2Org-Cows Software Documentation

Thomas Rahimi: thomas.rahimi@mailbox.org,
Boris Kulig: bkulig@uni-kassel.de

Universität Kassel,
Fachbereich 11,
ökologische Agrarwissenschaften Witzenhausen,
FG Agrartechnik

Contents

List of Figures

1 Introduction

The 2Org-Cows project is multinational and multi-partner project of research institutions in organic agriculture in Europe. It aims to improve the breeding of dairy cattle for organic dairy cow keeping and therefore aims to collect data to measure different parameters of animal welfare, health and keeping.
In order to analyze all these data, a huge database has been created, which consists of the following components:

- a webbased, graphical user interface for the final user to access and query the database using a standard webbrowser
- a variety of interfaces to other programs
- software to analyze database stored datasets using big data methods, mainly based on R and on Python3
- MySQL databases as the storage backbone of the whole software and project
- Ubuntu 16.04 operating system, PHP7 scripting language and Apache2 webserver to provide the user with the data required

In the following all parts of the software for the 2Org-Cows project should be documented. Please note that this file is work in progress and will therefore change steadily.
To allow easier reading, the following modifications to the text are of importance:

- Source code and configuration details are always printed in boxes such as this:

```
1  while true
```

- *File paths are printed as emphasized text*
- *Links are written in italics*
- `Function names and other parts from the code are written in monospace`

2 Common information regarding the coding

The project is generally based, in its programming, on several concepts, which should be briefly displayed in the following. The programming is therefore differenciated in the parts regarded.

Web Frontend The web frontend is largely written in PHP7 and HTML5, following the following paradigms:

- clear structure of code
- clear separation of display code and code for the processing of input
- easy to write, understand and maintain code
- Mostly procedural code style, only exception is the database system, which is based on object oriented code style
- All available standard security solutions for PHP and HTML code should be applied, to ensure secure operation of the database. This includes escaping of all strings, using `mysqli::real_escape_string()` for all values, which is executed for all strings being in some kind of contact to the database, and `htmlspecialchars()` for all other strings. The insert scripts perform type checking operations on the content to insert, to limit the risk of inconsistent data.
- SQL queries, including user generated content, are usually proceeded as prepared statements, to limit the risk of sql injections.

The appearance of all sites is specified in the *2Org-Cows.css*, which acts as the only stylesheet in the whole user interface. Only in some limited situations, style information is provided in HTML5 inline commands. These cases mainly occur in table-styled content, where the font definition is separetely on each sheet. In the first development stage only limited responsiveness of the design is achieved, further development will focus on the responsiveness of the design and the right alignment of user interface elements in HTML5 on small screen devices, such as smartphones. Due to the extensive size of all pages, this might lead to further reconstruction of the page and therefore to a special mobile version of the whole frontend.

Database The database design aims to provide a star schemed database, with as little programming and connection efforts as possible. Therefore, the overall database largely consists of independent tables, which are linked together by the code of the frontend and the database interfaces.

3 The webbased User Interface

The user interface for the database has been developed as a graphical interface to allow broad acces to the database. In the following, the different pages and their corresponding scripts, in the web frontend, are described in detail.

3.1 Session Management

Throughout the user's way through the web frontend, three different sessions are started and terminated, some of them containing crucial data to the processes in the web frontend. The first session started just checks, whether the user allows cookies in his browser or not. This session does not contain any data apart from the `session_id()`.
To log the user in, another session is started, which basically contains the server side csrf protection tokens for the login form. This session is closed and destroyed after evaluation of the submitted tokens, by the login_script.php script. The establishing of this session is crucial to the login process of the user, as otherwise the login_script.php discarges the login, as a csrf attack is suspected, if the credentials do not match.
During the login, the final session is started, keeps track of the users action and allows him to access the core database. Unlike the two sessions before, this session contains multiple session variables for many different purposes, as pointed out in the section to the *login_script.php*. If the user logs out from the session, the session will be destroyed with the call `session_destroy()` . In case the user quits without logging out, or the session is inactive for a longer time, sessions are terminated automatically after 24 hours of inactivity. This timespan is set under the following point in the */etc/php/7.0/apache2/php.ini*:

```
1 session.gc_maxlifetime = 1440
```

As all sessions are administrated in the individual `session_set_save_handler()`, the major difference between the different sessions is their name, which is set with `session_name()`. To allow the individual session_handler to manage sessions and to receive the incoming session calls from the different scripts, the */etc/php/7.0/apache2/php.ini* file has to be altered to match the following:

```
1 session.save_handler = user
```

For the sessions to work under this conditions, it is important to include the file for the session_handler file before `session_name()` and `session_start()` are called.

