Universita' degli Studi di Messina
Dipartimento di Matematica e Informatica

Database course project

veeForum

29 May 2015

Author:
Vittorio Romeo

Professor:
Massimo Villari

http://vittorioromeo.info

http://unime.it

Contents

Part I

Project specifications

The following part of the document describes the project and its design/development process without exploring its implementation details.

The part begins with a synthesis of the **client request**. After a careful analysis of the request, a **Software Requirements Specification** (SRS) was written.

Writing a correct and informative SRS is of utmost importance to achieve an high-quality final product and ensuring the development process goes smoothly.

The SRS will cover the following points in depth:

- **Scope and purpose**.
- **Feature and functions**.
- **External interface requirements**.
- **Functional requirements**.
- **Example use cases**.
- **Non-functional requirements**.
- **Analysis models**.

Chapter 1

Client request

The client requests the design and implementation of a **forum creation/management framework** and a **modern responsive web forum browsing/management application**.

The client intends using the requested forum framework **to build communication platforms** for various projects, both for internal employee usage and interaction with the public.

It is imperative for the system to allow administrators to easily well-organized create **content-section hierarchies** and **user-group hierarchies**.

Administrators also 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, tracking the date and the author of the content will be enough, but the system has to be designed in such a way that inserting additional creation information (e.g. browser/operating system used to post) will be trivial.

In the future, additional content types (e.g. videos, attachments) may be added to the system and their creation will have to be tracked as well.

Users and moderators will also need to be able to track user content through a **real-time notification system** directly from the web application interface.

This data needs to be independent from the contents, in order to easily allow administrators and project managers to gather statistical data on forum usage.

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 users permissions.

Chapter 2

Software Requirements Specification

1. Introduction

This **Software Requirements Specification** (SRS) chapter contains all the information needed by software engineers and project managers to design and implement the requested forum creation/management framework.

The SRS was written following the **Institute of Electrical and Electronics Engineers** (IEEE) guidelines on SRS creation.

1..1 Purpose

The SRS chapter is contained in the **non-technical** part of the thesis.

Its purpose is providing a **comprehensive description** of the objective and environment for the software under development.

The SRS fully describes **what the software will do** and **how it will be expected to perform**.

1..2 Scope

1..2.1 Identity

The software that will be designed and produced will be called **veeForum**.

1..2.2 Feature extents

The complete product will:

- Provide a framework for the **creation and the management of a forum system**.

- Allow its users to **deploy and administrate** multi-purpose forums.
- Give access to a **modern responsive web application** to setup, browse and manage the forum.

veeForum, however, will not:

- Provide infrastructure or implementation for a complete blog/website. The scope of the software is forum building.
- Implement instant private messaging - user-to-user chat is beyond the scope of the project.

1..2.3 Benefits and objectives

Deploying veeForum will give its users a number of important benefits and will fulfill specific objectives.

- Companies and individuals making use of veeForum will have access to an **easy-to-install** and **easy-to-use** forum creation and management platform.
- Users and moderators of the deployed forums will be able to **easily create, track and manage** content and other forum users.
- Forum administrators will be given **total control** of the forum structure, users and permissions through an **easy-to-use** responsive administration panel.

2. General description

2..1 Product perspective and functions

The product shares many basic aspects and features with existing forum frameworks such as **phpBB** or **vBulletin**: flat/threaded discussion support, nested sections, user attachments, etc.

veeForum improves on existing forum frameworks in the following ways:

- Provides a responsive web interface without postbacks.
- Allows users and moderators to subscribe and unsubscribe not only to posts, but to users and sections as well.
- Has a powerful real-time Facebook-like notification system that notifies users when tracked content has been added or edited.

- Gives administrator the possibility to design and manage complex permission hierarchies for user groups and single users.

2..2 User characteristics

veeForum needs to target both users that **only consume the content offered by deployed forums**, users that **actively create and manage content in deployed forums**, and users that **build and deploy forum instances**.

User-friendliness is essential for every target, but all the required functionality is effectively exposed to different user groups.

It is therefore required to have clear interfaces thar do not negatively affect the user experience by being either too complex or too simple (all features need to be exposed).

3. Specific requirements

3..1 External interface requirements

External interface requirements identify and document the interfaces to other systems and external entities within the project scope.

3..1.1 User interfaces

The product will provide both a desktop and a mobile user web interface.

- **Web interface**: it is required to provide a modern responsive web interface, compatible and tested with the most popular browsers (Internet Explorer 10+, Google Chrome, Mozilla Firefox). The web interface will give forum access to users and moderators, and administrator access to forum management staff.
- **Mobile interface**: is is required to provide a modern mobile application for the major platforms (Android, iOS, Windows Phone). The mobile application will allow browsing and content management of forums created with the product.

3..1.2 Software interfaces

The **open-source policy** of veeForum will allow framework users to expand or improve existing functionality and to interact with other existing technologies.

Accessing and modifying forum data (assuming permission requirements are satisfied by the user) will be possible through **RESTful** requests, returning and accepting **JSON** (Javascript Object Notation).

3..2 Functional requirements

In software engineering, a **functional requirement** defines a function of a system and its components.

Functional requirements may be **calculations**, **technical details**, **data manipulation and processing** and other specific functionality that define what a system is supposed to accomplish.

Behavioral requirements describing all the cases where the system uses the functional requirements are captured in **use cases**.

3..2.1 User/group management

- **Users**: users will be managed by the system. Users can register (or be manually added by an administrator). Registration can be configured to require a confirmation email or not.

- **Groups**: every user will be part of at least one group at all times. Groups are part of an hierarchy: they can inherit from each other. Groups can have permissions specific to sections and system-wide permissions.

3..2.2 Content hierarchy

- **Posts**: posts will be the base of the content hierarchy. They will contain HTML-enabled text and any number of attachments. Posts can be edited and deleted by the original owner.

- **Threads**: threads are groups of posts. Users with the correct permissions can create a thread in a specific section and have other users add posts or subscribe to it. Threads can be edited and deleted by the original owner.

- **Sections**: sections are content containers intended to group threads related to the same subject. Forum administrators and moderators can create sections and give users permissions to view or edit them.

3..2.3 Content tracking system

- **Creation data**: user-created content (posts, threads,attachments, etc) will have some data specific to its creation can be extended by forum administrators. Basic predefined data will consist of creation date and time. It will be possible to run statistical queries on content creation data.

- **Subscriptions**: users and moderators will be able to subscribe to specific sections, threads or user to track their contents. They will receive real-time notifications upon addition/editing of tracked content.

3..3 Example use cases

In software and systems engineering, a **use case** is a list of steps, typically defining interactions between one or more actors and a system, to achieve a goal.

3..3.1 Mobile game forum

A company developed a popular mobile game, with a wide audience. The company uses the **veeForum framework** to give users a place to discuss game strategy, give feedback on the quality of their product and receive technical support.

3..3.1.1 Actors

- Game developers.
- Game players.
- Forum management team.
- Technical support team.
- Feedback (PR) team.

3..3.1.2 Pre-conditions

- Release of a popular product with a wide audience.
- Game users need to register on the forum.

3..3.1.3 Flow of events

- Installation and configuration of a veeForum-enabled forum system by the forum management team.
- Creation of the sections and permission hierarchies by the forum management team.
- Registration and content creation by the game developers and game players.

3..3.1.4 Post-conditions

- Game players will be able to share their strategies and thoughts on the product.
- The technical support team will find all technical issues grouped in a convenient way and will be able to track individual issues. Technical support members will be able to communicate with each other in a private section.
- The feedback team will be able to track user suggestions and forward potential product improvements to the developer team.

3..3.2 Local city GNU/Linux usergroup forum

Some GNU/Linux users from the same city decide to start a local usergroup to discuss the GNU/Linux ecosystem and make new friends. In spirit with the open-source nature of the system, collaboration is extremely important. They require to easily assign specific permissions to users and groups to allow the forum to grow and be well-organized.

3..3.2.1 Actors

- Usergroup creators.
- Usergroup members.
- External visitors.

3..3.2.2 Pre-conditions

- Interest in a local GNU/Linux usergroup.
- Availability of people willing to collaborate.

3..3.2.3 Flow of events

- Installation and configuration of a veeForum-enabled forum system by the usergroup creators.
- Creation of the initial sections and permission hierarchies by the usergroup creators.
- Registration of usergroup members and external visitors.
- The usergroup creators give other usergroup members permissions to create and manage sections and users, starting a chain of collaborative forum content development.
- Usergroup members and external visitors contribute and make use of the content.

3..3.2.4 Post-conditions

- Local city usergroup members will be able to get to know and speak to each other.
- Usergroups members willing to contribute will be able to easily manage sections and write posts/articles.
- External visitors will be able to make use of the public content.

