Field Permissions (v1.21)

By Olaf Nöhring
https://datenbank-projekt.de
Version 2020-03-18 (1.21) fixed documentation (to clarify, thanks Chrisagon).
Version 2020-03-13 (1.2): Added hidden text to table view (if text is hidden, user can not see contents anymore in table view).
Changed behavior of detailview: Record is always saved (no more hint on suspected cheating) but hidden and locked values are read from the database and replace fields that user has seen. The _init function is heavily extended.
Changed behavior of detailview: FE check for locked fields: now only if user has insert or edit permissions on that table, because it's only needed in these cases
Version 2020-03-01 (1.15): Bug fix in field_permission_base.php Thanks for reporting aarlauskas.
Version 2020-01-31 11:00 (corrected PDF Step 3.4. thanks Paolo; added hint to preface; clarified step 2)
Version 2019-11-12 08:50

As AppGini (AG) in the current version 5.82 does not support field specific permissions, I decided to make a start.

The following describes, how to implement field specific permissions in your own AG application. You can easily LOCK and HIDE fields for specific usergroups. Single users are not supported at this time.
Please note:
a) LOCK and HIDE are applied in Detailview (where users normally are able to edit the data) using Javascript. This means, hidden elements can be seen if the user accesses the source of the page!
b) HIDE is applied in tableview completely. This means in table view the user will not be able to access contents of hidden fields at all (nothing to see, nothing in the source code either).

Installation time: approx. 30 minutes **(my guess, give me some feeback on that)**

This solution implements a frontend hiding and locking as well as a backend checking. If a users tries to cheat and fiddles around in the code of the page to change values he actually does not have access to (hidden/locked fields) the attempt to save will succeed, but hidden and locked fields will have the same value as before.
Keep in mind, that certain fields might be autoupdated by your AG application (editingTimestamp, editorUsername). Make sure these fields are not locked/hidden from a user, otherwise he will not be able to change a record!

Warning: No guarantees, that this extension is save to use! You use it at your own risk!

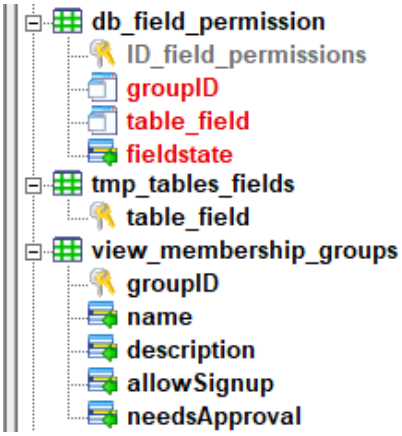
Important: This Field Permissions extension requires AppGini Helper! You need to buy AppGini Helper which can be obtained for small money from
https://www.bizzworxx.de/en/appgini-helper/
I decided to use Appgini Helper as it is a great script which supports all field types of AppGini and provides many new features to AppGini (AG Forum). Please take a look and support Jan Setzer (jsetzer) who is the programmer and great supporter of all users in the AppGini forum.

Updating

If you are updating, please replace all files that come with this and check if the edits needed for installation have changed. Apply those changes to your application.

Installation

The following steps describe what to do. Your AG application will be extended by two tables and a view:



Step 1

Place the following files in your /hooks folder:

```
field_permission_base.php

field_permission_functions.php

field_permission_tmp.php

field_permission_table_init.php
```

Step 2

Open the **each file that belongs to a table where you want to use field permissions** from the hooks folder in an editor and make the following adjustments in each table. Example: If you have a table named **customers**, you have to edit the file /hooks/customers.php. For the table **articles**, you will need to edit the file /hooks/articles.php

Step 2.1

Search for the ..._**init** function in that table.

Add this after the beginning of the function:

```
//Field Permissions Code
include("hooks/field_permission_table_init.php");
```

Side note to Step 2.1

If you want to change the characters that are displayed in hidden columns (of the table view), edit the file field_permission_table_init.php. Search and change this: `$hidden_text = '▒▒▒▒▒▒▒▒';` You can also change the prefix and postfix of the fieldnames for the filter by changing this `$unavailable_text_pre = '[-Unavailable: ';` and `$unavailable_text_post = '-]';` Simply change the things between the single quotes.