3.1.1 session-handler.php

The session-handler.php file is based on a user comment on the *php.net page*, which is altered to deal with the given *sessions table* and to match the programming style for database

queries. The session-handler.php file contains a class definition, which implements the `session_set_save_handler()` for this individual project. Within the class, the following functions for session handling are defined:

- open → processes the opening of a session, by creating an entry into the database, where the `session_id()` is stored
- close → closes the database connection to finish the session
- read → reads data, which are stored in the 'data' column of the sessions database and gives them back as $_SESSION variables, is called whenever $_SESSION variables are read
- write → writes new $_SESSION variables into the database, is called whenever $_SESSION variables are set or altered
- destroy → deletes the database entry corresponding to the individual session, is called if the user logs out and `session_destroy()` is called in the script
- gc [garbage collection] → removes old sessions, which have not been used for the timespan $maxlifetime

Throughout the functions, several different parameters are expected by the functions, to work properly. These parameters are:

- $savepath → contains the general path to store session data, which is set in the */etc/php/7.0/apache2/php.ini*
- $sessionName → contains the session name, set in `session_name()`, no further script interaction necessary to deliver it
- $id → contains the `session_id()`, no further script interaction necessary to deliver it
- $data → contains all $_SESSION variables, is set automatically from all existing variables of a session
- $maxlifetime → contains maximal lifetime of a session, which is set in */etc/php/7.0/apache2/php.ini*, see above

Throughout all functions, the database connection is initiated again, as this is the simplest way to deliver the connection token to all functions. For security reasons and to keep the code easier to overview, the sql queries are written as prepared statements in an object-oriented manner.
At the end of the file, the newly created class is embedded into the `session_set_save_handler()`, to be included in all files, which use it. The whole file is visible here:

```php
<?php
class SysSession implements SessionHandlerInterface
{

    private $link;

    public function open($savepath, $sessionName) {
            $link = new mysqli();
            $sql1 = "INSERT INTO sessions (`id`, `last_activity
               `) VALUES (?, NOW())";
            $stmt1 = $link->prepare($sql1);
            $stmt1->bind_param('s', $sessionName);
            $stmt1->execute();
            return true;
    }

    public function close() {
            $link = new mysqli();
            $link->close();
            return true;
    }

    public function read($id) {
            $link = new mysqli();
            $expire = date("U") - (60*69*24);
            $sql2 = "SELECT `data` FROM sessions WHERE `id` = ?
                AND `last_activity` > ?";
            $stmt2 = $link->prepare($sql2);
            $stmt2->bind_param('si', $id, $expire);
            $stmt2->execute();
            $result2 = $stmt2->get_result();
            if ($row = $result2->fetch_assoc()) {
                return $row['data'];
            } else {
                return " ";
            }
        }

```

```
37          public function write($id, $data) {
38              $link = new mysqli();
39              $date = date("U");
40              $newdate = $date + (60*60*24);
41              $sql3 = "REPLACE INTO sessions SET `id` = ?, `
                    last_activity` = ?, `data` = ?";
42              $stmt3 = $link->prepare($sql3);
43              $stmt3->bind_param('sis', $id, $newdate, $data);
44              $stmt3->execute();
45              return true;
46          }
47
48          public function destroy($id) {
49              $link = new mysqli();
50              $sql4 = "DELETE FROM sessions WHERE `id` = ?";
51              $stmt4 = $link->prepare($sql4);
52              $stmt4->bind_param('s', $id);
53              $stmt4->execute();
54              return true;
55          }
56
57          public function gc($maxlifetime) {
58              $expired = date("U") - $maxlifetime;
59              $link = new mysqli();
60              $sql5 = "DELETE FROM sessions WHERE `last_activity`
                    < ?";
61              $stmt5 = $link->prepare($sql5);
62              $stmt5->bind_param('i', $expired);
63              $stmt5->execute();
64              return true;
65          }
66  }
67
68  # https://secure.php.net/manual/de/function.session-set-save-
        handler.php#118225
69  $handler = new SysSession();
70  session_set_save_handler($handler, true);
71  ?>
```

3.2 login.php

The login.php site serves as the welcome site of the whole project, including basic information on the function of the sites for user access to the database.
For security reasons and to track the visitors of the page, basic information about the visitor is stored in a separate database. This includes IP addresses, user agent of the browser and the time of access. To comply with the lawful requirement not to store personal information, such as full IP addresses, the IP address is shortened to three blocks, which does not allow single user identification for non-logged-in users.
The page also starts a short test session, to check, whether cookies are are allowed in the browser. The result is not completely true, as it technically requires a reload of the page, which is not forced by the script. The login works with HTML5 forms, organized in a table to allow easier configuration to various screen sizes and offer the entering of username and password. These forms consists of a text input with the input widget for text, to enter the username and a password input with the password widget. To prevent CSRF-attacks on the visitors of the page, a random sequence is generated by the server and linked to the page. As there is just one form to be processed on this page, the token in generated with the call `bin2hex(random_bytes(32))`. The sequence is stored as both, the session variable and in an hidden input field, which leads to the a $_POST variable. This sequence is requested in the processing of the login during login_script.php to check for CSRF-attacks and to issue an alert if the sequences do not match. Furthermore there is a submit button, which is submits the form data. Submitting this form starts processing of the form data with login_script.php .

3.3 login_script.php

The login_script.php script processes the data submitted from the login form. For security reasons, authentication makes use of two different databases. The authentication database does not include any other data than the data required to initiate a session. In contrast, the agri_star_001 database includes more details on the specific user, including the insititution and the role of the user.
To ensure the integrity of the user request for authentication the form on login.php includes a random sequence, which is both stored, in the session and the POST submission. The comparison of the both results should ensure better protection from csrf-attacks.
Data processing happen in 4 steps, which come up sequencially.

