Chamilo LMS - Feature #7257

Split c_item_property into c_item_property and c_item_property_log

02/09/2014 13:49 - Yannick Warnier

Status:	Assigned	**Start date:**	02/09/2014
Priority:	Urgent	**Due date:**	
Assignee:	Julio Montoya	**% Done:**	0%
Category:	Database & API changes	**Estimated time:**	0.00 hour
Target version:	2.0	**Spent time:**	0.70 hour
Complexity:	Wizard-level	**SCRUM pts - complexity:**	40

Description

The item_property table is a mess, in particular because it actually serves (in a bad way) two purposes: it stores visibility information about resources and it track the changes on every item.
This makes distinguishing both and generating reports a mess (comment extracted from #5157)

The current table looks like this:
mysql> describe c_item_property;

Field	Type	Null	Key	Default	Extra
c_id	int(11)	NO	PRI	NULL	
id	int(11)	NO	PRI	NULL	auto_increment
tool	varchar(100)	NO	MUL		a literal identifying the tool (and as such the table)
insert_user_id	int(10) unsigned	NO		0	
insert_date	datetime	NO		0000-00-00 00:00:00	
lastedit_date	datetime	NO		0000-00-00 00:00:00	
ref	int(11)	NO		0	the id of the item inside the referred tool's table
lastedit_type	varchar(100)	NO			
lastedit_user_id	int(10) unsigned	NO		0	
to_group_id	int(10) unsigned	YES		NULL	
to_user_id	int(10) unsigned	YES		NULL	
visibility	tinyint(4)	NO		1	
start_visible	datetime	NO		0000-00-00 00:00:00	
end_visible	datetime	NO		0000-00-00 00:00:00	
id_session	int(11)	NO		0	

The idea (for v10) is to have two tables:

- one with only information aobut the item and its attributes (item, maybe?)
- one with only historical information about the changes on this item (item_changelog?)

item

A proposed definition (this must be tested) should be like follows:

mysql> describe item;

Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	NULL	auto_increment
tool_id	int(11)	NO		NULL	an ID identifying the tool (and as such the table)
ref_id	int(11)	NO		0	the iid of the item inside the referred tool's table
branch_id	int(11)	NO		NULL	
c_id	int(11)	NO		NULL	
session_id	int(11)	NO		0	
insert_user_id	int(10) unsigned	NO		0	
insert_date	datetime	NO		0000-00-00 00:00:00	
lastedit_date	datetime	NO		0000-00-00 00:00:00	
lastedit_user_id	int(10) unsigned	NO		0	
lastedit_type	varchar(100)	NO			
to_group_id	int(10) unsigned	YES		NULL	
to_user_id	int(10) unsigned	YES		NULL	
visibility	tinyint(4)	NO		1	
start_visible	datetime	NO		0000-00-00 00:00:00	
end_visible	datetime	NO		0000-00-00 00:00:00	
----------	----------	------	-----	----------	----------

So the fields here are **practically** untouched (apart from the order and names of the fields):

- c_id is not mandatory anymore, which allows us to later on associate the item to other courses (this is a first step for a future enhancement)
- tool_id (an integer) replaces the "tool" name literal (allowing for faster indexing) **but** we will need a general "tool" or "tool_definition" table to give an ID to each tool (including tools created by a plugin)
- ref_id replaces the previous "ref" and points directly to an item into the table
- session_id instead of id_session (we want to get rid of all "id_session" and replace them by session_id all over chamilo
- the rest is straightforward

item_changelog

This table stores the changes made to any item by anyone. It is assumed this table will be very large, but with this change in structure, it should not be read very often.

Given the unique ID in the "item" table, it would not seem necessary to store the course and session IDs. However, there might be cases in the future where we move an item from one course to another. Because of this, we have to consider an origin and destination course and session.
Reflecting on one change we had to do for one specific Chamilo implementation (branch_transaction), it would be a good idea to be able to reuse this table to replicate most changes to items. As such, we need to keep good track of what is done and define types of changes to have kind of a dictionary for that.