Step 2.2a

Search for the ... **_footer** function in that table.
Look for the `switch ($contentType) {` command.
Above this `switch` add the command to include the code that locks and hides your fields.

```
//Field Permissions Code
include("hooks/field_permission_base.php");
```

Step 2.2b

In the `switch` you will find several `case` commands. You will probably want to make adjustments in the *tableview*, *detailview* and *tableview+detailview*: Give the variable $extraJS_field_permission back in your footer like this:

```
$footer = $extraJS_field_permission;
```

or, if you already have something there, you might want to concat it:

```
$footer = $extraJS_field_permission . $NoRecordSelectors . '<%%FOOTER%%>';
```

Step 2.3a

Search for the ... **_before_update** function in that table.
Replace the `return TRUE;` with `return $myReturnValue;` at the end of the function:

Note: This new variable `$myReturnValue` will hold either the value TRUE or FALSE. If the variable is TRUE, the record can be saved (updated), if it's FALSE the record will not be saved (updated). If you use any other variable name already for a similar purpose, you must adjust the name `$myReturnValue` to your own variable name – this is even more true for the next step 2.3b!

Step 2.3b

Search for the ... **_before_update** function in that table (you might like to do this also in the _before_insert function, but be sure to check if your application still allows adding new records).

Note: The tested variable `$myReturnValue` in the followin if statement will hold either the value TRUE or FALSE. If the variable is TRUE, the record can be saved (updated), if it's FALSE the record will not be saved (updated).

If you use any other variable name already for a similar purpose, you must adjust the name `$myReturnValue` to your own variable name!

Add the following code **above** the `return $myReturnValue;` at the end of the function:

```
//Field-Permissions (Backend)
if ($myReturnValue === TRUE) {
      $myReturnValue = check_BE_field_permissions($data, $memberInfo, $_SESSI
ON['field_permission_tablenam'], $_SESSION['field_permission_tableID']);
}

return $myReturnValue;
```

Step 3

3.1 Create a VIEW in your database

Open your favorite MySQL tool (phpmyadmin, adminer) and run this SQL to create a view that is needed in the next step:

```
CREATE VIEW view_membership_groups AS SELECT * FROM membership_groups
```

Open your application in AG and create new tables (Step 3.2):

3.2 We need a “pseudo” table named view_membership_groups.

I am talking about pseudo because we have already created a view in the database with this name. Thus, when generating the application and running it the next time that table can not be created – “it” exists. But now we can use this pseudo table as source for a lookup field. So, go ahead and create such a table named **view_membership_groups** in AG.
Create the following fields in that table:

groupID (Integer, PK)

name (VarChar, Length 50, unique)

description (Rich (HTML area))

allowSignup (VarChar, Length 50)

needsApproval (VarChar, Length 50)

3.3 We need a temporary table which holds some values for later lookup (in the next table you will create):

Create a table named **tmp_tables_fields**
Add one field to that table:

table_field (VarChar, Length 200, PrimaryKey)

3.4 Now we need to create the table to define field permissions

Create a table **db_field_permission**
Add these fields:

ID_field_permissions (Integer, ReadOnly; PrimaryKey, AutoIncrement)

groupID (Integer). Set this to **Lookup Field** and choose as *Parent table* **view_membership_groups** and as *Parent caption field part 1* **name**.

table_field (VarChar, Length 200, Required). Set this to **Lookup Field**! Choose as *Parent table* **tmp_tables_fields** and as *Parent caption field part 1* **table_field**.

fieldstate (VarChar, 50, Required). Set this to **Option List** and enter this as *List values*: **locked;;hidden**

Step 4

Generate once, so that the files in the hooks-folder will be created.

Step 5

Adjust /hooks/db_field_permission.php (created by generating your code)

Step 5.1

Open the file and add **directly after the opening** with **<?php** this code:

```
//Field Permissions Code
if (!function_exists('fill_tmp_tables_fields')) {
  include("hooks/field_permission_tmp.php");
}
```

This way we make sure the list of tables and fields you can choose from in this table is always regenerated when this file is accessed.