1. At first the script checks, whether a username exists, which matches the username provided in the form. In case the username can not be found in the database “auth”, the script redirects the user to login.php and creates error message, indicating unknown username. Otherwise the script continues to step 2.

2. In case, a user has tried, a login with non matching credentials, too many times, the user

is redirected to the login.php page and told to wait for some time. This settings should slow down brute force attacks on available accounts.

3. If the username has been found, the script checks, whether the password, provided by the user, matches the password stored in the “auth“ database. Therefore the PHP function `password_verify()` is used. If the password provided does not match to the hash value, stored in the database, the user is redirected to login.php, receiving an error message about the password. Otherwise the script starts to initiate a session.

4. In the next step the script starts to create a session for the use of the research database. In order to ensure the security of the session, several steps are taken:

   - the IP address of the user is stored as a session variable to prevent session hijacking
   - the user agent of the user’s browser is also stored as a session variable to prevent session hijacking
   - the user’s institution is stored to regulate access to the data stored in the research database, this information is encoded in the group id, which is set as a session variable, instead of the group name and the department
   - the username is stored into the session to allow recognition of the user
   - the user-id is stored into the session, because it is the primary key to the users in the different databases
   - the session-id is stored into the session aswell, to allow the recognition of the session in the further session administration system

   Afterwards, the user is redirected to home.php, which acts as the main page. In order to check on the usage of the website, all successful logins are logged into a separate table in the logging database. To minimize the tracking of single users according to their IP address, the IP address is shortened to three blocks, which still allows sufficient data for the analysis of user interaction with the database.

5. In case no login information has been provided by the user, the user is redirected to login.php .

In case the login failed, the attempt is written into the Login_Failed table in the logdb database, to track bruteforce attacks on user accounts.
The database connections are closed at the end of each script to prevent accumulation of unused database connection by various scripts.

3.4 check-session.php and check-session_restricted.php

The both scripts check-session.php and check-session_restricted.php act as the connecting scripts between the different pages within the project. The basic intention of the scripts is to check, whether a session for the user exists and whether the user, therefore, is allowed to access pages within the project. Furthermore, the check-session_restricted.php script checks, whether the logged in user has sufficient permissions, to access limited pages.
The session check conditions are the same for both scripts, in both cases they check for existence and matching of parameters, set in the login_script.php. The major aim to check for is the `session_id()`, which is set as a session variable in the initialization of the session. By checking the session variable of the `session_id()` against the actual `session_id()`, session hijackings might be detected and it is ensured that the session is sticked to one single session cookie. In case the client's IP address is set, the script also checks for the matching of this session variable with the current client's IP address, which should also prohibit session hijacking. As the browser agent of the regarding browser should be stable throughout the session, it is also set as a security measure to detect session hijacking.
In case any of the mentioned security tests fails, the user is referred to the login.php page, receiving a message that the session was aborted due to security reasons. The same happens, if the session timed out due to long inactivity of the user. In all other cases, the current session is kept alive and the user refered to the destined page.
For the case a user tries to access a page, to which he does not have sufficient permissions, the user is refered to the home.php page, receiving a message, indicating his insufficient rights.

3.5 home.php

The home.php page serves as the central page of the whole project, providing basic information on the use of the frontend and showing all navigation entries to refer to other pages within the frontend. To ensure that non permissioned users can not access pages such as create-user.php, these entries are just shown to users with roles higher than 1 or 2. The same happens for the dropdown menu, which is included in all pages, to allow easy navigation.

3.6 traffic.php

The traffic page is the evaluation page to the logdb database, which keeps track of users accessing the login.php page and successful logins. As these data are highly sensitive and not important for the daily use of the database, the access to this page is limited to users with the role admin (role code 4). The presented data are split into different tables, one for the login.php page, displaying amount of accesses, time and IP address. Within that table, there is one line for all accesses, ordered by IP address corresponding to that access. Another table displays the data for the logged in users, showing IP address, time of login and user invoked. And finally

there is a table displaying all login attempts, which failed due to incorrect credentials. This table should enable the recognition of brute force attacks on given accounts.
The different table lines rely on different SQL queries, to gain there results, even if they request the same table.

3.7 database-update.php

The database-update.php provides the graphical interface to import external data into the database. Therefore, it is the only way, the users of the interface can modify the database on a larger scale, than in the manual inputs, which are provided in the other pages. For the upload the user has to choose the appropriate requirements for the file, which include the file type and the measurement procedure applied during the data collection of the data in the file. The processing of the files takes place in the following upload.php file. Similar to the create-user.php page, the database-update.php page can just be accessed by users with higher roles.

3.8 upload.php

The upload.php script processes the files selected in the database-update.php file and pastes them into the files database table in the agri_star_001 database. Before the actual upload, the user is requested to select the type of the file, which should be uploaded to the database. According to the user choice, the entries are set in the file database and the file is processed in such a way. If there are errors during entering the file type, the file might not be processed correctly and be dropped during the reading process.

