Cette activité permet de travailler sur un projet de migration d’une application métier avec changement de version et de système.

Table des matières

[Constat de départ et travail global à faire 1](#_Toc129288101)

[Etape 0 : installation des 2 serveurs 2](#_Toc129288102)

[Etape 1 : installation de la version 1.4.6 sur l’ancien serveur (jessie) 2](#_Toc129288103)

[Etape 2 : installation de cette même version sur le nouveau serveur (buster) 3](#_Toc129288104)

[Etape 3 : Installation de la version 1.9.2 de l’application MRBS sur le buster 3](#_Toc129288105)

[Installation de la version mrbs-1.9.2 sur le serveur buster 4](#_Toc129288106)

[Etape 4 : Migration des données entre les 2 serveurs 4](#_Toc129288107)

[Migration manuelle des données (Et là c’est le drame !) 4](#_Toc129288108)

[Migration manuelle des données (Et là c’est le drame !) (PARTIE SLAM) 5](#_Toc129288109)

[Migration automatique 5](#_Toc129288110)

[Observation des fichiers php de migration (partie SLAM) 6](#_Toc129288111)

[ANNEXES 7](#_Toc129288112)

[Code de la fonction do\_upgrade($upgrade\_handle) 7](#_Toc129288113)

[Code de la fonction upgrade\_database($local, $from, $to, DB $upgrade\_handle) 7](#_Toc129288114)

# Constat de départ et travail global à faire

Un de vos clients, la maison de Ligues de Lorraine utilise comme application web de réservation des salles une solution logicielle libre nommée MRBS <https://mrbs.sourceforge.io/>. Cette application tourne depuis plusieurs années sur un serveur LAMP installé sur un système Debian Jessie. Pour des raisons de sécurité, le service informatique est obligé de mettre à jour le serveur et la nouvelle version installée est Debian Buster.

Un nouveau **ticket de demande** a donc été créé sur le service-center et vous devez le traiter et produire des éléments pour la base de connaissance.

La migration de la solution web actuelle semble délicate et la solution MRBS a également évolué depuis sa première installation (aucune mise à jour n’a été faite car l’application était suffisamment fonctionnelle).

Ces deux contraintes vont nécessiter que vous testiez la migration sur un environnement de test et que vous **établissiez un mode opératoire de migration** qui servira ensuite pour les serveurs en production.

Pour cela, vous disposerez de 2 machines serveurs LAMP opérationnels de tests :

* Une Debian 8 Jessie identique au serveur actuel qui héberge le service ;
* Une Debian 10 Buster identique au nouveau serveur qui devra héberger le service.

*Les configurations de ces deux machines sont détaillées dans le fichier config-servers.txt*

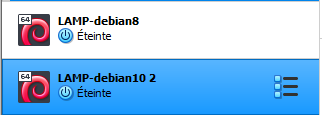
Vous disposez également d’un dossier *Install-MRBS* contenant le code de l’application web actuellement installée, un jeu d’essai de la base de données pour cette application et un mode opératoire pour l’installer sur le serveur Jessie.

Sur ce TP, vous allez pouvoir travailler en binôme en créant un tableau de bord pour l’avancement du projet sur planner et en vous partageant et attribuant les taches à réaliser. Utilisez un document collaboratif pour collaborer également à l’écriture du rapport.

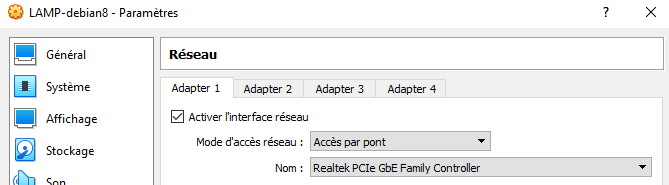
**Documents à rendre pour chaque binôme** :

* Compte-rendu d'activité pour chaque binôme contenant : l'explication du contexte, l'organisation du travail (planner), les étapes réalisées et le mode opératoire de l’installation de la nouvelle version de l'application, des tests et de la migration des données. Le rapport doit contenir des illustrations et des copies d’écran ;
* Annexes au rapport : Nouveau jeu de données d’essai donnees2.my.sql pour la nouvelle version de l’application