3..4 Non-functional requirements

Functional requirements are supported by **non-functional requirements** (also known as quality requirements), which impose constraints on the design or implementation (such as performance requirements, security, or reliability).

3..4.1 Performance

The system will be designed from the ground-up with emphasis on performance. As the forum may have huge amounts of contents and concurrent usage after its deployment, optimizing is a must.

When possible, functions will be implemented **directly in the database**, for maximum performance.

Web backend functions will also be carefully **optimized both for memory and speed**.

3..4.2 Reliability

The system will have to be reliable and keep working in case of errors.

Database queries and functions will be executed in **safe wrappers** that catch and handle errors carefully.

3..4.3 Security

veeForum needs to guarantee privacy and security for users and administrator of the system.

Well-tested and well-received **security idioms** and **encryption algorithms** will have to be used throughout the implementation of the whole system.

3..4.4 Maintainability and portability

Being an open-source project, **maintainability**, **extensibility** and **portability** are key.

The code layer will be carefully designed and organized to allow easy maintenance, bug-fixing and feature addition.

To ensure maximum portability, the product will be designed to work on the most popular **GNU/Linux** distributions and will be thoroughly tested on different platforms.

3..5 Logical Database Requirements

aaa

4. Analysis models

aaa

4..1 Sequence Diagrams

aaa

4..2 Data Flow Diagrams

aaa

4..3 State-Transition Diagrams

aaa

Part II

Technical analysis

The following part of the thesis will cover all implementation choices and details for veeForum in depth.

Firstly, the **development environment and tools** and **chosen technologies** will be described and motivated.

Afterwards, the technical details, including code examples and APIs, will be described for the two modules of the application: the **database** and the **web application**.

Every **table** of the database will be analyzed in detail, directly showing commented **DDL** code. The database also contains important **stored procedures** and **triggers** that are core part of the system's logic and that need to be explained in depth - the related **DML** code will be shown and commented.

The web application itself is divided in multiple modules:

- A **database interface backend module**, that interfaces with the database and wraps its tables and stored procedures.
- A **HTML5 generation module**, that greatly simplifies the creation of dynamic forum web pages by wrapping HTML5 controls in **object-oriented wrappers** that can be easily bound to callbacks and database events.
- A **modern responsive AJAX frontend** that allows users and interact with the backend module from multiple device, limiting postbacks and page refreshes.

Chapter 3

Development process

1. Environment and tools

All modules of veeForum have been developed on **Arch Linux x64**, a lightweight GNU/Linux distribution.

Arch is installed as a minimal base system, configured by the user upon which their own ideal environment is assembled by installing only what is required or desired for their unique purposes. GUI configuration utilities are not officially provided, and most system configuration is performed from the shell and a text editor. Based on a rolling-release model, Arch strives to stay bleeding edge, and typically offers the latest stable versions of most software.

No particular integrated development environments (IDEs) were used during the development - a modern graphical text editor, **Sublime Text 3**, was used instead.

2. Docker

Docker is an open-source project that **automates the deployment of applications** inside software containers, by providing an additional layer of abstraction and automation of operating-system-level virtualization on Linux.

Docker uses resource isolation features of the Linux kernel such as **cgroups** and **kernel namespaces** to allow independent containers to run within a single Linux instance.

This technology has been used since the beginning of the development process to **separate veeForum data and packages** from the host system and to dramatically increase **portability** and **ease of testing**.

Docker is also used for the installation of the product on target systems - with a single command it is possible to **retrieve all required dependencies**, correctly **configure the system** and **automatically install veeForum**.

3. Version control system

Version control systems (VCSs) allow the **management of changes** to documents, computer programs, large web sites, and other collections of information.

Nowadays, a version control system is **essential** for the development of any project. Being able to track changes, develop features in separate **branches**, have multiple programmers work on the same code base without conflicts and much more is extremely important for projects of any scope and size.

The chosen VCS is **Git**, a distributed revision control system with an emphasis on **speed**, **data integrity**, and support for **distributed, non-linear workflows**.

Git is widely appreciated in the private and open-source programming communities - it was initially designed and developed by **Linus Torvalds** for Linux kernel development in 2005, and has since become the most widely adopted version control system for software development.

The veeForum project is **open-source** and **appreciates feedback and contributions**. It is hosted on **GitHub**, a web-based Git repository hosting service, which offers all of the distributed revision control and source code management (SCM) functionality of Git, while adding **additional features** that make collaboration and public contributions easy and accessible.

4. Apache

The Apache HTTP server is the world's most widely used web server software.

Apache has been under open-source development for about 20 years - it supports all modern server-side technologies and programming languages, and also is **extremely reliable** and **secure**.

5. Thesis

The current document was written using LaTeX, an high-quality typesetting system; it includes features designed for the production of **technical and scientific documentation**.

LaTeXwas chosen for the current document because of the visually pleasant typography, its extensibility features and its abilities to include and highlight source code.

5..1 LatexPP

A small **C++14** LaTeXpreprocessor named **LatexPP** was developed for the composition of this thesis.

LatexPP allows to use an intuitive syntax that avoids markup repetition for code highlighting and macros.

Preprocessing and compiling a LaTeXdocument using LatexPP is simple and can be automated using a simple **bash** script.

```
1  #!/bin/bash
2
3  latexpp ./thesis.lpp > ./thesis.tex
4  pdflatex -shell-escape ./thesis.tex && chromium ./thesis.pdf
```

LatexPP is available as an open-source project on GitHub:
`https://github.com/SuperV1234/Experiments/Random`

Chapter 4

Project structure

The project folder and file structure is organized as such:

- **./doc/**

  Folder containing the documentation of the project.

  - **./latex/**
    LatexPP and LaTeX source and output files.

- **./sql/**

  Folder containing the SQL DDL scripts.

  - **./scripts/**
    Contains all the parts that make up the complete SQL initialization script.
  - **./mkScript.sh**
    Builds the complete SQL initialization scripts from the files in ./scripts/.
  - **./script.sql**
    Complete SQL initialization scripts that sets up a database suitable veeForum.

- **./exe/**

  Folder containing executable scripts to setup the system.

  - **./docker/**
    Docker-related scripts.
    * **./start.sh**
      Starts a Docker instance containing veeForum.
    * **./cleanup.sh**
      Cleans any running veeForum Docker instance.

        * **./shell.sh**
          Starts a Docker instance containing veeForum, controlling an instance of bash inside it.
        * **./httpdLog.sh**
          Prints the Apache error log of the current running veeForum Docker instance.

- **./www/**

  Folder containing web application data.

  - **./css/**
    CSS3 stylesheets.
  - **./js/**
    ECMAScript 5 script files.
  - **./json/**
    Non-relational data storage files, in JSON format.
  - **./php/**
    PHP backend code.
    * **./lib/**
      Backend to database interface library and HTML5 generation library.
    * **./core/**
      PHP frontend files that generate the responsive HTML5 web application user interface.

1. PHP Module

aaa

2. SQL Module

aaa

3. Other data

aaa

Chapter 5

SQL

1. Database setup

veeForum is supposed to be installed on a clean instance of MySQL server. The following script correctly initializes the required database and cleans any previous version of veeForum.

1..1 db

1..1.1 Code

```
1  ##################################################################################################
2  # Copyright (c) 2013-2015 Vittorio Romeo
3  # License: Academic Free License ("AFL") v. 3.0
4  # AFL License page: http://opensource.org/licenses/AFL-3.0
5  ##################################################################################################
6  # http://vittorioromeo.info
7  # vittorio.romeo@outlook.com
8  ##################################################################################################
9
10 ##################################################################################################
11 # veeForum forum framework initialization and creation script
12 ##################################################################################################
13
14 ##################################################################################################
15 # This script is meant to be run once to create and initialize
16 # from scratch the whole MySQL veeForum backend.
17 # Therefore, we drop the database if exists and re-create it.
18 drop database if exists db_forum_new$
19 create database db_forum_new$
20 use db_forum_new$
21 ##################################################################################################
```

1..1.2 Explanation

This script is meant to be run once to create and initialize from scratch the whole MySQL veeForum backend. Therefore, we drop the database if exists and re-create it.

2. Tables

A big amount of tables is required to make veeForum satisfy all requirements. Every table in the project is documented in the following section - the full **DDL** commented code and an explanation is provided for every table.

2..1 log

2..1.1 Code

```
##################################################################################################
# TABLE
# * This table deals with log messages.
##################################################################################################
create table tbl_log
(
    # Primary key
    id int auto_increment primary key,

    # Log type
    type int not null default 0,

    # Entry timestamp
    creation_timestamp timestamp not null default 0,

    # Name
    value varchar(512) not null
)$
##################################################################################################
```

2..1.2 Explanation

The **log** table is a simple non-relational list of log messages that can be used for debugging and security purposes.