Step 5.2

Look for the function ..._**init** and

before the line `return TRUE;` add this line of code:

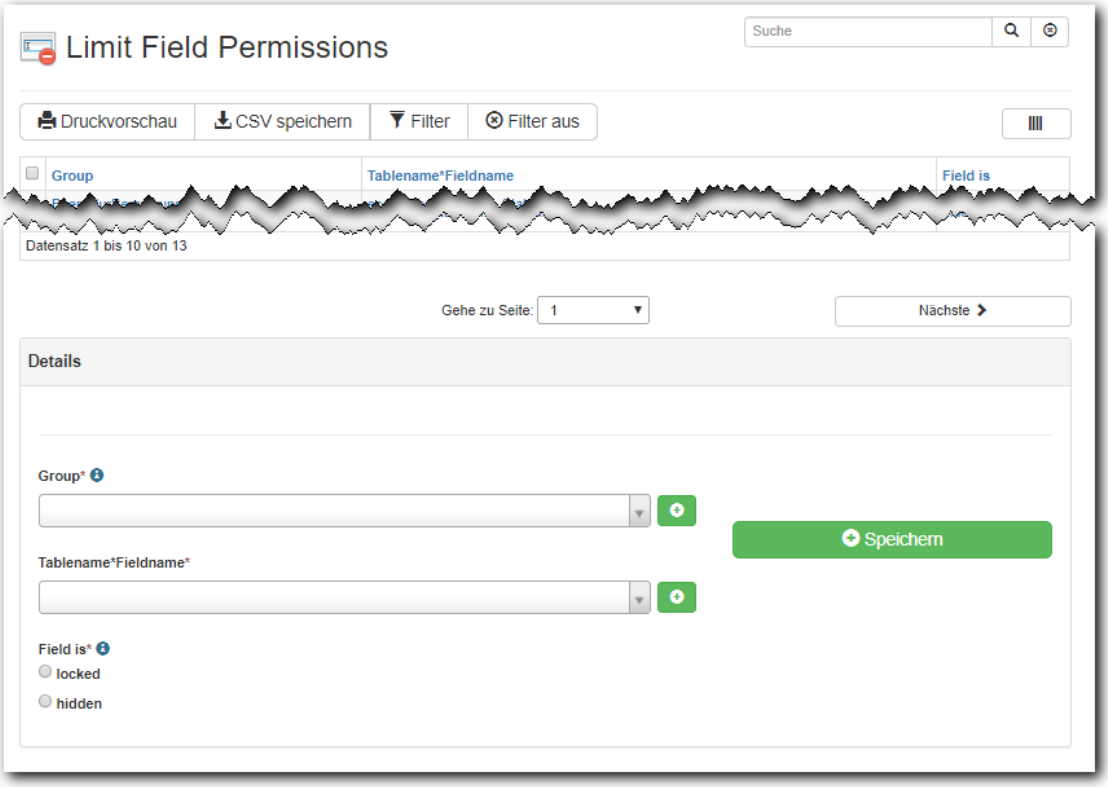
```
fill_tmp_tables_fields();
```

Extra Tip

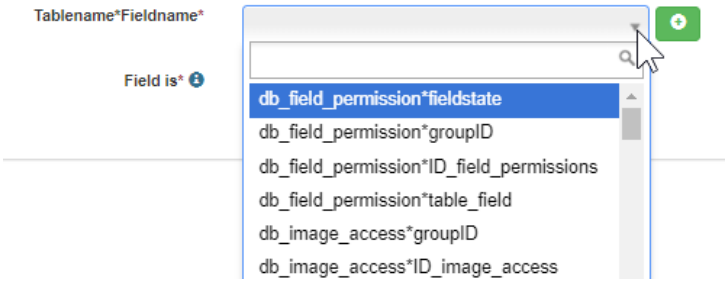
You might want to set some index in your table using phpmyadmin or adminer: It's nice to have UNIQUE KEY for the combination of *groupID* and *table_field*.

Ready Set Go

Now you should be ready to roll.
Now open your application and go to the db_field_permission table. It should look somehow like this:



The two lookup/dropdown fields give you access to all groups in your database and to all tables with all fields (format: tablename*fieldname).