As a rule of thumb, a ref_id (and the tool_id) should never change by the effects of a "change" on an item. If the item changes tool, then it must be deleted and re-created. As such, we don't need to keep track of tool_id and ref_id here.

mysql> describe item_changelog;

Field	Type	Null	Key	Default	Extra

id	bigint	NO	PRI	NULL	auto_increment
item_id	int(11)				
action_id	int(11)				
branch_id	int(11)				
c_id	int(11)	NO		NULL	
session_id	int(11)	NO		0	
action_date	datetime	NO		0000-00-00 00:00:00	
action_user_id	int(10) unsigned	NO		0	
action_access_url_id	int(10) unsigned	NO		0	
details	text	NO		0000-00-00 00:00:00	

Obviously this table should be indexed properly and all queries should preferrably follow the index order to speed it up.

Reasons for branch_id and access_url_id

If we want to make these tables thoroughly resistant to future changes, they should also have a branch_id (ID of the branch of the organization on which it is installed), which will come from the branch table on multiple-branch organizations' Chamilo installations.

The access_url_id field is, however, already defined through the session_id, by restraint, although it would to have it in the item_changelog table (although not sql-normal) to reduce the weight of querying that table on one specific portal.

Placing these fields in these tables will actually remove a lot of weight from other tables, as otherwise the fields might need to be added to many other item tables.

Ideas for a migration script

The migration script should look like this (I guess):

```
CREATE TABLE item_changelog (
 ...
);
ALTER TABLE c_item_property RENAME item (
 ...
);
```

Then for each item in the item table, it should create a first record in item_changelog.

Notes

There is currently a track_e_item_property table intended for the same purpose of item_changelog, so track_e_item_property should be changed to look like item_changelog, probably. This hasn't been looked into just yet.

Related issues:

Related to Chamilo LMS - Feature #8015: Manage resources	**New**	**23/12/2015**
Related to Chamilo LMS - Feature #7935: Migration from 1.10.x to 2.0 [master]	**Needs testing**	**02/11/2015**

History

#1 - 02/09/2014 14:13 - Yannick Warnier

- Description updated

#2 - 02/09/2014 14:21 - Yannick Warnier

- Description updated

- Status changed from New to Assigned

#3 - 02/09/2014 14:44 - Yannick Warnier

- Description updated

#4 - 03/09/2014 15:42 - Julio Montoya

- Assignee changed from Julio Montoya to Yannick Warnier

Well I have another structure proposition, is kind of the similar.

We have "items" in chamilo (notebook item, glossary item, document item, etc)
All of them have an id for example for notebook, it will be (id, name, description)
Then this item needs to "registered" or "shared" with a (a course, a course/group, course/session or to a user).

In order to do that we need to create a new table called "resource" with basic static contents:

```
mysql> describe resource_node;
+-------------+---------------+------+-----+---------+----------------+
| Field       | Type          | Null | Key | Default | Extra          |
+-------------+---------------+------+-----+---------+----------------+
| id          | int(11)       | NO   | PRI | NULL    | auto_increment |
| creator_id  | int(11)       | NO   | MUL | NULL    |                |
| tool        | varchar(255)  | NO   |     | NULL    |                |
| name        | varchar(255)  | NO   |     | NULL    |                | <-we need the name of the object in or
der to create trees later
| resource_id | int(11)       | NO   |     | NULL    |                | <--- this is the notebook id.
+-------------+---------------+------+-----+---------+----------------+
```

In the notebook example, we will have just a table with an id an description, because the rest will be provided by the system:

Notebook:

```
describe c_notebook;
+-------------+--------------+------+-----+---------+----------------+
| Field       | Type         | Null | Key | Default | Extra          |
+-------------+--------------+------+-----+---------+----------------+
| id          | int(11)      | NO   | PRI | NULL    | auto_increment |
| description | longtext     | NO   |     | NULL    |                |
+-------------+--------------+------+-----+---------+----------------+
```

So in order to have notebook (id, description), the rest of the information will be saved there.