2..2 tag

2..2.1 Code

```
1  ##################################################################################################
2  # TABLE
3  # * This table deals with tag archetypes.
4  ##################################################################################################
5  create table tbl_tag
6  (
7      # Primary key
8      id int auto_increment primary key,
9
10     # Name
11     value varchar(32) not null unique
12 )$
13 ##################################################################################################
```

2..2.2 Explanation

The **tag** table is a simple non-relational list of unique tags that can be attached to user-created content.

2..3 group

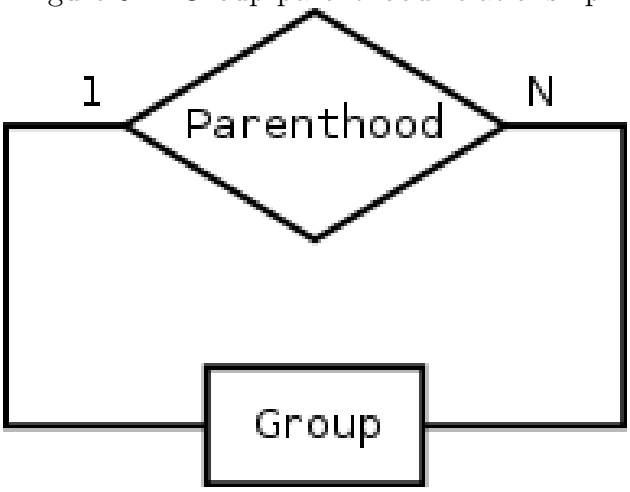
2..3.1 Code

```
1  ##################################################################################################
2  # TABLE
3  # * This table deals with groups.
4  # * Every group row also contains its forum-wide privileges.
5  ##################################################################################################
6  create table tbl_group
7  (
8      # Primary key
9      id int auto_increment primary key,
10
11     # Parent group (null is allowed)
12     id_parent int,
13
14     # Name,
15     name varchar(255) not null,
16
17     # Privs
18     is_superadmin boolean not null default false,
19     can_manage_sections boolean not null default false,
20     can_manage_users boolean not null default false,
21     can_manage_groups boolean not null default false,
22     can_manage_permissions boolean not null default false,
23
24     foreign key (id_parent)
25         references tbl_group(id)
26         on update cascade
27         on delete cascade
28 )$
29 ##################################################################################################
```

2..3.2 Explanation

The **group** table defines the groups users can belong to. Every row defines a different group and assigns forum-wide permissions to them. Groups can inherit from each other thanks to the `id_parent` field, which is the id of the parent group and can be `NULL`.

Figure 5.1: Group parenthood relationship.



2..4 user

2..4.1 Code

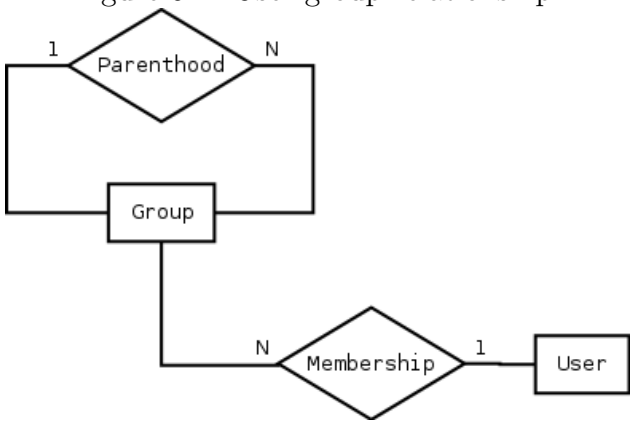
```
1  ##############################################################################################
2  # TABLE
3  # * This table deals with users.
4  ##############################################################################################
5  create table tbl_user
6  (
7      # Primary key
8      id int auto_increment primary key,
9
10     # Group of the user
11     id_group int not null,
12
13     # Credentials
14     username varchar(255) not null,
15     password_hash varchar(255) not null,
16     email varchar(255) not null,
17     registration_date date not null,
18
19     # Personal info
20     firstname varchar(255),
21     lastname varchar(255),
```

```
22      birth_date date,
23
24      foreign key (id_group)
25          references tbl_group(id)
26          on update cascade
27          on delete cascade
28  )$
29  ##############################################################################################
```

2..4.2 Explanation

The **group** table contains the users registered to the forum system. Every user **needs** to belong to a group, whose id is stored in `id_group`. Every row stores user credentials data and personal info.

Figure 5.2: User-group relationship.



2..5 section

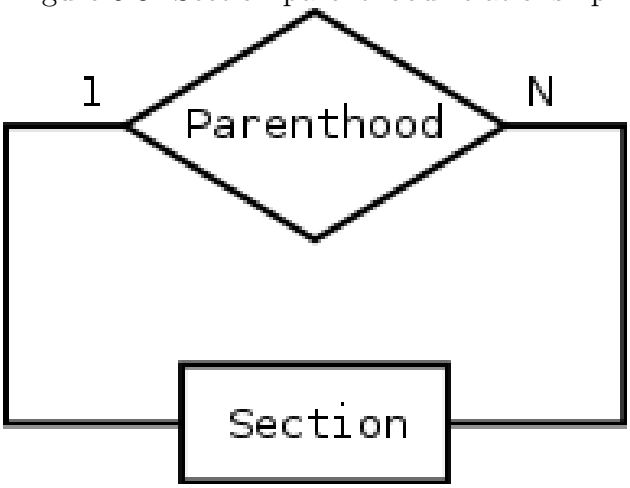
2..5.1 Code

```
1  ##############################################################################################
2  # TABLE
3  # * This table deals with sections.
4  ##############################################################################################
5  create table tbl_section
6  (
7      # Primary key
8      id int auto_increment primary key,
9
10     # Parent section (null is allowed)
11     id_parent int,
12
13     # Data
14     name varchar(255) not null,
15
16     foreign key (id_parent)
17         references tbl_section(id)
18         on update no action
19         on delete no action
20 )$
21 ##############################################################################################
```

2..5.2 Explanation

The **section** table contains all forum sections, defining the base hierarchy for content. Sections have a name and can inherit from each other thanks to the `id_parent` field, which is the id of the parent section and can be `NULL`.

Figure 5.3: Section parenthood relationship.



2..6 fileData

2..6.1 Code

```
##############################################################################################
# TABLE
# * This table deals with binary file data.
# * Used for attachments.
##############################################################################################
create table tbl_file_data
(
    # Primary key
    id int auto_increment primary key,

    # Data
    filename varchar(255) not null,
    data blob not null
)$
##############################################################################################
```

2..6.2 Explanation

The **fileData** table stores binary data and a filename for attachments. It makes use of the `blob` MySQL data type to directly store binary data in the database backend.

2..7 contentBase

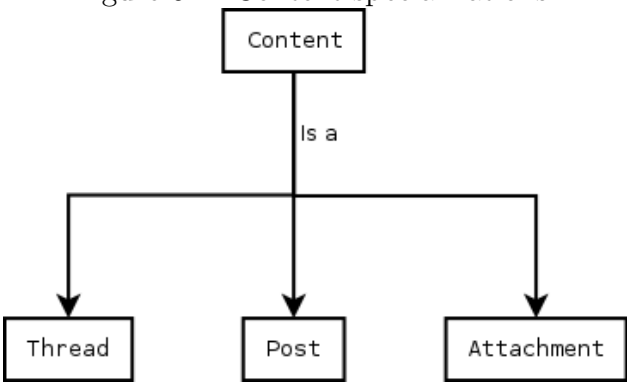
2..7.1 Code

```
1  ##################################################################################################
2  # TABLE
3  # * This table deals with content shared data.
4  ##################################################################################################
5  # HIERARCHY
6  # * Is base of: tbl_content_thread, tbl_content_post,
7  #               tbl_content_attachment
8  ##################################################################################################
9  create table tbl_content_base
10 (
11     # Primary key
12     id int auto_increment primary key,
13
14     # Data
15     creation_timestamp timestamp not null default 0,
16     id_author int not null,
17
18     foreign key (id_author)
19         references tbl_user(id)
20         on update no action
21         on delete no action
22 )$
23 ##################################################################################################
```

2..7.2 Explanation

The **contentBase** table defines the base entity of the content inheritance tree. Derived content types are: **threads**, **posts** and **attachments**. All content types share a `creation_timestamp` and an author, identified by `id_author`.

Figure 5.4: Content specializations.