You can simply choose the group whos access permissions you want to limit on a field basis. This assumes, that the groups actually have access to the tables where you limit field access to: If a group can not see a table you do not need to set anything. If a group can see, but not edit, you might want to set fields to hidden. If a group can not edit a table (AG admin settings) the group will not have this possibility at all – it does not matter what you set here.

PS: The tablename*fieldname lookup list will be recreated everytime you access this table.

In Action

At this time we need to make a difference between table view and detail view. At this time, you should assume users have the ability to see the value of a hidden column as well, as the data is only hidden from view, but not from the source code.

In Table View

Please note, that in table view HIDDEN data is actually not available to the user (not even in the source code).

Users can **not use the filter** function to filter for hidden columns and they can **not use the quick search** (top right are of your table view page) to search in hidden columns either. Using these methods and searching/filtering for something will return no results from the hidden columns.

If users use the CSV export, the hidden fields will not be revealed.

In the next image, the (contents of the) columns *Kunde*, *Bemerkung* and *Abteilung* are hidden from the user. In detailview, *Kunde* is a textfield, *Bemerkung* a textarea and *Abteilung* a lookup.
The table view example image might show a different setting than the one from details view example.

I have not tried the other types of fields (opion/checkboxes/images etc.)! Please send me a note if it works

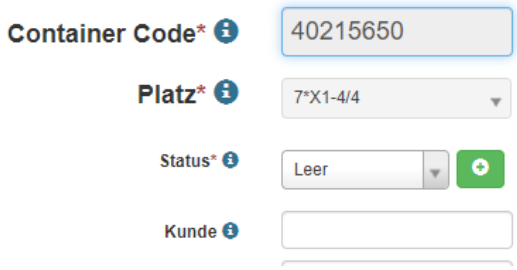
In Detail View

When the users opens a table (where his/her group has edit permissions to) the script will adjust the field display according to the settings made by you.

Note: All lookup fields will be locked once Ajax has finished.

It looks like this: *Container Code* is locked, so is *Platz*, *Status* and *Kunde* have no extra permissions set. HIDDEN fields are not shown – as they are hidden ;-)

The details view example image might show a different setting than the one from table view example.



If the users tries to cheat, the save edits command will work, but the values of the hidden and locked fields will stay unchanged. Technical info: Before saving the values of hidden and locked fields are read from the database and replace those coming from the user.

Additional information: SQL for the table(s) and view(s)

```
SET NAMES utf8;
SET time_zone = '+00:00';
SET foreign_key_checks = 0;
SET sql_mode = 'NO_AUTO_VALUE_ON_ZERO';

DROP TABLE IF EXISTS `db_field_permission`;
CREATE TABLE `db_field_permission` (
  `ID_field_permissions` int(11) NOT NULL AUTO_INCREMENT,
  `groupID` int(11) NOT NULL,
  `table_field` varchar(200) NOT NULL,
  `fieldstate` varchar(50) NOT NULL DEFAULT 'No',
  PRIMARY KEY (`ID_field_permissions`),
  UNIQUE KEY `komni_groupID_table_field` (`groupID`,`table_field`),
  KEY `groupID` (`groupID`),
  KEY `table_field` (`table_field`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;


DROP TABLE IF EXISTS `tmp_tables_fields`;
CREATE TABLE `tmp_tables_fields` (
  `table_field` varchar(200) NOT NULL,
  PRIMARY KEY (`table_field`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;


DROP VIEW IF EXISTS `view_membership_groups`;
CREATE TABLE `view_membership_groups` (`groupID` int(10) unsigned, `name`
varchar(100), `description` text, `allowSignup` tinyint(4), `needsApproval`
tinyint(4));


DROP TABLE IF EXISTS `view_membership_groups`;
CREATE ALGORITHM=UNDEFINED SQL SECURITY DEFINER VIEW `view_membership_groups` AS
select `membership_groups`.`groupID` AS `groupID`,`membership_groups`.`name` AS
`name`,`membership_groups`.`description` AS
`description`,`membership_groups`.`allowSignup` AS
`allowSignup`,`membership_groups`.`needsApproval` AS `needsApproval` from
`membership_groups`;
```