3.9 create-measurement.php

In order to map results of trials to secific measurements, measurement procedures are stored in a specific database, which is then linked to the results.

3.10 create-user.php

The create-user.php site offers the opportunity to create a new user and to set the right for this specific user. User creation follows the idea of dropping rights, which only allows a user to create another user with less rights than he has himself. To ensure this, create-user.php is just visible to users with a role, higher than student. This includes the following roles: admin [role code: 4], professor [role code: 3], scientific coworker [role code: 2].
Furthermore only professors and admins are allowed to create users at a different institution, scientific coworker are only allowed to create users at the same institution as they are. In the same way, the erasing of users is limited to professors and admin users.
Creating a user leads to modifications in the Dim_User table and the authentication database.

This database updates are proceeded, if the user clicks on the “create user” button, which calls the “create_user_script.php” file to process the form data and insert them into the database.
The creation of new institution is also limited to people logged in as professors and admin. Each department is created separetely, even if an institution owns several departments, taking part in the 2Org-Cows project. This improves the adjustability of rights sticked to the specific department in case of splitted user rights on certain datasets. The data inserted for the creation of an institution are processed by the create_institution_script.php file.
Furthermore, the create-user.php allows the deletion of existing users. This is limited to users with the role code 4 and 3 (admin and professor), therefore the form is just visible to users of that role. To prevent accidental deletions, a separate checkbox has to be checked, to delete a user. User deletion is limited to users from the same institution and exclude the current user. The deletion of the user is proceeded by the delete_user_script.php .
To allow users, who forgot their password, to regain access to their accounts, users with higher roles can set the password for users in their institution. As there is, up till now, no further security in the password updates, these actions are limited to high profile users. In case it might be necessary, further protection mechanisms to the password updates, such as requesting a secret, etc. could be implemented. The password updates are processed by the set-password.php script.
As the data in the database are bound to specific groups, the rights management on the group level has to be performed with the administration aswell. Therefore, all departments are retrieved from the database and listed in a table. To improve the overview to the departments, the table is splitted into institutions and departments. Finally, the table includes one checkbox per department, which sets a value in the corresponding table. The table also shows with the checkboxes, which departments are granted access rights to the data tables of the institution invoked. As all permissions are stored in a translation table in the database, an own script updates this table and sets the value for the access on the data.
As there are multiple forms on one page, the csrf protection tokens are applied as split calculation tokens. In this case, the $_SESSION token just contains an random number, generated with `uniqid()`. For the hidden input field, this number is part of the input for an SHA256 hash calculation with `hash_hmac()`, where a string and the number from the $_SESSION variable are the input. The same operation takes place in all the scripts, the output is then compared to the submitted $_POST variable, using `strcmp()`.

3.11 create_user_script.php

The create_user_script.php is the processing file for the entries to create a new user in the web interface for the database. Basically the script just performs the validity checks of the form data provided in create-user.php and an **INSERT** query to the MySQL database.
At the beginning the script checks whether the provided check bytes submitted in create-

user.php match those from the session. Again this is a CSRF-protection in case the session has been hijacked. If there are no problems on the CSRF protection, the script checks for the validity of the form provided. This part already takes place in the html, where the tag “required” is set for all fields, but a controll in the script should avoid empty fields in the database.
For the validity checks the following steps are taken:

- validity check of the e-mail adress provided
- check whether the desired username is available
- check, whether the password matches matches the control field
- check whether the user permissions, entered in the form, are allowed to be performed by the user invoked
- check whether the user invoked is allowed to select an institution different from his own
- check whether the entered permissions are met by the other data entered into the database, such as check, if a user role is possible or if a group exists.

All manual input on free text fields, such as the username, etc. is not checked again by the script and taken for true and granted. During the script, only one value is changed automatically, which is the username, as it is set to a small letter string.
If any of these checks fails, the user receives an error notice and is directed to the form.As the group, department and the country entries are stored in a different way in the database, than in the user session, the database values have to be retrieved from the database. Therefore, the corresponding institution, department and country are queried from the Dim_Group table. As the country is stored as an integer value, the corresponding ID_Country has to be queried from the Dim_Country in the next step.
The passwords are stored as bcrypt hash values in a separate database, the hashing is performed after the validity checks for the form. The next step is to insert all data into the Dim_User table in the agri_star_001 database. During this step the ID_User value, which identifies the user during further operation, is set as the resulting auto-increment value for the primary key from the database. To allow sufficiently large user amounts, the ID_User field is of the type BIGINT. The insert is proceeded in an objective style, generating the query first, then assigning the values to the query (mysqli::prepare($query), mysqli::bind_param(parameters)) and than executing the query (mysqli::execute). The transaction id of that insert is retrieved, as it is used as ID_User, to map the data in the auth database to the corresponding user. In the next step the username, the password hash and the ID_User are inserted into the auth database, using the same process as for the Dim_User table.
The scripts quits with a success message.