2..8 contentThread

2..8.1 Code

```
1  ##################################################################################################
2  # TABLE
3  # * This table deals with threads, a type of content.
4  ##################################################################################################
5  # HIERARCHY
6  # * Derives from: tbl_content_base
7  ##################################################################################################
8  create table tbl_content_thread
9  (
10     # Primary key
11     id int auto_increment primary key,
12
13     # Content base
14     id_base int not null,
15
16     # Parent section
17     id_section int not null,
18
19     # Data
20     title varchar(255) not null,
21
22     foreign key (id_base)
23         references tbl_content_base(id)
24         on update cascade
```

```
25          on delete no action,
26
27      foreign key (id_section)
28          references tbl_section(id)
29          on update no action
30          on delete no action
31  )$
32  ##############################################################################################
```

2..8.2 Explanation

Content specialization for **threads**. A thread belongs to a section (identified by `id_section`) and has a `title`. The base content instance is identified by `id_base`.

2..9 contentPost

2..9.1 Code

```
##################################################################################################
# TABLE
# * This table deals with posts, a type of content.
##################################################################################################
# HIERARCHY
# * Derives from: tbl_content_base
##################################################################################################
create table tbl_content_post
(
    # Primary key
    id int auto_increment primary key,

    # Creation data
    id_base int not null,

    # Parent thread
    id_thread int not null,

    # Data
    contents text not null,

    foreign key (id_base)
        references tbl_content_base(id)
        on update cascade
        on delete no action,

    foreign key (id_thread)
        references tbl_content_thread(id)
        on update no action
        on delete no action
)$
##################################################################################################
```

2..9.2 Explanation

Content specialization for **posts**. A post belongs to a thread (identified by `id_thread`) and has text `contents`. The base content instance is identified by `id_base`.

2..10 contentAttachment

2..10.1 Code

```
##################################################################################################
# TABLE
# * This table deals with attachments, a type of content.
##################################################################################################
# HIERARCHY
# * Derives from: tbl_content_base
##################################################################################################
create table tbl_content_attachment
(
    # Primary key
    id int auto_increment primary key,

    # Creation data
    id_base int not null,

    # Parent post
    id_post int not null,

    # File data
    id_file_data int not null,

    foreign key (id_base)
        references tbl_content_base(id)
        on update cascade
        on delete cascade, # TODO: use a trigger

    foreign key (id_post)
        references tbl_content_post(id)
        on update no action
        on delete no action,

    foreign key (id_file_data)
        references tbl_file_data(id)
        on update no action
        on delete no action
)$
##################################################################################################
```

2..10.2 Explanation

Content specialization for **attachments**. An attachment belongs to a post (identified by `id_post`) and points to a specific file data instance `id_file_data`. The base content instance is identified by `id_base`.

2..11 subscriptionBase

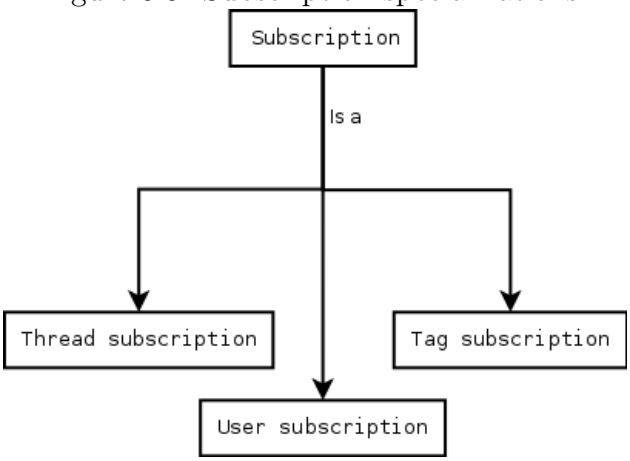
2..11.1 Code

```
1  ##############################################################################################
2  # TABLE
3  # * This table deals with subscription shared data.
4  # * Subscriptions allow users to track content or other users.
5  ##############################################################################################
6  # HIERARCHY
7  # * Is base of: tbl_subscription_thread, tbl_subscription_tag,
8  #               tbl_subscription_user
9  ##############################################################################################
10 create table tbl_subscription_base
11 (
12     # Primary key
13     id int auto_increment primary key,
14
15     # Subscriptor user
16     id_subscriptor int not null,
17
18     # Timestamp of beginning
19     creation_timestamp timestamp not null default 0,
20
21     # Active/inactive
22     active boolean not null default true,
23
24     foreign key (id_subscriptor)
25         references tbl_user(id)
26         on update cascade
27         on delete cascade
28 )$
29 ##############################################################################################
```

2..11.2 Explanation

The **subscriptionBase** table defines the base entity of the subscription inheritance tree. Derived subscription types are: **thread subscriptions**, **user subscriptions** and **tag subscriptions**. All subscription types share a `creation_timestamp` (beginning of the subscription), a subscriptor (identified by `id_subscriptor`) and an `active` flag that can be turned on and off from the web interface by the subscriptor.

Figure 5.5: Subscription specializations.



2..12 subscriptionThread

2..12.1 Code

```
1  ##############################################################################################
2  # TABLE
3  # * This table deals with thread subscriptions.
4  ##############################################################################################
5  # HIERARCHY
6  # * Derives from: tbl_subscription_base
7  ##############################################################################################
8  create table tbl_subscription_thread
9  (
10     # Primary key
11     id int auto_increment primary key,
12
13     # Base implementation id
14     id_base int not null,
15
16     # Target thread
17     id_thread int not null,
18
19     foreign key (id_base)
```

```
20          references tbl_subscription_base(id)
21          on update cascade
22          on delete cascade, # TODO: use a trigger
23
24      foreign key (id_thread)
25          references tbl_content_thread(id)
26          on update cascade
27          on delete no action # Triggers do not get fired with 'cascade'
28  )$
29  ##############################################################################################
```

2..12.2 Explanation

Subscription specialization for **thread subscriptions**. Allows to track a thread (identified by `id_thread`) for new content additions. The base subscription instance is identified by `id_base`.

2..13 subscriptionUser

2..13.1 Code

```
##############################################################################################
# TABLE
# * This table deals with user subscriptions.
##############################################################################################
# HIERARCHY
# * Derives from: tbl_subscription_base
##############################################################################################
create table tbl_subscription_user
(
    # Primary key
    id int auto_increment primary key,

    # Base implementation id
    id_base int not null,

    # Target user
    id_user int not null,

    foreign key (id_base)
        references tbl_subscription_base(id)
        on update cascade
        on delete cascade, # TODO: use a trigger

    foreign key (id_user)
        references tbl_user(id)
        on update cascade
        on delete no action # Triggers do not get fired with 'cascade'
)$
##############################################################################################
```

2..13.2 Explanation

Subscription specialization for **user subscriptions**. Allows to track an user (identified by `id_user`) for new content additions. The base subscription instance is identified by `id_base`.

2..14 subscriptionTag

2..14.1 Code

```
##############################################################################################
# TABLE
# * This table deals with tag subscriptions.
##############################################################################################
# HIERARCHY
# * Derives from: tbl_subscription_base
##############################################################################################
create table tbl_subscription_tag
(
    # Primary key
    id int auto_increment primary key,

    # Base implementation id
    id_base int not null,

    # Target tag
    id_tag int not null,

    foreign key (id_base)
        references tbl_subscription_base(id)
        on update cascade
        on delete cascade,

    foreign key (id_tag)
        references tbl_tag(id)
        on update cascade
        on delete no action # Triggers do not get fired with 'cascade'
)$
##############################################################################################
```

2..14.2 Explanation

Subscription specialization for **tag subscriptions**. Allows to track a tag (identified by `id_tag`) for new content additions. The base subscription instance is identified by `id_base`.

2..15 notificationBase

2..15.1 Code

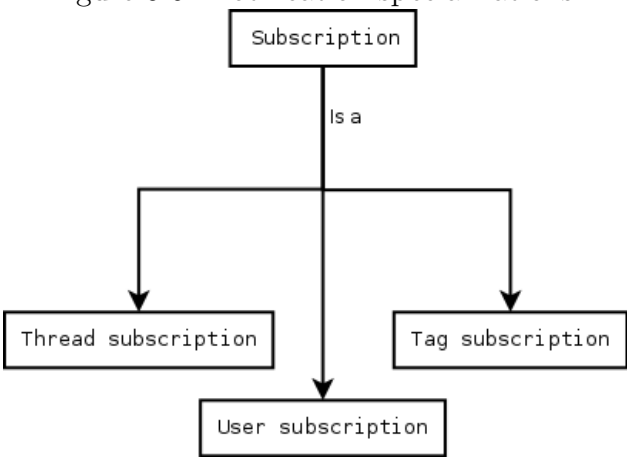
```
1  ##################################################################################################
2  # TABLE
3  # * This table deals with notification shared data.
4  # * Notifications are created when users need to be notified
5  #   about content they are subscribed to.
6  ##################################################################################################
7  # HIERARCHY
8  # * Is base of: tbl_notification_user, tbl_notification_thread,
9  #               tbl_notification_tag
10 ##################################################################################################
11 create table tbl_notification_base
12 (
13     # Primary key
14     id int auto_increment primary key,
15
16     # Receiver of the notification
17     id_receiver int not null,
18
19     # Notification seen?
20     seen boolean not null default false,
21
22     # Notification data creation timestamp
23     creation_timestamp timestamp not null default 0,
24
25     foreign key (id_receiver)
26         references tbl_user(id)
27         on update cascade
28         on delete cascade
29 )$
30 ##################################################################################################
```

2..15.2 Explanation

The **notificationBase** table defines the base entity of the notification inheritance tree. Derived notification types are: **thread notifications**, **user notifications** and **tag notifications**. All notifications types share a `seen` flag (which is set to `true` if the receiver seen a particular notification), a receiver (identified by `id_receiver`) and a `creation_timestamp`.