This new item need to be related with something. So that's why the "resource_link" need to be created.
For every new "connection" a new row must be created.

```
mysql> describe resource_link;
+------------------+----------+------+-----+---------+----------------+
| Field            | Type     | Null | Key | Default | Extra          |
+------------------+----------+------+-----+---------+----------------+
| id               | int(11)  | NO   | PRI | NULL    | auto_increment |
| resource_node_id | int(11)  | YES  | UNI | NULL    |                |
| created_at       | datetime | NO   |     | NULL    |                |
| updated_at       | datetime | NO   |     | NULL    |                |
| session_id       | int(11)  | NO   | MUL | NULL    |                |
| user_id          | int(11)  | NO   | MUL | NULL    |                |
| c_id             | int(11)  | NO   | MUL | NULL    |                |
| group_id         | int(11)  | YES  |     | NULL    |                |
| branch_id        | int(11)  | YES  |     | NULL    |                |
| url_id           | int(11)  | YES  |     | NULL    |                |
+------------------+----------+------+-----+---------+----------------+
8 rows in set (0.00 sec)
```

The visibility will be **only affected** to a resource_link and not to main object.

```
mysql> describe resource_visibility;
+------------------+----------+------+-----+---------+----------------+
| Field            | Type     | Null | Key | Default | Extra          |
+------------------+----------+------+-----+---------+----------------+
| id               | int(11)  | NO   | PRI | NULL    | auto_increment |
| resource_link_id | int(11)  | YES  | UNI | NULL    |                |
| visibility       | int(11)  | NO   | MUL | NULL    |                |
| start_visibility_at int(11) | NO   | MUL | NULL    |                |
| end_visibility_at| int(11)  | NO   | MUL | NULL    |                |
```

```
| created_at       | datetime | NO   |     | NULL    |                |
| updated_at       | datetime | NO   |     | NULL    |                |
| created_by       | string   | YES  |     | NULL    |                |
| updated_by       | string   | YES  |     | NULL    |                |
+------------------+----------+------+-----+---------+----------------+
8 rows in set (0.00 sec)
```

Then we can create another table called "logs", with the recent changes made to that link and why:

```
mysql> describe resource_log;
+------------------+----------+------+-----+---------+----------------+
| Field            | Type     | Null | Key | Default | Extra          |
+------------------+----------+------+-----+---------+----------------+
| id               | int(11)  | NO   | PRI | NULL    | auto_increment |
| resource_link_id | int(11)  | YES  | UNI | NULL    |                |
| creator_id       | string   | YES  |     | NULL    |                |
| action           | string   | YES  |     | NULL    |                |
| text             | string   | YES  |     | NULL    |                |
| created_at       | datetime | NO   |     | NULL    |                |
+------------------+----------+------+-----+---------+----------------+
8 rows in set (0.00 sec)
```

We can also have another table for resource_roles:

```
mysql> describe resource_roles;
+------------------+----------+------+-----+---------+----------------+
| Field            | Type     | Null | Key | Default | Extra          |
+------------------+----------+------+-----+---------+----------------+
| id               | int(11)  | NO   | PRI | NULL    | auto_increment |
| resource_link_id | int(11)  | YES  | UNI | NULL    |                |
| roles             | string   | YES  |     | NULL    |                |

8 rows in set (0.00 sec)
```

Well, the "roles" could be saved in the resource_visibility ...

#5 - 03/09/2014 15:47 - Julio Montoya

- Status changed from Assigned to Needs more info

#6 - 05/09/2014 08:19 - Julio Montoya

I need feedback about this in order to continue.

The problem with that new table "item" will be the same with the current item_property, we will have many "states" for an object.
So no a unique object id for all the platform. What happens if I sent an announcement to a course in a group to 4 users? I think that right now, we will have like different item_properties ...

#7 - 05/09/2014 10:06 - Yannick Warnier

- Assignee changed from Yannick Warnier to Julio Montoya

OK, so let's split it into three different tables then:

- item to identify all items uniquely
- item_visibility (with item_id, c_id, session_id, group_id, user_id to identify for whom it is visible this way)
- item_changelog to identify all changes to an item (be them visibility or item properties)

Would that be enough?