3.12 create_institution_script.php

The create_institution_script.php is the processing script for the entries in the institution form. Again it is basically an **INSERT** query to the database, where no external have to be retrieved from other tables in the database. The only type checking for an input field, being performed by the create_institution_script.php, is the check of the provided e-mail address. This check just checks for the basic match with e-mail regulations. Furthermore, a logical check is performed on the selected country, where the script checks, whether the selected country exists in the database. In all other cases, the user input is not checked again by the script and taken for granted and true. If all fields are filled, the provided data is put into the Dim_Group table in the database, using a similar method as for the users.
To allow groupwise right grants, a separate table with all institutions in the Dim_Group keeps track of the granted rights. Therefore, it is important to create connections between all groups created during the project. In order to do this, the transaction ID is retrieved from the database and all other ID_Group values are queried from the database. Than these values are written into the grants table, where one column is the giving group and another column is the receiving group.
Finally the script quits with a success message.

3.13 delete_user_script.php

The deletion of a user does not lead to all his data being wiped out of the Dim_User and the auth table, instead the user is simply kept from successfully logging into the frontend. Therefore, the password in the auth table is updated to a cryptographically save binary string. Furthermore, the column user_not_active in the Dim_User table is set to 1, which does not show the user in any active state anymore. The script exits with a success message and refers to the create-user.php page.

3.14 set-password.php

In case a user has forgotten his password, these can be resetted by a user with a role higher than 2 (professor and admin). As there is no further checking, whether the user taking the action is allowed to set the password, this is type of a trust system, which should exist in the institution taking part in the project. The set-password.php script does the following steps:

- check if the two password fields match
- hashing the password, using the bcrypt standard
- saving the password hash to the auth database

If all entries are fine, the script quits successfully and refers to the create-user.php page again.

3.15 grant-rights.php

The permissions administration in the rights table is written with the grant-rights.php into the database. Unlike the former scripts, which basically performed an INSERT query on the database, the grant-rights.php first checks, whether an entry corresponding to the two mentioned groups exists in the grants table. This just occurs, if the user has chosen a group, with which he wants to share data. Otherwise, nothing happens to the table and not selected groups are neither inserted nor updated. In case the entry with the two groups does not exist both ID_Group values and the value for data access (1) are written into the database.
If the matching of both groups already exists, as it was created during a group creation, the script just updates the access value to one.
To withdraw rights from existing groups, the corresponding checkboxes on the admin page have to be unchecked. As the content of the checkboxes is transmitted as an array, containing all group numbers delivered by checking the checkboxes, the grant-rights.php script can check for the existance of all shown groups in the array. If an ID_Group value does not come up in the array, the connected checkbox has either not been checked or has been unchecked.
As the first step, the script gathers all groups, to which the current user's group has granted rights, from the database, where entries to these groups must exist, as they have been at least created during the rights permission process. In the loop to create the array from the database results, the script performs a check, if the currently processed group comes up in the checkbox array from the script. If the ID_Group value does not come up the checkbox array, an SQL query is executed to replace the value for data access (1) with a 0. Otherwise, the loop ends, when the array has been completely parsed.
Then the scripts exits with a success message, but does not indicate, which results have been achieved. This can easily be done by checking the checkboxes, which show the current right status for other groups.

4 Interfaces

The interfaces to the database are largely written in Python3 and implemented to act independent from the user interface. This separation should minimize ressource conflicts between the user interface and the database interfaces, if large files are processed.

5 Database

The database for the projects are API-compatible, SQL databases, which run on freely available linux systems. Eventually the used and tested MySQL or MariaDB databases could be replaced by other systems, such as MSSQL. Nevertheless, a full compatibility of other databases with the source code provided can not be guaranteed. To ensure safe operation of the server, databases, which are not directly linked to the project data, are separated from the main database. As a result, the login database and the external logging database are separated from the main database. Therefore, three different databases exist for the purpose of the project.
To increase the security of the database and to limit the risk of damage in case of SQL-injections, all databases are accessed by non-privileged users. Furthermore, there is a separation in the users, regarding the function of the database invoked, to limit access to sensitive data in the database. As the project does not require external programm access to the database, the network access to the database is limited to localhost.

5.1 agri_star_001

The agri_star_001 database is the main productive database in the whole project, as it serves as the major saving space for data gathered in the project. In the project's initial shape, the agri_star_001 should have been the only database in the project, but during the development of the web frontend, further databases have been added for several reasons.
As already indicated in the introduction, the database uses a star schemed structure, to allow quick access to all data in the different tables. The tables are the following:

- **Dim_Animal** → contains information related to an individual animal
- **Dim_Country** → contains the ISO abbreviations for the countries, where the contributing institutions come from
- **Dim_Farm** → contains information on the farm level
- **Dim_Gage** → contains information on the applied measuring methods in the project
- **Dim_Group** → group related information, such as the address, the responsible person and the country. As already mentioned in the create_institution_script.php, the group

table includes the information for each department individually. Therefore, an institution with multiple departments, taking part in the project, has multiple entries in the Dim_Group table.