Notifications are created from subscriptions, using triggers.

Figure 5.6: Notification specializations.



2..16 notificationUser

2..16.1 Code

```
1  ##############################################################################################
2  # TABLE
3  # * This table deals with user notifications.
4  ##############################################################################################
5  # HIERARCHY
6  # * Derives from: tbl_notification_base
7  ##############################################################################################
8  create table tbl_notification_user
9  (
10     # Primary key
11     id int auto_increment primary key,
12
13     # Base
14     id_base int not null,
15
16     # Subscription
17     id_subscription_user int not null,
18
19     # Content posted by the user
```

```
20      id_content int not null,
21
22      foreign key (id_base)
23          references tbl_notification_base(id)
24          on update cascade
25          on delete cascade,
26
27      foreign key (id_subscription_user)
28          references tbl_subscription_user(id)
29          on update cascade
30          on delete no action, # Triggers do not get fired with 'cascade'
31
32      foreign key (id_content)
33          references tbl_content_base(id)
34          on update cascade
35          on delete no action # TODO Triggers do not get fired with 'cascade'
36  )$
37  ##############################################################################################
```

2..16.2 Explanation

Notification specialization for **user notifications**. Generated when a tracked user creates new content. Points to the subscription that generated the notification (identified by `id_subscription_user`) and to the created content (identified by `id_content`). The base notification instance is identified by `id_base`.

2..17 notificationThread

2..17.1 Code

```
##############################################################################################
# TABLE
# * This table deals with thread notifications.
##############################################################################################
# HIERARCHY
# * Derives from: tbl_notification_base
##############################################################################################
create table tbl_notification_thread
(
    # Primary key
    id int auto_increment primary key,

    # Base
    id_base int not null,

    # Subscription
    id_subscription_thread int not null,

    # Newly created post
    id_post int not null,

    foreign key (id_base)
        references tbl_notification_base(id)
        on update cascade
        on delete cascade,

    foreign key (id_subscription_thread)
        references tbl_subscription_thread(id)
        on update cascade
        on delete no action, # Triggers do not get fired with 'cascade'

    foreign key (id_post)
        references tbl_content_post(id)
        on update cascade
        on delete no action # Triggers do not get fired with 'cascade'
)$
##############################################################################################
```

2..17.2 Explanation

Notification specialization for **thread notifications**. Generated when new content is added to a tracked thread. Points to the subscription that generated the notification (identified by `id_subscription_thread`) and to the created content (identified by `id_content`). The base notification instance is identified by `id_base`.

2..18 notificationTag

2..18.1 Code

```
##################################################################################################
# TABLE
# * This table deals with tag notifications.
##################################################################################################
# HIERARCHY
# * Derives from: tbl_notification_base
##################################################################################################
create table tbl_notification_tag
(
    # Primary key
    id int auto_increment primary key,

    # Base
    id_base int not null,

    # Subscription
    id_subscription_tag int not null,

    foreign key (id_base)
        references tbl_notification_base(id)
        on update cascade
        on delete cascade,

    foreign key (id_subscription_tag)
        references tbl_subscription_tag(id)
        on update cascade
        on delete no action # Triggers do not get fired with 'cascade'
)$
##################################################################################################
```

2..18.2 Explanation

Notification specialization for **tag notifications**. Generated when new content is labeled with the tracked tag. Points to the subscription that generated the notification (identified by `id_subscription_tag`) and to the created content (identified by `id_content`). The base notification instance is identified by `id_base`.

2..19 tagContent

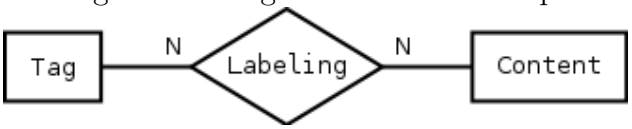
2..19.1 Code

```
##################################################################################################
# TABLE
# * This table deals with the many-to-many tag-content relationship.
##################################################################################################
create table tbl_tag_content
(
    # Primary key
    id int auto_increment primary key,

    # Tag
    id_tag int not null,

    # Content base
    id_content int not null,

    foreign key (id_tag)
        references tbl_tag(id)
        on update cascade
        on delete cascade,

    foreign key (id_content)
        references tbl_content_base(id)
        on update cascade
        on delete cascade
)$
##################################################################################################
```

2..19.2 Explanation

The **tagContent** table labels content to tags. It is a **N to N** relationship table.

Figure 5.7: Tag-content relationship.



2..20 groupSectionPermission

2..20.1 Code

```
1  ##################################################################################################
2  # TABLE
3  # * This table deals with the many-to-many group-section permissions
4  #   relationship.
5  ##################################################################################################
6  create table tbl_group_section_permission
7  (
8      # Primary key
9      id int auto_increment primary key,
10
11     # Relationship (group <-> section)
12     id_group int not null,
13     id_section int not null,
14
15     # Data
16     can_view boolean not null,
17     can_post boolean not null,
18     can_create_thread boolean not null,
19     can_delete_post boolean not null,
20     can_delete_thread boolean not null,
21     can_delete_section boolean not null,
22
23     foreign key (id_group)
24         references tbl_group(id)
25         on update cascade
26         on delete cascade,
27
28     foreign key (id_section)
29         references tbl_section(id)
30         on update cascade
31         on delete cascade
32 )$
33 ##################################################################################################
```

2..20.2 Explanation

The **groupSectionPermission** table links groups to sections, giving users belonging to the selected group a set of permissions for the selected section. It is a **N to N** relationship table.

3. Stored procedures

To ensure **maximum performance** and to **minimize coupling** with the PHP backend, the logic of the forum system is, where possible, implemented with SQL **stored procedures**. A stored procedure is a subroutine available to applications that access a relational database system, and it is actually stored in the database data dictionary.

3..1 mkContent

3..1.1 Code

```
1  ##################################################################################################
2  # PROCEDURE
3  # * Create a content base and return its ID.
4  ##################################################################################################
5  create procedure mk_content_base
6  (
7      in v_id_author int,
8      out v_created_id int
9  )
10 begin
11     insert into tbl_content_base
12         (id_author, creation_timestamp)
13         values(v_id_author, now());
14 
15     set v_created_id := LAST_INSERT_ID();
16 end$
17 ##################################################################################################
18 
19 
20 
21 ##################################################################################################
22 # PROCEDURE
23 # * Create a content base + content thread.
24 ##################################################################################################
25 create procedure mk_content_thread
26 (
27     in v_id_author int,
28     in v_id_section int,
29     in v_title varchar(255)
30 )
31 begin
32     call mk_content_base(v_id_author, @out_id_base);
33 
34     insert into tbl_content_thread
```

```
        (id_base, id_section, title)
        values(@out_id_base, v_id_section, v_title);
end$
#################################################################################################



#################################################################################################
# PROCEDURE
# * Create a content base + content post.
#################################################################################################
create procedure mk_content_post
(
    in v_id_author int,
    in v_id_thread int,
    in v_contents text
)
begin
    call mk_content_base(v_id_author, @out_id_base);

    insert into tbl_content_post
        (id_base, id_thread, contents)
        values(@out_id_base, v_id_thread, v_contents);
end$
#################################################################################################



#################################################################################################
# PROCEDURE
# * Create a content base + content attachment.
#################################################################################################
create procedure mk_content_attachment
(
    in v_id_author int,
    in v_id_post int,
    in v_id_file_data int
)
begin
    call mk_content_base(v_id_author, @out_id_base);

    insert into tbl_content_attachment
        (id_base, id_post, id_file_data)
        values(@out_id_base, v_id_post, v_id_file_data);
end$
#################################################################################################
```

3..1.2 Explanation

The procedures in the code listed above deal with the creation of content. To create content, it is necessary to instantiate both a `content_base` row and a specialization data row. These procedures automatically create both the required rows and make sure they relate to each other correctly, thanks to the `LAST_INSERT_ID()` MySQL function.

