutoInsertUser

AFTER INSERT ON \_User

FOR EACH ROW

BEGIN

INSERT INTO DeletedUser(DeletedUserID, DeletedUserDate, DeletedUserReason) VALUES(NEW.UserID, NULL, "");

END ?

Everytime the table \_User is updated, DeletedUser table will also be updated right after initializing the attribute DeletedUserDate NULL by default.

Following by that, we also program a trigger preventing any deletion occurs on \_User table

DELIMITER ?

CREATE TRIGGER UserDeleteLog

BEFORE DELETE ON \_User

FOR EACH ROW

BEGIN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Please do the soft-deletion by updating the DeletedUserDate in DeletedUser table;';

END ?

## Trigger for tracking Old username:

For a forum, users usually change their username frequently. Users may exploit this privilege to cause trouble and unpleasant events, so we add a feature of tracking username change attempts to be able to track users identity if required. The trigger AutoInsertOldName will automatically update old username and its coresponding owner into the table OldUserName:

DELIMITER ?

CREATE TRIGGER AutoInsertOldName

AFTER UPDATE ON \_User

FOR EACH ROW

BEGIN

IF OLD.UserDisplayName <> NEW.UserDisplayName THEN

INSERT INTO OldName( OldNameUserID, OldNameDisplayName, OldNameChangedDate) VALUES(NEW.UserID, OLD.UserDisplayName, NOW());

END IF;

END ?

## Prevent Admin of a box from deleting account:

It is a norm to prevent an Admin of a box to delete their account. The admin must pass the Admin privilege to another member before leaving. We provide a Trigger to do this:

DELIMITER ?

CREATE TRIGGER PreventAdminUserDeleting

BEFORE UPDATE ON DeletedUser

FOR EACH ROW

BEGIN

DECLARE CheckAdmin INT;

SELECT COUNT(\*) INTO CheckAdmin FROM Box WHERE BoxAdminID = OLD.DeletedUserID;

IF CheckAdmin > 0 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'This user is currently an admin of at least one box and can not delete his account.';

END IF;

END ?

## Image source validity:

An image can belong exclusively to either a post or a comment. To enforce this, we implement the ImageSourceValidating trigger that ensures exactly one of the foreign keys - ImagePostID or ImageCommentID - is NULL, while the other is not NULL.

DELIMITER ?  
CREATE TRIGGER ImageSourceValidating  
BEFORE INSERT ON Image

FOR EACH ROW

BEGIN

IF (NEW.ImagePostID IS NULL AND NEW.ImageCommentID IS NULL) OR

(NEW.ImagePostID IS NOT NULL AND NEW.ImageCommentID IS NOT NULL) THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'An image can only belong to a post or a comment.';

END IF;

END ?

Those are numerous varifying process that will be likely happens in any forum database.

# View:

View is virtual table that display the proper datas instead of the whole table. By doing this, we can limit what users may see, prevent sensitive or unecessary data being leaked. We implement various views which may be useful when retrieving data in the future:

#============

# View all active user (not deleted)

#============

CREATE VIEW UserList AS

SELECT \_User.\*

FROM \_User JOIN DeletedUser ON \_User.UserID = DeletedUser.DeletedUserID

WHERE DeletedUser.DeletedUserDate IS NULL;

#========================

# View all deleted user

#========================

CREATE VIEW DeletedUserList AS

SELECT \_User.\*, DeletedUser.DeletedUserDate, DeletedUser.DeletedUserReason

FROM \_User JOIN DeletedUser ON \_User.UserID = DeletedUser.DeletedUserID

WHERE DeletedUser.DeletedUserDate IS NOT NULL;

#==============================================

# View all User and their history name changes

#==============================================

CREATE VIEW UserNameHistory AS

SELECT \_User.\*, DeletedUser.DeletedUserDate, DeletedUser.DeletedUserReason

FROM \_User JOIN DeletedUser ON \_User.UserID = DeletedUser.DeletedUserID

WHERE DeletedUser.DeletedUserDate IS NOT NULL;

#==========================

# View all posts and its box

#==========================

CREATE VIEW PostAndBox AS

SELECT Post.\*

FROM Post JOIN \_User ON Post.PostUserId = \_User.UserID

JOIN DeletedUser ON \_User.UserID = DeletedUser.DeletedUserID

WHERE DeletedUser.DeletedUserDate IS NULL;

#==============================

# View all user and their posts

#==============================

CREATE VIEW UsersAndPosts AS

SELECT \_User.UserID, \_User.UserDisplayname, Box.BoxID, Box.BoxName, Post.PostID, Post.PostText

FROM Box JOIN Post ON Box.BoxID = Post.PostBoxID

JOIN \_User ON Post.PostUserId = \_User.UserID

JOIN DeletedUser ON \_User.UserID = DeletedUser.DeletedUserID

WHERE DeletedUser.DeletedUserDate IS NULL;

# Procedure:

## Users’s general info:

We implement a procedure with an input parameter UserID. The procedure returns an output table of the user’s information with that ID

DELIMITER ??

CREATE PROCEDURE UserInfo (IN InUserID INT)

BEGIN

SELECT

InUserID AS 'TheUserID',

(SELECT UserDisplayName FROM \_User WHERE UserID = InUserID) AS DisplayName,

(SELECT COUNT(BoxID) FROM BoxToUser

WHERE UserID = InUserID) AS NumberOfBoxesJoined,

(SELECT COUNT(UserID) FROM UserToRole

WHERE UserID = InUserID) AS NumberOfRoles,

(SELECT COUNT(PostID) FROM Post

WHERE PostUserID = InUserID) AS NumberOfPosts,

(SELECT COUNT(CommentID) FROM \_Comment

WHERE CommentUserID = InUserID) AS NumberOfComments,

(SELECT COUNT(OldNameID) FROM OldName

WHERE OldNameUserID = InUserID) AS NumberOfOldNames,

(SELECT COUNT(ViolationID) FROM Violation

WHERE ViolationUserID = InUserID) AS NumberOfViolations;

END??

We call the procedure with:

CALL UserInfo(1);

And result in:



Several other parameters can be added into the procedure, the above ones is an example.

## Handling many-to-many relationships:

We provide procedures called UserJoinBox (IN InUserID INT, IN InBoxID INT) to handle the event of an user joining a box, and UserAddRole (IN InUserID INT, IN InRoleID INT) to handle the even of an user obtain a role. Basically when we want to update the event of an user joining a box, we need to upgrade the bridge table UserJoinBox, which is instead of updating \_User table or Box table, this is because of the many-to-many relationship between these two entities, same happens with \_User and \_Role.

DELIMITER ??

CREATE PROCEDURE UserJoinBox (IN InUserID INT, IN InBoxID INT)

BEGIN

INSERT INTO BoxToUser(UserID, BoxID) VALUES (InUserID, InBoxID);

END??

DELIMITER ??

CREATE PROCEDURE UserAddRole (IN InUserID INT, IN InRoleID INT)

BEGIN

INSERT INTO UserToRole(UserID, InRoleID) VALUES (InUserID, InRoleID);

END??

# Conclusion:

Our database model basically respresent a forum model, with the core entities working properly with each other. However, it still can be improve further, as we can implement more features into this database, such as chatting feature, sharing post feature and many other sides features that neccessary to enhance users’ experience.