- **Dim_Time** →
- **Dim_Timezone** → table to keep the timezones for the different partners in the project
- **Dim_Trait** →
- **Dim_User** → user related information, such as related institution, etc.
- **Search** → Important to save the searches of users, to allow multiple search requests with the same settings
- **dates** →
- **files** → table to save files for the import into the database. Mainly used by the interface scripts of the web frontend and the database scripts
- **grants** → Translation table to implement an access control list (ACL) system. This table stores all possible connections between groups, to allow foreign access on data
- **numbers** →
- **numbers_kl** →
- **numbers_small** →

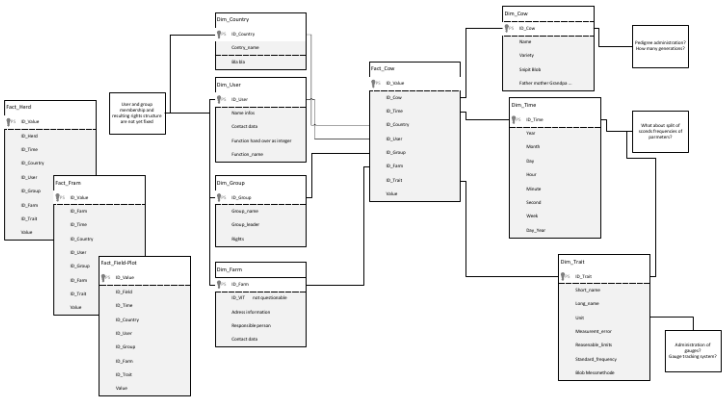

Figure 1: Original Structure of the agri_star_001 Database

For data access during operation, the sql requests might be combined, if the required data are spread on different tables. This might lead to JOIN requests, combining different sql queries in one go. Further connection of the tables happens over the interface software, which usually treats the tables as single units, but can also connect them.

In many tables, the primary key is set automatically as auto increment, to ensure that all datasets have unique keys. Manual setting of the keys might under certain circumstances lead to multiple data sets, having the same primary key, as the random function to generate the key number might output the same number under different circumstances.

5.2 user_auth

The user_auth database is an auxiliary database to the agri_star_001 database. It is required to allow logins to the web frontend, as it saves the authentication information for the users. Unlike agri_star_001 database, user_auth just contains one table (auth), which includes all required information, basically the username, the password hash and the ID_User. For security reasons, these information have been separated from the Dim_User table in agri_star_001. The ID_User in the user_auth database matches the ID_User value in the Dim_User table, as it is taken over from the transaction id of the user creation in the Dim_User table. During login process, the first requests are all the user_auth database, to prevent external attacks on the main database. Furthermore, the user_auth database has an own dedicated user, which does not have any rights on other databases, to access the database.

5.3 logdb

For security reasons, the web frontend includes some logging, which writes the results into a designated database. This database consists of three tables, which are not depending on any other frontend solution, except for the traffic.php page. If a browser hits the login.php page, the access IP, the user agent of the browser invoked and the time of access are logged. To comply with the requirement not to store unnecessary individual data, the IP address is shortened to three blocks.
In case the user logs in, the same data, including the ID_User are stored, whereas failed login attempts are logged with time, username, user agent and IP address. The use of this data is just for internal issues, such as preventing brute force attacks. It is not part of the project, to collect user data.
To avoid the aggregation of old logdata into the database, an event is created to ensure the frequent deletion of old log entries on the access database. Therefore, the following SQL command is used:

```
1 CREATE EVENT logrotate ON SCHEDULE EVERY 1 WEEK DO DELETE FROM
    Log_Table WHERE time < NOW() - 60*60*24*7;
```

5.4 sessions

In order to allow scalability of the whole user interface and to improve the quality of the sessions, session data are stored in a separate database, which is called sessions. The database has only one table, which is called sessions aswell and consists of three columns: **ID**, **last_activity** and **data**. The first column contains the `session_id()`, the second one a timestamp with the most recent activity in the session and the last column contains all session data for the individual session. All entries are administrated by the *session handler* for php.

6 Operating System

6.1 Ubuntu

In Ubuntu less modifications than in CentOS are necessary to allow a stable and safe operation of the server. As the operating system ships another mandatory access controll system (MAC), configuration of user rights is far easier and faster proceded. During the installation of the operating system, the LAMP stack and the openssh server have been chosen for automatic installation. After the installation and the required reboot, updates should be checked and installed.

6.1.1 Securing the server

To improve the server security, just logged in users can make use of the server. To prohibit external login attempts using ssh, the server is secured in different manners, of which one is the setup of groups, allowed to login using ssh. Therefore, a specified group is created, to which all login users are added, requiring the following commands:

```
1 sudo addgroup --system [groupname]
2 sudo adduser [username] [groupname]
```

Than the */etc/ssh/sshd_config* file is modified, including adding and modifying the following lines in the file:

```
1 AllowGroups [groupname]
2 PermitRootLogin no
```

Finally the ssh server has to be restarted.
Another security mechanism is the definition of available services in the */etc/hosts.allow* and */etc/hosts.deny* files on the server. This allows to block access all other webservices, except for the required ones from other IP-addresses. This feature has been implemented in parts.
Furthermore, the server makes use of a firewall, to prohibit attacks on not required open ports. Therefore, the ufw interface to iptables has been chosen. As the server is not required to listen on other ports than the standard ports for the required services, the general rule is to deny all incoming traffic. At the same time, traffic is generally allowed with the following commands:

```
1 sudo ufw default deny incoming
2 sudo ufw default allow outgoing
```