- `mk_content_base`: creates a content base record and returns its id.
- `mk_content_thread`: calls `mk_content_base`, then creates a thread specialization row linked to it. Takes the author id and title of the thread as input parameters.
- `mk_content_post`: calls `mk_content_base`, then creates a thread specialization row linked to it. Takes the author id and id of the parent thread as input parameters.
- `mk_content_attachment`: calls `mk_content_base`, then creates a thread specialization row linked to it. Takes the author id and id of the parent post as input parameters.

3..2 mkSubscription

3..2.1 Code

```
1  ##############################################################################################
2  # PROCEDURE
3  # * Create a subscription base and return its ID.
4  ##############################################################################################
5  create procedure mk_subscription_base
6  (
7      in v_id_subscriptor int,
8      out v_created_id int
9  )
10 begin
11     insert into tbl_subscription_base
12         (id_subscriptor, creation_timestamp, active)
13         values(v_id_subscriptor, now(), true);
14
15     set v_created_id := LAST_INSERT_ID();
16 end$
17 ##############################################################################################
18
19
20
21 ##############################################################################################
22 # PROCEDURE
23 # * Create a subscription base + subscription user.
24 ##############################################################################################
25 create procedure mk_subscription_user
```

```
(
    in v_id_subscriptor int,
    in v_id_user int
)
begin
    call mk_subscription_base(v_id_subscriptor, @out_id_base);

    insert into tbl_subscription_user
        (id_base, id_user)
        values(@out_id_base, v_id_user);
end$
##################################################################################################



##################################################################################################
# PROCEDURE
# * Create a subscription base + subscription thread.
##################################################################################################
create procedure mk_subscription_thread
(
    in v_id_subscriptor int,
    in v_id_thread int
)
begin
    call mk_subscription_base(v_id_subscriptor, @out_id_base);

    insert into tbl_subscription_thread
        (id_base, id_thread)
        values(@out_id_base, v_id_thread);
end$
##################################################################################################



##################################################################################################
# PROCEDURE
# * Create a subscription base + subscription tag.
##################################################################################################
create procedure mk_subscription_tag
(
    in v_id_subscriptor int,
    in v_id_tag int
)
begin
    call mk_subscription_base(v_id_subscriptor, @out_id_base);
```

```
72
73      insert into tbl_subscription_tag
74          (id_base, id_tag)
75          values(@out_id_base, v_id_tag);
76  end$
77  ##############################################################################################
```

3..2.2 Explanation

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

3..3 mkNotification

3..3.1 Code

```
1  ##############################################################################################
2  # PROCEDURE
3  # * Create a notification base and return its ID.
4  ##############################################################################################
5  create procedure mk_notification_base
6  (
7      in v_id_receiver int,
8      out v_created_id int
9  )
10 begin
11     insert into tbl_notification_base
12         (id_receiver, seen, creation_timestamp)
13         values(v_id_receiver, false, now());
14
15     set v_created_id := LAST_INSERT_ID();
16 end$
17 ##############################################################################################
18
19
20
21 ##############################################################################################
22 # PROCEDURE
23 # * Create a notification base + notification user.
24 ##############################################################################################
25 create procedure mk_notification_user
26 (
27     in v_id_receiver int,
28     in v_id_subscription_user int,
29     in v_id_content int
```

```
)
begin
    call mk_notification_base(v_id_receiver, @out_id_base);

    insert into tbl_notification_user
        (id_base, id_subscription_user, id_content)
        values(@out_id_base, v_id_subscription_user, v_id_content);
end$
##################################################################################################



##################################################################################################
# PROCEDURE
# * Create a notification base + notification thread.
##################################################################################################
create procedure mk_notification_thread
(
    in v_id_receiver int,
    in v_id_subscription_thread int,
    in v_id_post int
)
begin
    call mk_notification_base(v_id_receiver, @out_id_base);

    insert into tbl_notification_thread
        (id_base, id_subscription_thread, id_post)
        values(@out_id_base, v_id_subscription_thread, v_id_post);
end$
##################################################################################################



##################################################################################################
# PROCEDURE
# * Create a notification base + notification tag.
##################################################################################################
create procedure mk_notification_tag
(
    in v_id_receiver int,
    in v_id_subscription_tag int
)
begin
    call mk_notification_base(v_id_receiver, @out_id_base);

    insert into tbl_notification_tag
```

```
76          (id_base, id_subscription_tag)
77          values(@out_id_base, v_id_subscription_tag);
78  end$
79  ##############################################################################################
```

3..3.2 Explanation

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

3..4 utils

3..4.1 Code

```
1  ##############################################################################################
2  # PROCEDURE
3  # * Return the subscriptor ID from a subscription base ID.
4  ##############################################################################################
5  create procedure get_subscriptor
6  (
7      in v_id_base int,
8      out v_id_subscriptor int
9  )
10 begin
11     select id_subscriptor
12     into v_id_subscriptor
13     from tbl_subscription_base
14     where id = v_id_base;
15 end$
16 ##############################################################################################
17
18
19
20 ##############################################################################################
21 # PROCEDURE
22 # * Returns true if an unseen notification user with a specific
23 #   subscriptor ID and a specific user ID exists.
24 ##############################################################################################
25 create procedure check_notification_unseen_existance_user
26 (
27     in v_id_subscriptor int,
28     in v_id_user int,
29     out v_result boolean
```

```
)
begin
    set v_result := exists
    (
        select tb.id_receiver, tb.seen, ts.id_user
        from tbl_notification_base as tb
            inner join tbl_notification_user as td on tb.id = td.id_base
            inner join tbl_subscription_user as ts on td.id_subscription_user = ts.id
        where
            tb.seen = false
            and tb.id_receiver = v_id_subscriptor
            and ts.id_user = v_id_user
    );
end$
##################################################################################################



##################################################################################################
# PROCEDURE
# * Returns true if an unseen notification thread with a specific
#   subscriptor ID and a specific thread ID exists.
##################################################################################################
create procedure check_notification_unseen_existance_thread
(
    in v_id_subscriptor int,
    in v_id_thread int,
    out v_result boolean
)
begin
    set v_result := exists
    (
        select tb.id_receiver, tb.seen, ts.id_thread
        from tbl_notification_base as tb
            inner join tbl_notification_thread as td on tb.id = td.id_base
            inner join tbl_subscription_thread as ts on td.id_subscription_thread = ts.id
        where
            tb.seen = false
            and tb.id_receiver = v_id_subscriptor
            and ts.id_thread = v_id_thread
    );
end$
##################################################################################################



```

```
76  ##############################################################################################
77  # PROCEDURE
78  # * Returns true if an unseen notification user with a specific
79  #   subscriptor ID and a specific tag ID exists.
80  ##############################################################################################
81  create procedure check_notification_unseen_existance_tag
82  (
83      in v_id_subscriptor int,
84      in v_id_tag int,
85      out v_result boolean
86  )
87  begin
88      set v_result := exists
89      (
90          select tb.id_receiver, tb.seen, ts.id_tag
91          from tbl_notification_base as tb
92              inner join tbl_notification_tag as td on tb.id = td.id_base
93              inner join tbl_subscription_tag as ts on td.id_subscription_tag = ts.id
94          where
95              tb.seen = false
96              and tb.id_receiver = v_id_subscriptor
97              and ts.id_user = v_id_tag
98      );
99  end$
100 ##############################################################################################
```

3..4.2 Explanation

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

3..5 gNUser

3..5.1 Code

```
1  ##############################################################################################
2  # PROCEDURE
3  # * Generate notifications for every subscriber to the author of the
4  #   last created content.
5  ##############################################################################################
6  create procedure generate_notifications_user
7  (
8      in v_last_content_id int,
9      in v_last_content_author int
10 )
11 begin
12     declare loop_done int default false;
13     declare var_id_sub, var_id_sub_base, var_id_sub_tracked_user,
14             current_id_subscriptor int;
15     declare itr cursor for select id, id_base, id_user from tbl_subscription_user;
16     declare continue handler for not found set loop_done = true;
17
18     open itr;
19
20     label_loop:
21     loop
22         fetch itr into var_id_sub, var_id_sub_base, var_id_sub_tracked_user;
23
24         if loop_done then
25             leave label_loop;
26         end if;
27
28         if var_id_sub_tracked_user = v_last_content_author then
29             call get_subscriptor(var_id_sub_base, current_id_subscriptor);
```

```
30              call mk_notification_user(current_id_subscriptor, var_id_sub,
                ↪  v_last_content_id);
31          end if;
32      end loop;
33
34      close itr;
35  end$
36  ##############################################################################################
```