#8 - 05/09/2014 10:17 - Julio Montoya

Yannick Warnier wrote:

> OK, so let's split it into three different tables then:
>
> - item to identify all items uniquely
> - item_visibility (with item_id, c_id, session_id, group_id, user_id to identify for whom it is visible this way)
> - item_changelog to identify all changes to an item (be them visibility or item properties)
>
> Would that be enough?

Well changing the table name will be better in order to migrate all item_property to the new structure, otherwise it could exist a mix between old and new structure ...

#9 - 08/09/2014 16:10 - Yannick Warnier

Julio Montoya wrote:

> Well I have another structure proposition, is kind of the similar.
>
> We have "items" in chamilo (notebook item, glossary item, document item, etc)
> All of them have an id for example for notebook, it will be (id, name, description)
> Then this item needs to "registered" or "shared" with a (a course, a course/group, course/session or to a user).
>
> In order to do that we need to create a new table called "resource" with basic static contents:
>
> [...]

The resource_node table should be **only** fixed data (no varchar) for speed purposes. If the name is required for trees, I believe we could limit this "name" field to a reduced version of the name, or remove the name completely and move it to the depending tables.

The "tool" column should be a numeric ID, and this ID should come from a "tool" table. See recent developments by Daniel Barreto in icpna:virtual branch, table sequence_type_entity:

```
CREATE TABLE IF NOT EXISTS `sequence_type_entity` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `name` varchar(50) NOT NULL DEFAULT '',
  `description` text,
  `ent_table` varchar(50) NOT NULL,
  PRIMARY KEY (`id`)
);
```

It's time we got a table to list the tools (or "entity types") and their tables, anyway.

> In the notebook example, we will have just a table with an id an description, because the rest will be provided by the system:
>
> Notebook:
>
> [...]
>
> So in order to have notebook (id, description), the rest of the information will be saved there.

Following what I said above, this table should also contain the full name (while the resource_node table contains only a short, 10 or 15-characters long) version of the name.

> This new item need to be related with something. So that's why the "resource_link" need to be created.
> For every new "connection" a new row must be created.
>
> [...]
>
> The visibility will be **only affected** to a resource_link and not to main object.
>
> [...]

I cannot see why the fields:

- visibility
- start_visibility_at
- end_visibility_at

cannot be part of the resource_link table. When would we have 2 visibilities for one resource link?

> Then we can create another table called "logs", with the recent changes made to that link and why:
>
> [...]
>
> We can also have another table for resource_roles:
>
> [...]
>
> Well, the "roles" could be saved in the resource_visibility ...

Agreed. I don't think we need another table for that.

In general, I'm OK but I'd like to:

- reduce the load on the central tables (in this case resource_node should be only fixed-length data and if possible should not include the name)
- avoid doing a lot of JOINs (in this case avoid creating resource_visibility and resource_roles, as they don't seem necessary)

This is why I was suggesting only item, item_visibility and item_changelog (and no intermediate item_link).

Any counter-argument?

#10 - 09/09/2014 09:25 - Julio Montoya

Yannick Warnier wrote:

> Julio Montoya wrote:
>
> > Well I have another structure proposition, is kind of the similar.
> >
> > We have "items" in chamilo (notebook item, glossary item, document item, etc)
> > All of them have an id for example for notebook, it will be (id, name, description)
> > Then this item needs to "registered" or "shared" with a (a course, a course/group, course/session or to a user).
> >
> > In order to do that we need to create a new table called "resource" with basic static contents:
> >
> > [...]
>
> The resource_node table should be **only** fixed data (no varchar) for speed purposes. If the name is required for trees, I believe we could limit this "name" field to a reduced version of the name, or remove the name completely and move it to the depending tables.
>
> The "tool" column should be a numeric ID, and this ID should come from a "tool" table. See recent developments by Daniel Barreto in icpna:virtual branch, table sequence_type_entity:

Well, I'm going to do a quick update how tools are working now at least, for the Notebook.
The tool creation is more easy now. You don't need a new table to register a tool.