In the next step, all required network services, are excluded from the general denial of the services, using the following commands:

```
1 sudo ufw allow ssh
```

```
sudo ufw allow http
sudo ufw allow https
```

After the configuration is finished, the firewall is set active with the following command:

```
sudo ufw enable
```

6.1.2 Database installation and configuration

As Ubuntu ships MySQL as its default database, MySQL is used in this installation aswell. The database comes with a basic configuration, which should be checked after installation to match the conditions, under which the database server is operating.
If necessary the configuration file under */etc/mysql/conf.d/server.conf* has to be altered. It is important to check, if the networking access to the database server is allowed and if, which clients are gained access to the databases. For security and compatibility the setting listen: 127.0.0.1 has been chosen here, which allows the network interface of PHP7 to access the database locally, but hinders external clients from initiating a network connection to the database.

6.1.3 Web server installation and configuration

Both, apache2 and PHP7 have been installed in the initial installation, together with the database. As the web server should be configured as a virtual host, it is important to modify the settings here. There have been two virtual hosts defined, both referring to the same content folder, one host serves as the non-tls demo installation, whereas the other host is set up with tls support to deliver the results of the requests as encrypted material. The non-tls host is not intended to be used for login, as all user data sent to this host are in danger of being intercepted. The configuration proceeds in the following steps:

1. First the directories for the virtualhost are created, using the following command:

```
sudo mkdir -p /var/www/path_to_directory/content
```

2. Next the permissions on the folder are set to the user invoked or a specific user for the deployment of the web server:

```
sudo chown -R $USER:$USER /var/www/path_to_directory/
    content
```

3. As the next step, the configuration files for the virtual host are created. To minimize spelling errors, the files are generated from the default configuration, which is copied into a new file:

```
sudo cp /etc/apache2/sites-available/000-default.conf /
    etc/apache2/sites-available/name_virtual_host.conf
```

4. Now the virtual host has to be configured, with the following settings:

```
<VirtualHost *:443>
  SSLEngine On
  SSLCertificateFile /etc/ssl/certs/name_of_cert.crt
  SSLCertificateKeyFile /etc/ssl/private/key.key
  SSLCACertificateFile /etc/ssl/certs/certificate.crt #
     can be missing, depending on tls configuration of
     the server
  ServerName www.servername.com #is missing in case of
     the server, to allow host resolution to the default
     virtual host and avoid DNS conflicts
 ServerAlias servername.com #is missing in case of the
    server, to allow host resolution to the default
    virtual host and avoid DNS conflicts
  ServerAdmin admin@e-mail.com
  DirectoryIndex login.php
  ErrorDocument 404 /404 #should avoid 404 page not found
      errors
  DocumentRoot /var/www/path_to_directory/content
  ErrorLog /var/www/path_to_directory/error.log
  CustomLog /var/www/path_to_directory/access.log
     combined
  Alias "/admin" /var/www/path_to_directory/content/
     create-user.php
  Alias "/measurement" /var/www/path_to_directory/content
     /create-measurement.php
  Alias "/search" /var/www/path_to_directory/content/
     search.php
  Alias "/results" /var/www/path_to_directory/content/
     search-results.php
  Alias "/license" /var/www/path_to_directory/content/
     license.php
  Alias "/upload" /var/www/path_to_directory/content/
     database-update.php
  Alias "/cow" /var/www/path_to_directory/content/cow.php
```

```
21      Alias "/logout" /var/www/path_to_directory/content/
            scripts/logout.php
22      Alias "/home" /var/www/path_to_directory/content/home.
            php
23      Alias "/user" /var/www/path_to_directory/content/user-
            properties.php
24      Alias "/404" /var/www/path_to_directory/content/error.
            php # should avoid 404 page not found errors to the
            error.php page
25  </VirtualHost>
```

The "Alias" settings maskerade the files, visible to the user with a shorter, easier URL. As the php code mostly refers to the shortened Alias URLs, these entries are essential for the server to work. The **ServerName** and **ServerAlias** might be missing, to comply with local DNS settings.

5. To ensure that users do not use the unencrypted site by coincidence, the following redirect for the non tls-host is created:

```
1   <VirtualHost *:80>
2   ServerName www.servername.com #is missing in case of the
        server, to allow host resolution to the default
        virtual host and avoid DNS conflicts
3   ServerAlias servername.com #is missing in case of the
        server, to allow host resolution to the default
        virtual host and avoid DNS conflicts
4     ServerAdmin admin@e-mail.com
5     Redirect permanent / https://www.servername.com #
          Redirecting from non-tls host to tls-host
6     ErrorLog /var/www/path_to_directory/error.log
7     CustomLog /var/www/path_to_directory/access.log
          combined
```

6. In the next step, all files named in the virtual host configuration, must be created or copied to the server. To create the files, the following command is used:

```
1    sudo touch /var/www/path_to_directory/error.log
2    sudo touch /var/www/path_to_directory/access.log
```

