veeForum

"Basi di dati 1" project – UNIME

Vittorio Romeo
http://vittorioromeo.info

Request

The client requests the implementation of **a forum system** and **a responsive web application**.

The forum system is intended as **a communication platform** for various projects, both for **internal employee communication** and **communication with the public**.

It is imperative that the system allows administrators to **easily create section hierarchies** and **user-group hierarchies**. Administrators need to be able to give groups **specific permissions for every section**.
Some sections will only be visible and editable to employee groups *(e.g. internal discussion)*, some sections will be visible but not editable by the public *(e.g. announcements)*, and others will need to be completely open to the public *(e.g. technical support)*.

Being able to **keep track of user-created content** is also very important for the client.
Initially, the date and the author of the content will be enough to track, but the system has to be designed in such a way that adding additional creation information *(e.g. browser used to post)* is easy.

The web application has to be **extremely simple** but **flexible** as well. Administrators need be able to perform all functions described above through a **responsive admin panel**.

Content consumers and creators should be able to **view and create content from the same responsive interface**.

Moderators and administrators should be able to **edit and delete posts through the same interface** as well.
User interface controls will be shown/hidden depending on the user's permissions.

Requirement and implementation analysis

To build the forum system, it is necessary to implement both **a database for data storage** and **a modern web application** to allow users and administrators to interact with the backend.

MariaDB, a modern drop-in replacement for **MySQL** will be used as the DBMS.

HTML5, **PHP5** and **JavaScript** conformant to the **5.1 ECMAScript specification** will be used for the development of the web application. The **AJAX** paradigm will be used to ensure that the application feels responsive and that user interaction is immediately reflected on the web application.

The client requested the possibility of defining **hierarchical user-group and content-section structures**. Therefore, **groups and sections need to refer to themselves recursively**. A good way of implementing this recursion is storing the id of the parent instance in the entity tables.

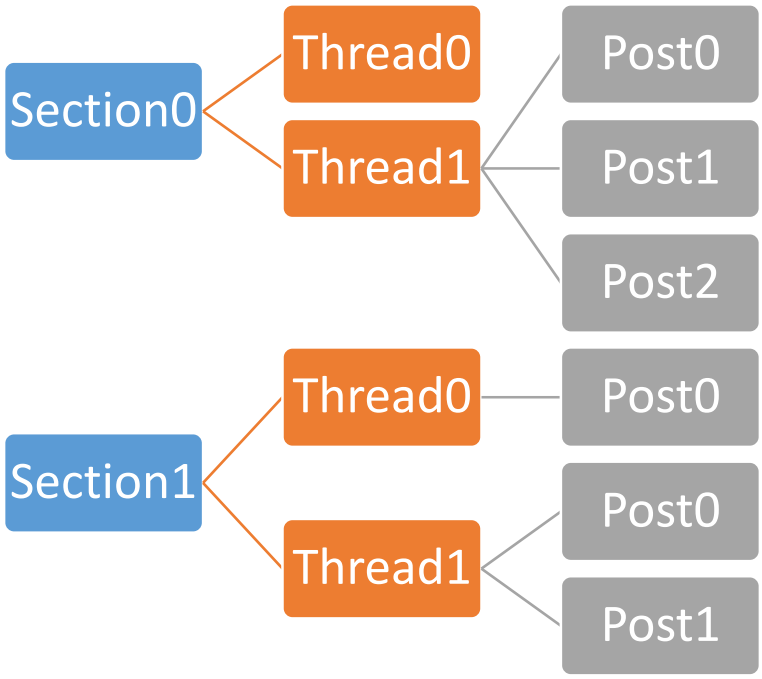
Another important feature the client desires is the ability to assign **specific permissions to groups for specific sections**. The implementation of this design will require an additional table, representing a **many-to-many relationship** between groups and sections. The table fields will then determine the permissions.

Threads and **posts** will be stored as separate tables. They, however, **share fields** regarding their creation. As these fields are duplicate between the two tables and considering the client **explicitly requested the ability to track content-creation data**, the shared fields will be implemented as a separate table that threads and posts can refer to. This design also becomes very beneficial if additional fields regarding the creation of content need to be added, as the client explained.

Application-wide permissions will be called **privileges** and will be stored in groups. A **bitset-like** data structure will be used to allow easy calculation of privileges inherited by parent groups. Privileges need to be separated from permissions as they are not related to specific sections.

Users will be stored in a specific table. A user has to belong to **one group**. Complex privilege and permission inheritance will be resolved by traversing the group hierarchy starting from the leaf, moving towards the root. As groups can have at most one parent, calculating the final permissions and privileges for a user is extremely easy – again, a bitset-like structure can be used.