3..5.2 Explanation

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

3..6 gNThread

3..6.1 Code

```
1  ##############################################################################################
2  # PROCEDURE
3  # * Generate notifications for every subscriber to the thread of the
4  #   last created post.
5  ##############################################################################################
6  create procedure generate_notifications_thread
7  (
8      in v_last_post_id int,
9      in v_last_post_thread int
10 )
11 begin
12     declare loop_done int default false;
13     declare var_id_sub, var_id_sub_base, var_id_sub_tracked_thread,
14             current_id_subscriptor int;
15     declare itr cursor for select id, id_base, id_thread from tbl_subscription_thread;
16     declare continue handler for not found set loop_done = true;
17
18     open itr;
19
20     label_loop:
21     loop
22         fetch itr into var_id_sub, var_id_sub_base, var_id_sub_tracked_thread;
23
24         if loop_done then
25             leave label_loop;
26         end if;
27
28         if var_id_sub_tracked_thread = v_last_post_thread then
29             call get_subscriptor(var_id_sub_base, current_id_subscriptor);
```

```
30              call mk_notification_thread(current_id_subscriptor, var_id_sub,
                ↪  v_last_post_id);
31          end if;
32      end loop;
33
34      close itr;
35  end$
36  ##############################################################################################
```

3..6.2 Explanation

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

3..7 gNTag

3..7.1 Code

```
1  ##############################################################################################
2  # PROCEDURE
3  # * Generate notifications for every subscriber to the tag of the
4  #   last created content.
5  ##############################################################################################
6  create procedure generate_notifications_tag
7  (
8      in v_last_tc_tag int, # TODO: use
9      in v_last_tc_content int
10 )
11 begin
12     declare loop_done int default false;
13     declare var_id_sub, var_id_sub_base, var_id_sub_tracked_tag,
14             current_id_subscriptor int;
15     declare itr cursor for select id, id_base, id_tag from tbl_subscription_tag;
16     declare continue handler for not found set loop_done = true;
17
18     open itr;
19
20     label_loop:
21     loop
22         fetch itr into var_id_sub, var_id_sub_base, var_id_sub_tracked_tag;
23
24         if loop_done then
25             leave label_loop;
26         end if;
27
28         if var_id_sub_tracked_tag = v_last_tc_tag then
29             call get_subscriptor(var_id_sub_base, current_id_subscriptor);
```

```
30              call mk_notification_tag(current_id_subscriptor, var_id_sub);
31          end if;
32      end loop;
33
34      close itr;
35  end$
36  ##############################################################################################
```

3..7.2 Explanation

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

3..8 calcPrivs

3..8.1 Code

```
1  ##################################################################################################
2  # PROCEDURE
3  # * Calculate the final privileges of a user by inheriting them from the group hierarchy
4  # * they belong to.
5  ##################################################################################################
6  create procedure calculate_final_privileges
7  (
8      in v_id_user int,
9      out v_is_superadmin boolean,
10     out v_can_manage_sections boolean,
11     out v_can_manage_users boolean,
12     out v_can_manage_groups boolean,
13     out v_can_manage_permissions boolean
14 )
15 begin
16     # Set initial out values
17     set v_is_superadmin := false;
18     set v_can_manage_sections := false;
19     set v_can_manage_users := false;
20     set v_can_manage_groups := false;
21     set v_can_manage_permissions := false;
22
23     # Get user group
24     select id_group
25     into @current_id_group
26     from tbl_user
27     where id = v_id_user;
28
29     # Traverse the hierarchy and set privileges
```

```
    label_loop:
    loop
        set @last_id_group := @current_id_group;

        select id_parent, is_superadmin, can_manage_sections,
               can_manage_users, can_manage_groups, can_manage_permissions
        into @current_id_group, @p0, @p1, @p2, @p3, @p4
        from tbl_group
        where id = @last_id_group;

        set v_is_superadmin := v_is_superadmin or @p0;
        set v_can_manage_sections := v_can_manage_sections or @p1;
        set v_can_manage_users := v_can_manage_users or @p2;
        set v_can_manage_groups := v_can_manage_groups or @p3;
        set v_can_manage_permissions := v_can_manage_permissions or @p4;

        if @current_id_group is null then
            leave label_loop;
        end if;
    end loop;
end$
##############################################################################################
```

3..8.2 Explanation

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

3..9 calcPerms

3..9.1 Code

```
1  ##############################################################################################
2  # PROCEDURE
3  # * Calculate the final permissions of a user by inheriting them from the group hierarchy
4  # * they belong to, towards a specific section.
5  ##############################################################################################
6  create procedure calculate_final_permissions
7  (
8      in v_id_user int,
9      in v_id_section int,
10     out v_can_view boolean,
11     out v_can_post boolean,
12     out v_can_create_thread boolean,
13     out v_can_delete_post boolean,
14     out v_can_delete_thread boolean,
15     out v_can_delete_section boolean
16 )
17 begin
18     # Set initial out values
19     set v_can_view := false;
20     set v_can_post := false;
21     set v_can_create_thread := false;
22     set v_can_delete_post := false;
23     set v_can_delete_thread := false;
24     set v_can_delete_section := false;
25
26     # Get user group
27     select id_group
28     into @current_id_group
29     from tbl_user
```

```
30      where id = v_id_user;
31
32      # Traverse the hierarchy and set permissions
33      label_loop:
34      loop
35          set @last_id_group := @current_id_group;
36
37          select id_parent
38          into @current_id_group
39          from tbl_group
40          where id = @last_id_group;
41
42          select can_view, can_post, can_create_thread,
43                 can_delete_post, can_delete_thread, can_delete_section
44          into @p0, @p1, @p2, @p3, @p4, @p5
45          from tbl_group_section_permission
46          where id_group = @last_id_group and id_section = v_id_section;
47
48          set v_can_view := v_can_view or @p0;
49          set v_can_post := v_can_post or @p1;
50          set v_can_create_thread := v_can_create_thread or @p2;
51          set v_can_delete_post := v_can_delete_post or @p3;
52          set v_can_delete_thread := v_can_delete_thread or @p4;
53          set v_can_delete_section := v_can_delete_section or @p5;
54
55          if @current_id_group is null then
56              leave label_loop;
57          end if;
58      end loop;
59  end$
60  ##############################################################################################
```

3..9.2 Explanation

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

4. Triggers

4..1 notifications

4..1.1 Code

```
##############################################################################################
# TRIGGER
# * Generate notifications for user subscriptions after content
#   creation.
##############################################################################################
create trigger trg_notifications_user
    after insert on tbl_content_base
    for each row
begin
    call generate_notifications_user(NEW.id, NEW.id_author);
end$
##############################################################################################



##############################################################################################
# TRIGGER
# * Generate notifications for thread subscriptions after post
#   creation.
##############################################################################################
create trigger trg_notifications_thread
    after insert on tbl_content_post
    for each row
begin
    call generate_notifications_thread(NEW.id, NEW.id_thread);
end$
##############################################################################################



##############################################################################################
# TRIGGER
# * Generate notifications for tag subscriptions after content
#   creation.
##############################################################################################
create trigger trg_notifications_tag
    after insert on tbl_tag_content
    for each row
begin
    call generate_notifications_tag(NEW.id_tag, NEW.id_content);
```

```
41 end$
42 ##############################################################################
```

4..1.2 Explanation

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

4..2 contentBase

4..2.1 Code

```
1  ##############################################################################################
2  # TRIGGER
3  # * Delete content base left behind by derived content types.
4  ##############################################################################################
5  create trigger trg_del_content_base_thread
6      after delete on tbl_content_thread
7      for each row
8  begin
9      delete from tbl_content_base
10     where id = OLD.id_base;
11 end$
12 ##############################################################################################
13
14
15
16 ##############################################################################################
17 # TRIGGER
18 # * Delete content base left behind by derived content types.
19 ##############################################################################################
20 create trigger trg_del_content_base_post
21     after delete on tbl_content_post
22     for each row
23 begin
24     delete from tbl_content_base
25     where id = OLD.id_base;
26 end$
27 ##############################################################################################
28
29
```

```
30
31  ##################################################################################################
32  # TRIGGER
33  # * Delete content base left behind by derived content types.
34  ##################################################################################################
35  create trigger trg_del_content_base_attachment
36      after delete on tbl_content_attachment
37      for each row
38  begin
39      delete from tbl_content_base
40      where id = OLD.id_base;
41  end$
42  ##################################################################################################
43
44
45
46
47
48
49
50
51
52
53  ##################################################################################################
54  # TRIGGER
55  # * TODO
56  ##################################################################################################
57  create trigger trg_del_ntf_user_on_post_del
58      before delete on tbl_content_base
59      for each row
60  begin
61      delete from tbl_notification_user
62      where id_content = OLD.id;
63  end$
64  ##################################################################################################
65
66
67  ##################################################################################################
68  # TRIGGER
69  # * TODO
70  ##################################################################################################
71  create trigger trg_del_ntf_thread_on_post_del
72      before delete on tbl_content_post
73      for each row
74  begin
75      delete from tbl_notification_thread
```

```
76      where id_post = OLD.id;
77  end$
78  ##############################################################################################
```