For example, for the Notebook bundle, you create a bundle, you put "some" configuration in
src/Chamilo/NotebookBundle/Resources/config/services.yml

Then the "Course" during the creation process recognizes that there's a Notebook tool (based in some "symfony tags" in that services.yml file in this case it read the tag: "chamilo_course.tool").

If you want a list of tools you do "$container->get('chamilo_course.tool_chain');" and it will give you the list of course tools in the platform. Each tool has global (platform settings) and course settings out of the box. (You have to create the only the forms types).

So the creation of a new tool doesn't mean exactly that you need a tool table. There will be an option tool "enabled/disabled" or not by platform.

I need to have the tool name (name of the service) in order to process that object. It could be a char(100) some examples:

```
chamilo_notebook.tool.notebook
chamilo_course.tool.learning_path
pepito_namespace.tool.great_tool
```

> [...]
>
> It's time we got a table to list the tools (or "entity types") and their tables, anyway.

This could be avoided because there's no need to save any data of the tools (information is already in the class)
The settings of the tool are saved in the global settings (settings_current) and the course settings (c_course_setting)

> > In the notebook example, we will have just a table with an id an description, because the rest will be provided by the system:
> >
> > Notebook:
> >
> > [...]
> >
> > So in order to have notebook (id, description), the rest of the information will be saved there.
>
> Following what I said above, this table should also contain the full name (while the resource_node table contains only a short, 10 or 15-characters long) version of the name.

I agree with the difference of full and short name, but it should be a "slug name" and not short name. It should be as big as the original title otherwise there will be this kind of errors:

```
This is a test for a big title 1 -> this-is-a-test-for-a
This is a test for a big title 2 -> this-is-a-test-for-a
```

> This new item need to be related with something. So that's why the "resource_link" need to be created.
> For every new "connection" a new row must be created.
>
> [...]
>
> The visibility will be **only affected** to a resource_link and not to main object.
>
> [...]

I cannot see why the fields:

- visibility
- start_visibility_at
- end_visibility_at

cannot be part of the resource_link table. When would we have 2 visibilities for one resource link?

Yes, I'm agree with that. But you will loose the information when the visibility was changed or when the connection was created.
If that's not a problem then we can just put inside the table.

> Then we can create another table called "logs", with the recent changes made to that link and why:
>
> [...]
>
> We can also have another table for resource_roles:
>
> [...]
>
> Well, the "roles" could be saved in the resource_visibility ...

Agreed. I don't think we need another table for that.

So you don't whant to log the changes made for any resource_link?

In general, I'm OK but I'd like to:

- reduce the load on the central tables (in this case resource_node should be only fixed-length data and if possible should not include the name)
- avoid doing a lot of JOINs (in this case avoid creating resource_visibility and resource_roles, as they don't seem necessary)

Agree that I don't need the resource_visibility (could be merged with the resource_link) and resource_roles was a suggestion.
But if you want to keep track of the changes of any "resource_link" you will need a "resource_log" or a resource_audit(a bundle that handles the changes in a table).

"resource_link" could be renamed to "resource_conection". I'm not sure if we are going to save that old resource_link. It should be removed I think.

> This is why I was suggesting only item, item_visibility and item_changelog (and no intermediate item_link).

Your proposition is different. Your "item" table has all the information (session_id, group, user, etc) so in some cases I will have this:

```
id tool session_id course_id group_id user_id ref_id
1  doc  0             1         0         0      5
2  doc  1            1         0         0      5
```

Here the id is not important, the important is the ref_id. Is somehow like my resource_link, but you don't keep a "global repository" of objects like in the resource_node table.

#11 - 12/09/2014 11:05 - Julio Montoya

- Assignee changed from Julio Montoya to Yannick Warnier

#12 - 18/09/2014 14:30 - Julio Montoya

ping ...