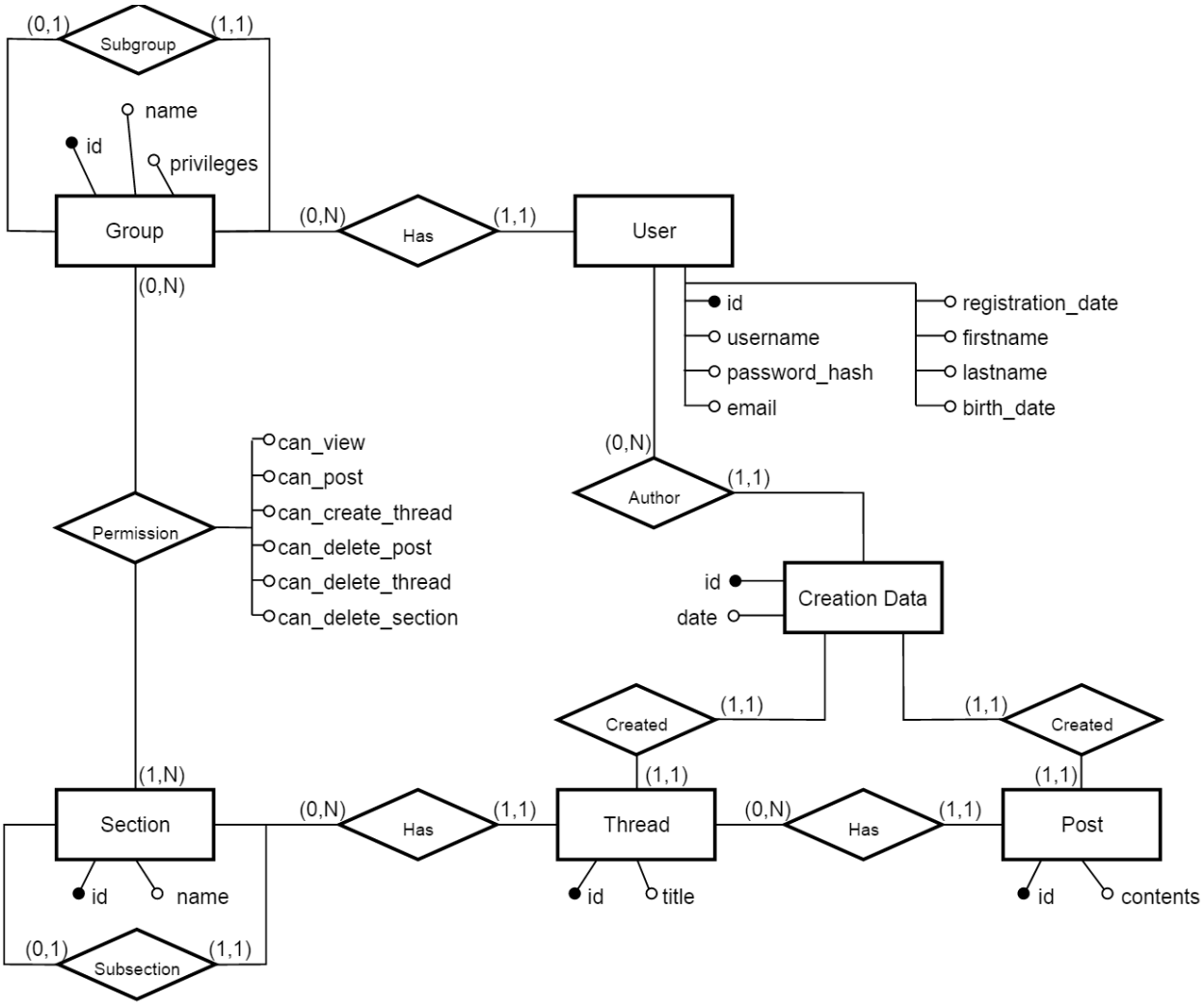
Sections will have a **one-to-many** relationship with **threads**. **Threads** will have a **one-to-many relationship** with **posts**. This is best illustrated with a simple diagram:



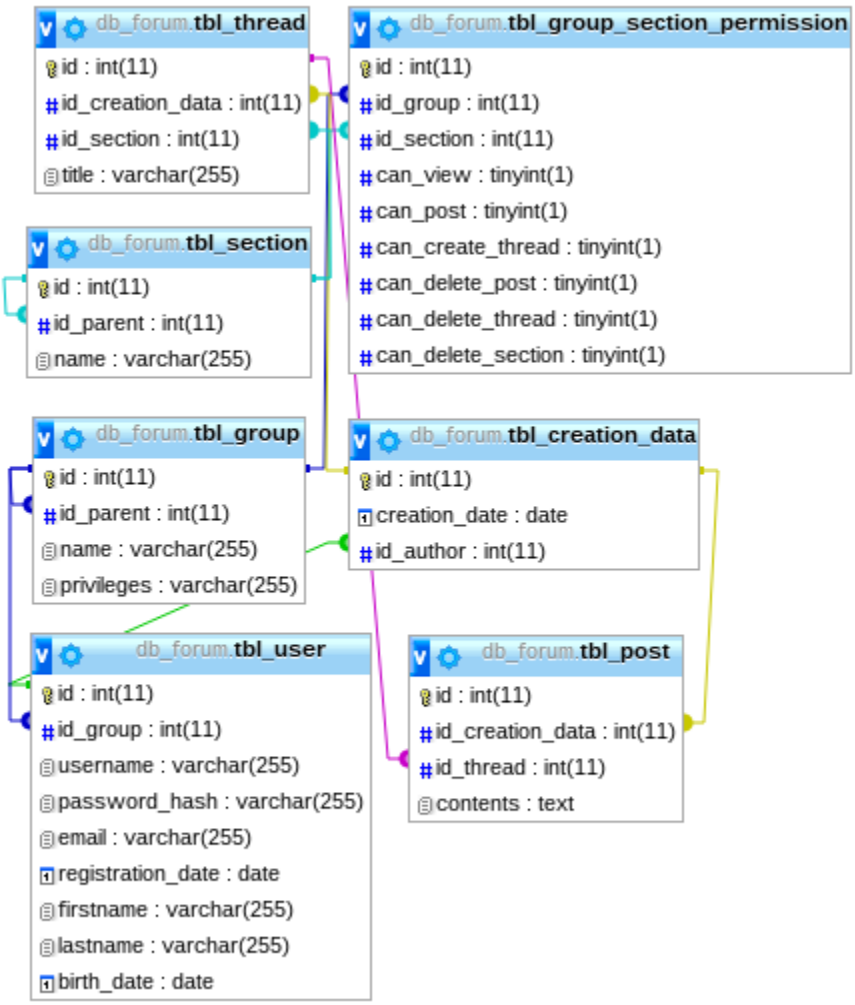
Posts will be simple entities with HTML content.

Database: Conceptual design

Here's the entity-relationship diagram of the database:



Database: Logical design



Web application: implementation details

Object-oriented design will be used as much as possible in the PHP5 backend code. The web application will be divided in two major modules: **library** and **web**.

The **library module** will contain functions and classes used throughout the whole application.

The **web module** will contain the actual web pages, divided in individual self-contained modules.

Web application: library module features

Session-stored variables will be managed through a static **Session** class, using statically-stored keys, creating a safe interface and making debugging easier.

Debugging will be handled through a static **Debug** class. Logging of errors and query information can be enabled and disabled from administrators, and will be automatically displayed using **AJAX**.

The **database connection** will be managed using the **mysqli PHP5** module. Every global database operation such as queries and connection will be wrapped in a safe interface that allows easy debugging and prevents security breaches.

Privileges and **permissions** will be loaded/saved from/to the database using **bitset-like** class instances, that support all basic bitset operations. Their underlying implementation is separated from their API – this allows developers to optimize or modify the bit storage without affecting code in the **web module**.

AJAX and **shortcut functions** for **HTML** generation will be handled through the **Gen** static class and the **Actions** static class. AJAX requests will directly call functions *(if valid)* from the **Actions** class, which return **HTML**, **JSON**, or plain text. **Gen** functions will be used from the **web module** to make the page structure more modular and avoid markup duplication.

Signing in and out and **current user data** will be managed from the **Credentials** static class. It will contain easy-to-use functions to check privileges and permissions, and also to handle login/logout.

Last, but not least, **database table interaction** will be handled by a very developer-friendly **object-oriented** interface. Every table in the database will have a corresponding **class**, derived from a generic **Table** class.

The **Table** class provides an object-oriented interface for common queries and **CRUD** operations. It also provides some very convenient methods to perform an action on every row matching a specific predicate or every row that's part of a hierarchy. Their usage, combined with **PHP5 lambda functions,** will make usually complex hierarchy-traversing operations easy to write and debug. These functions are available for every table in the database. The classes derived from **Table** will implement functionality that is unique for specific database entities. **Insertion** and **edit fields** will be specified in the constructor of these classes, allowing the developer to use a very convenient and clean syntax for the insertion/editing of table rows.