4..2.2 Explanation

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

4..3 subscriptionBase

4..3.1 Code

```
1  ##############################################################################################
2  # TRIGGER
3  # * Delete subscription base left behind by derived subscription types.
4  ##############################################################################################
5  create trigger trg_del_subscription_base_thread
6      after delete on tbl_subscription_thread
7      for each row
8  begin
9      delete from tbl_subscription_base
10     where id = OLD.id_base;
11 end$
12 ##############################################################################################
13
14
15
16 ##############################################################################################
17 # TRIGGER
18 # * Delete subscription base left behind by derived subscription types.
19 ##############################################################################################
20 create trigger trg_del_subscription_base_user
21     after delete on tbl_subscription_user
22     for each row
23 begin
24     delete from tbl_subscription_base
25     where id = OLD.id_base;
26 end$
27 ##############################################################################################
28
29
```

```
30
31 ##############################################################################################
32 # TRIGGER
33 # * Delete subscription base left behind by derived subscription types.
34 ##############################################################################################
35 create trigger trg_del_subscription_base_tag
36     after delete on tbl_subscription_tag
37     for each row
38 begin
39     delete from tbl_subscription_base
40     where id = OLD.id_base;
41 end$
42 ##############################################################################################
```

4..3.2 Explanation

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

4..4 notificationBase

4..4.1 Code

```
1  ##############################################################################################
2  # TRIGGER
3  # * Delete notification base left behind by derived notification types.
4  ##############################################################################################
5  create trigger trg_del_notification_base_thread
6      after delete on tbl_notification_thread
7      for each row
8  begin
9      delete from tbl_notification_base
10     where id = OLD.id_base;
11 end$
12 ##############################################################################################
13
14
15
16 ##############################################################################################
17 # TRIGGER
18 # * Delete notification base left behind by derived notification types.
19 ##############################################################################################
20 create trigger trg_del_notification_base_user
21     after delete on tbl_notification_user
22     for each row
23 begin
24     delete from tbl_notification_base
25     where id = OLD.id_base;
26 end$
27 ##############################################################################################
28
29
```

```
30
31 ##############################################################################################
32 # TRIGGER
33 # * Delete notification base left behind by derived notification types.
34 ##############################################################################################
35 create trigger trg_del_notification_base_tag
36     after delete on tbl_notification_tag
37     for each row
38 begin
39     delete from tbl_notification_base
40     where id = OLD.id_base;
41 end$
42 ##############################################################################################
```

4..4.2 Explanation

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

4..5 subscriptionNtf

4..5.1 Code

```
1  ##############################################################################################
2  # TRIGGER
3  # * Delete notifications that point to the deleted subscription.
4  ##############################################################################################
5  create trigger trg_del_subscription_ntf_thread
6      before delete on tbl_subscription_thread
7      for each row
8  begin
9      delete from tbl_notification_thread
10     where id_subscription_thread = OLD.id;
11 end$
12 ##############################################################################################
13
14
15
16 ##############################################################################################
17 # TRIGGER
18 # * Delete notifications that point to the deleted subscription.
19 ##############################################################################################
20 create trigger trg_del_subscription_ntf_user
21     before delete on tbl_subscription_user
22     for each row
23 begin
24     delete from tbl_notification_user
25     where id_subscription_user = OLD.id;
26 end$
27 ##############################################################################################
28
29
```

```
30
31 ##############################################################################################
32 # TRIGGER
33 # * Delete notifications that point to the deleted subscription.
34 ##############################################################################################
35 create trigger trg_del_subscription_ntf_tag
36     before delete on tbl_subscription_tag
37     for each row
38 begin
39     delete from tbl_notification_tag
40     where id_subscription_tag = OLD.id;
41 end$
42 ##############################################################################################
```

4..5.2 Explanation

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

4..6 delSubCnt

4..6.1 Code

```
1  ##############################################################################################
2  # TRIGGER
3  # * TODO
4  ##############################################################################################
5  create trigger trg_del_subscription_cnt_thread
6      before delete on tbl_content_thread
7      for each row
8  begin
9      delete from tbl_subscription_thread
10     where id_thread = OLD.id;
11 end$
12 ##############################################################################################
13
14
15 ##############################################################################################
16 # TRIGGER
17 # * TODO
18 ##############################################################################################
19 create trigger trg_del_subscription_cnt_user
20     before delete on tbl_user
21     for each row
22 begin
23     delete from tbl_subscription_user
24     where id_user = OLD.id;
25 end$
26 ##############################################################################################
27
28
29
```

```
30 ##############################################################################################
31 # TRIGGER
32 # * TODO
33 ##############################################################################################
34 create trigger trg_del_subscription_cnt_tag
35     before delete on tbl_tag
36     for each row
37 begin
38     delete from tbl_subscription_tag
39     where id_tag = OLD.id;
40 end$
41 ##############################################################################################
```

4..6.2 Explanation

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

5. Database inizialization

5..1 initialize

5..1.1 Code

```
1  ##############################################################################################
2  # PROCEDURE
3  # * Initialization procedure
4  # * Create necessary data for veeForum initalization
5  ##############################################################################################
6  create procedure initialize_veeForum()
7  begin
8      # Create Superadmin group (ID: 1)
9      insert into tbl_group
10         (id_parent, name, is_superadmin, can_manage_sections, can_manage_users,
11              can_manage_groups, can_manage_permissions)
12         values(null, 'Superadmin', true, true, true, true, true);
13
14     # Create Basic group (ID: 2) (default registration group)
15     insert into tbl_group
16         (id_parent, name, is_superadmin, can_manage_sections, can_manage_users,
17             can_manage_groups, can_manage_permissions)
18         values(null, 'Basic', false, false, false, false, false);
19
20     # Create SuperAdmin user (ID: 1) with (admin, admin) credentials
21     insert into tbl_user
22         (id_group, username, password_hash, email, registration_date, firstname,
23             lastname, birth_date)
24         values(1, 'admin', '21232f297a57a5a743894a0e4a801fc3',
25             'vittorio.romeo@outlook.com', curdate(), 'Vittorio', 'Romeo', curdate());
26
27     # Insert log message with the date of the forum framework installation
28     insert into tbl_log
29         (type, creation_timestamp, value)
30         values(0, now(), 'veeForum initialized');
31 end$
32 ##############################################################################################
33
34
35
36 ##############################################################################################
37 # PROCEDURE
38 # * Testing procedure
39 # * Create some test data to speed up development/testing
40 ##############################################################################################
```

```
41  create procedure create_test_data()
42  begin
43      insert into tbl_user
44          (id_group, username, password_hash, email, registration_date)
45          values(2, 'user1', 'pass1', 'email1', curdate());
46
47      insert into tbl_user
48          (id_group, username, password_hash, email, registration_date)
49          values(2, 'user2', 'pass2', 'email2', curdate());
50
51      insert into tbl_section
52          (id_parent, name)
53          values(null, 'section1');
54
55      insert into tbl_group_section_permission
56          (id_group, id_section, can_view, can_post, can_create_thread, can_delete_post,
57              can_delete_thread, can_delete_section)
58          values(1, 1, true, true, true, true, true, true);
59
60      call mk_subscription_user(2, 3);
61  end$
62  ##############################################################################################
63
64
65
66  ##############################################################################################
67  # COMMANDS
68  # * Initial commands required to set up veeForum
69  ##############################################################################################
70  call initialize_veeForum()$
71  call create_test_data()$
72  ##############################################################################################
73
74
75
76  ##############################################################################################
77  # Copyright (c) 2013-2015 Vittorio Romeo
78  # License: Academic Free License ("AFL") v. 3.0
79  # AFL License page: http://opensource.org/licenses/AFL-3.0
80  ##############################################################################################
81  # http://vittorioromeo.info
82  # vittorio.romeo@outlook.com
83  ##############################################################################################
```

5..1.2 Explanation

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Chapter 6

Installation

aaa

Chapter 7

Conceptual model

aaa

Chapter 8

Logical model

aaa

Chapter 9

Table details

aaa

Chapter 10

Web interface

aaa

Chapter 11

Sample queries

aaa

Part III

Conclusion

aaa

Chapter 12

Final product

aaa

Chapter 13

What I learned

aaa

Chapter 14

Future

aaa

Chapter 15

References

aaa