#13 - 25/09/2014 10:10 - Julio Montoya

ping 2

#14 - 23/12/2015 15:59 - Yannick Warnier

Julio Montoya wrote:

> Yannick Warnier wrote:
>
> > Julio Montoya wrote:
> >
> > > Well I have another structure proposition, is kind of the similar.
> > >
> > > We have "items" in chamilo (notebook item, glossary item, document item, etc)
> > > All of them have an id for example for notebook, it will be (id, name, description)
> > > Then this item needs to "registered" or "shared" with a (a course, a course/group, course/session or to a user).
> > >
> > > In order to do that we need to create a new table called "resource" with basic static contents:
> > >
> > > [...]
> >
> > The resource_node table should be **only** fixed data (no varchar) for speed purposes. If the name is required for trees, I believe we could limit this "name" field to a reduced version of the name, or remove the name completely and move it to the depending tables.
> >
> > The "tool" column should be a numeric ID, and this ID should come from a "tool" table. See recent developments by Daniel Barreto in icpna:virtual branch, table sequence_type_entity:
>
> Well, I'm going to do a quick update how tools are working now at least, for the Notebook.
> The tool creation is more easy now. You don't need a new table to register a tool.
>
> For example, for the Notebook bundle, you create a bundle, you put "some" configuration in
> src/Chamilo/NotebookBundle/Resources/config/services.yml

In general terms, I don't really like the fact that things have to be configured in a Symfony yml file. I found these really cryptic. Let's see how it goes, but I have a bad feeling about them.

> Then the "Course" during the creation process recognizes that there's a Notebook tool (based in some "symfony tags" in that services.yml file in this case it read the tag: "chamilo_course.tool").

OK. What happens when you enable a new tool in a new version of Chamilo? Will we able to do the correct migration process to add this new tool to all courses? Or do the courses always check the services.yml, even the course already exist (when it's not being installed right now)?

> If you want a list of tools you do "$container->get('chamilo_course.tool_chain');" and it will give you the list of course tools in the platform. Each tool has global (platform settings) and course settings out of the box. (You only have to create the forms types).

What do you mean by form types?

> > [...]
> >
> > It's time we got a table to list the tools (or "entity types") and their tables, anyway.
>
> This could be avoided because there's no need to save any data of the tools (information is already in the class)
> The settings of the tool are saved in the global settings (settings_current) and the course settings (c_course_setting)

OK, fine by me.

> > In the notebook example, we will have just a table with an id an description, because the rest will be provided by the system:
> >
> > Notebook:
> >
> > [...]
> >
> > So in order to have notebook (id, description), the rest of the information will be saved there.
>
> Following what I said above, this table should also contain the full name (while the resource_node table contains only a short, 10 or 15-characters long) version of the name.

I agree with the difference of full and short name, but it should be a "slug name" and not short name. It should be as big as the original title otherwise there will be this kind of errors:

[...]

Agreed.

This new item need to be related with something. So that's why the "resource_link" need to be created.
For every new "connection" a new row must be created.

[...]

The visibility will be **only affected** to a resource_link and not to main object.

[...]

I cannot see why the fields:

- visibility
- start_visibility_at
- end_visibility_at

cannot be part of the resource_link table. When would we have 2 visibilities for one resource link?

Yes, I'm agree with that. But you will loose the information when the visibility was changed or when the connection was created.
If that's not a problem then we can just put inside the table.

I was thinking about having the historical information (logs) about visibility in the resource_logs table. To avoid having a table **just** for visibilities.
Let me put it another way:

- if you store visibility in the resource_node and resource_link table, then you only need one query to know if the item is currently visible or not. Historical information about who changed the visibility and when could be found in the resource_log table, which is the right place to keep logs.
- if you store the visibility in a table called resource_visibility, then:
  - you will have to JOIN resource_node (or resource_link) and resource_visibility **every time** you want to show something on screen
  - because of the logging of past actions, resource_visibility will get very big, very fast, and might slow down the requests on joined resource_node (or resource_link).

Then we can create another table called "logs", with the recent changes made to that link and why:

[...]

We can also have another table for resource_roles:

[...]

Well, the "roles" could be saved in the resource_visibility ...

Agreed. I don't think we need another table for that.

So you don't whant to log the changes made for any resource_link?

I do, I just think resource_logs (or resource_audit, the name and method do not matter much) can be used to store all changes to resource_node or resource_link.
Or better said: resource_node_logs + resource_link_logs.