7. To ensure logrotate to compress the log files of the virtualhost, the file */etc/logrotate.d/apache2* has to be altered to the following:

```
/var/log/apache2/*.log /var/www/path_to_directory/*.log
    {
        daily
        missingok
        rotate 14
        compress
        delaycompress
        notifempty
        create 640 root adm
        sharedscripts
        postrotate
                if /etc/init.d/apache2 status > /dev/null ;
                    then \
                    /etc/init.d/apache2 reload > /dev/null;
                        \
                fi;
        endscript
        prerotate
                if [ -d /etc/logrotate.d/httpd-prerotate ];
                    then \
                        run-parts /etc/logrotate.d/httpd-
                            prerotate; \
                fi; \
        endscript
}
```

8. In the next step, the encryption certificates for the tls service have to be created. The certificates are self-signed certificates, using SHA256 as hashing algorithm and having a validity of 10 years (3650 days). For this, the following command is used:

```
openssl req -sha512 -x509 -nodes -days 3650 -newkey rsa
    :4096 -keyout /etc/ssl/private/[servername].key -out
    /etc/ssl/certs/[servername].crt
```

During the creation process, the information for the certificate have to be provided. Also a Diffie-Hellman file has to be created, to improve the security of the tls connection by using the Diffie-Hellman paradigma for PerfectForwardSecrecy of the sessions:

```
sudo openssl dhparam -out /etc/ssl/certs/dhparam.pem
    4096
```

9. In the next step, the tls setting of the whole server are modified to meet most recent requirements on the cryptographic strength of the transaction. Therefore, a designated file under the following location */etc/apache2/conf-available/ssl-params.conf* is created. The content of the file is the following:

```
#modern security configuration
SSLProtocol                all -SSLv3 -TLSv1 -TLSv1.1
SSLCipherSuite             ECDHE-ECDSA-AES256-GCM-SHA384:
    ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-
    POLY1305:ECDHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-
    AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-
    ECDSA-AES256-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-
    ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256
SSLHonorCipherOrder        on
SSLCompression             off
SSLSessionTickets          off

# OCSP Stapling, only in httpd 2.3.3 and later
SSLUseStapling             on
SSLStaplingResponderTimeout 5
SSLStaplingReturnResponderErrors off
SSLStaplingCache           shmcb:/var/run/ocsp(128000)

SSLOpenSSLConfCmd DHParameters "/etc/ssl/certs/dhparam.
    pem"
```

This file defines the encryption standards, which the server accepts for connections. These standards might exclude older clients, such as Internet Explorer on Windows XP.

10. To enable the configuration, the following commands have to be executed:

- to enable the ssl module of apache2:

```
sudo a2enmod ssl
```

- to allow redirection from the non-https site to the https site:

```
sudo a2enmod headers
```

- to enable the https site:

```
sudo a2ensite sitename
```

- to enable the ssl-config in apache2:

```
sudo a2enconf ssl-params
```

- and finally the web server has to be restarted:

```
sudo systemctl restart apache2
```

Other files than those mentioned do not need to be altered to allow steady operation of the server. To allow public reaching of the server, the reuqired ports, which is essential 443, have to be cleared by the firewall.

6.2 CentOS 7

In CentOS 7 several modifications were necessary to provide a stable and safe operation of the database and the interfaces. The following describes the installation of the required packages to a scratch installation. In case a LAMP installation has been chosen during the installation process of the operating system, some of these steps might be superfluous.

6.2.1 Database installation and configuration

First the database had to be installed. Therefore **MariaDB** has been chosen, as it is the default database environment for RHEL-based Linux distributions. For the installation, mariadb-server has to be chosen, all dependencies are automatically fixed by the yum RPM wrapper. Furthermore PHP7 had to be installed, here it is important to install the database drivers as well, which are shipped in the package php70w-mysqlnd. Together with PHP7 httpd, as the web server, needs to be installed. For the access of httpd to the database it is important to execute the following commands:

```
sudo sestatus
sudo getsebool -a | grep httpd
sudo setsebool -P httpd_can_network_connect_db 1
```

6.2.2 Web server installation and configuration

As the LAMP stack is used for the project development, **apache2** is deployed as the default web server. apache2 is installed as a dependency of PHP7 and is shipped in the package httpd. To ensure proper operation, the server has to be configured for shipping of multiple websites beside each other, which is ensured by using virtual hosts. Therefore the following steps are necessary:

1. create a folder for the virtual host, using the following command:

```
sudo mkdir -P /var/www/path_to_directory/content
```

2. grant the required permissions on the folder to the user:

```
sudo chown -R $user:$user /var/www/path_to_directory/
    content
sudo chmod -R 755 /var/www/
```

3. create configuration files for the virtual host:

   - Create the folders for the virtual host configuration:

   ```
   sudo mkdir /etc/httpd/sites-available
   sudo mkdir /etc/httpd/sites-enabled
   ```

   - Add a line at the end of /etc/httpd/conf/httpd.conf:

   ```
   IncludeOptional sites-enabled/*.conf
   ```

7 License

8 Attachements