In general, I'm OK but I'd like to:

- reduce the load on the central tables (in this case resource_node should be only fixed-length data and if possible should not include the name)
- avoid doing a lot of JOINs (in this case avoid creating resource_visibility and resource_roles, as they don't seem necessary)

Agree that I don't need the resource_visibility (could be merged with the resource_link) and resource_roles was a suggestion.
But if you want to keep track of the changes of any "resource_link" you will need a "resource_log" or a resource_audit(a bundle that handles the changes in a table).

"resource_link" could be renamed to "resource_connection". I'm not sure if we are going to save that old resource_link. It should be removed I think.

The name of the table should represent the context in which we use the resource. Why not resource_context?

> This is why I was suggesting only item, item_visibility and item_changelog (and no intermediate item_link).

> Your proposition is different. Your "item" table has all the information (session_id, group, user, etc) so in some cases I will have this:
>
> [...]
>
> Here the id is not important, the important is the ref_id. Is somehow like my resource_link, but you don't keep a "global repository" of objects like in the resource_node table.

OK

#15 - 23/12/2015 16:42 - Julio Montoya

> In general terms, I don't really like the fact that things have to be configured in a Symfony yml file. I found these really cryptic. Let's see how it goes, but I have a bad feeling about them.

ok

> OK. What happens when you enable a new tool in a new version of Chamilo? Will we able to do the correct migration process to add this new tool to all courses? Or do the courses always check the services.yml, even the course already exist (when it's not being installed right now)?

Well I don't remember that implementation was in chamilo experimental. But I guess yes.

> What do you mean by form types?

Well when you have a new tool in chamilo 1.9.x and you have to add new settings in the database you have to manually created using SQL. In this new version (if implemented) you create the form type for those settings, example:

```
$form->add('config_for_my_notebook_tool_1', 'text');
$form->add('config_for_my_notebook_tool_2', 'text');
```

> I was thinking about having the historical information (logs) about visibility in the resource_logs table. To avoid having a table just for visibilities.
> Let me put it another way:
> if you store visibility in the resource_node and resource_link table, then you only need one query to know if the item is currently visible or not. Historical information about who changed the visibility and when could be found in the resource_log table, which is the right place to keep logs.
> if you store the visibility in a table called resource_visibility, then:
> you will have to JOIN resource_node (or resource_link) and resource_visibility every time you want to show something on screen
> because of the logging of past actions, resource_visibility will get very big, very fast, and might slow down the requests on joined resource_node (or resource_link).

Ok agree (for now, until I remember why i did that way) to remove that resource_visibility and include visibility in resource_link.

Talking about logs, I think every table should have a table log using the audit doctrine extension, so the information is automatically saved.

#16 - 23/12/2015 16:45 - Yannick Warnier

Julio Montoya wrote:

> I was thinking about having the historical information (logs) about visibility in the resource_logs table. To avoid having a table just for visibilities.
> Let me put it another way:
> if you store visibility in the resource_node and resource_link table, then you only need one query to know if the item is currently visible or not. Historical information about who changed the visibility and when could be found in the resource_log table, which is the right place to keep logs.
> if you store the visibility in a table called resource_visibility, then:
> you will have to JOIN resource_node (or resource_link) and resource_visibility every time you want to show something on screen
> because of the logging of past actions, resource_visibility will get very big, very fast, and might slow down the requests on joined resource_node (or resource_link).

> Ok agree (for now, until I remember why i did that way) to remove that resource_visibility and include visibility in resource_link.

OK

> Talking about logs, I think every table should have a table log using the audit doctrine extension, so the information is automatically saved.

I agree.

#17 - 23/12/2015 16:49 - Julio Montoya

Ok I will recover some code from experimental to implement what is said here.

#18 - 28/11/2016 02:53 - Yannick Warnier

- Status changed from Needs more info to Assigned

- Assignee changed from Yannick Warnier to Julio Montoya

- SCRUM pts - complexity changed from ? to 40

This is probably the most urgent and complex database refactoring needed for 2.0.