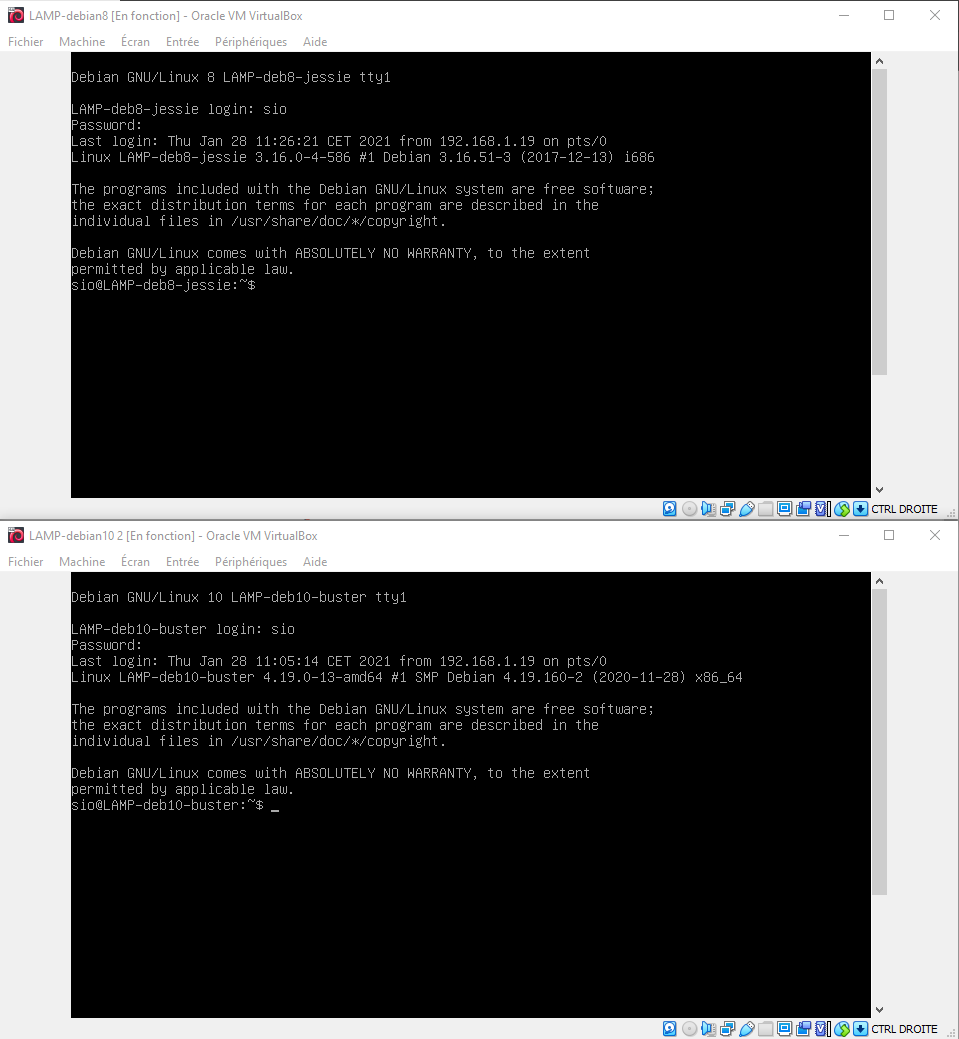
# Etape 0 : installation des 2 serveurs



Décompressez les 2 OVA pour les serveurs et vérifiez que les cartes réseaux de ces machines sont en accès par pont.



Connexion avec sio/sio



Lancez les 2 VM, changez leurs noms (modifier le hostname) et notez leurs IP pour faciliter la suite de votre activité.



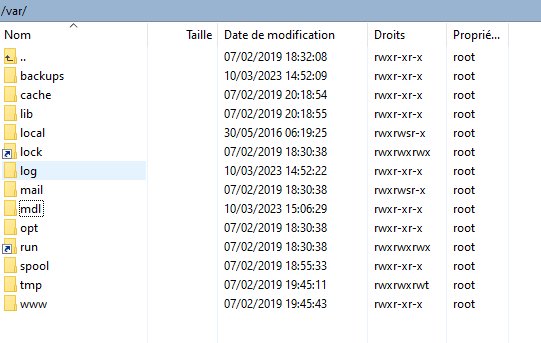


**IP serveur Jessie : 172.31.1.171**

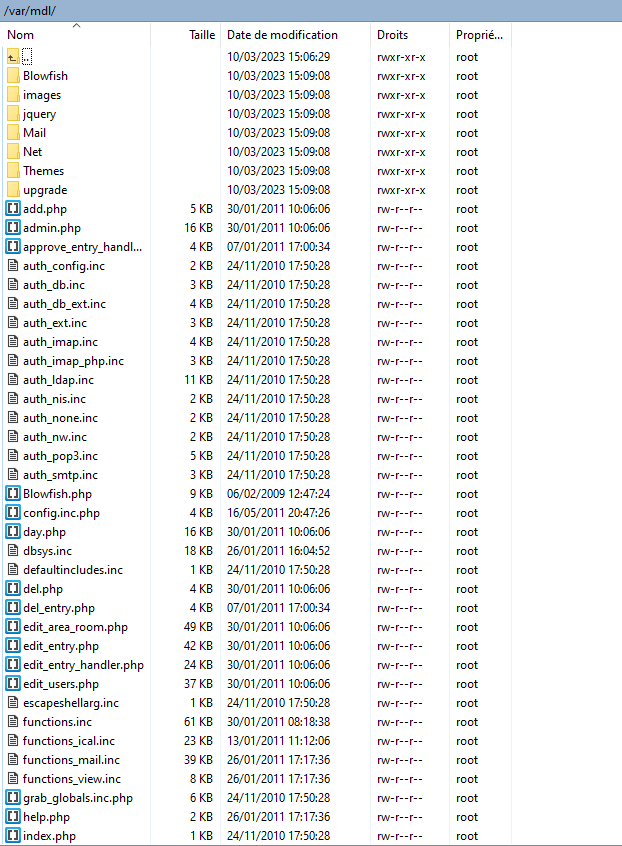
**IP serveur Buster : 172.31.1.70**

# Etape 1 : installation de la version 1.4.6 sur l’ancien serveur (jessie)

Installation de la version actuelle de l'application MRBS (mrbs-1.4.6) sur le serveur Jessie en utilisant les étapes du mode opératoire ***Install-MRBS-1.4.6.docx*** fourni.

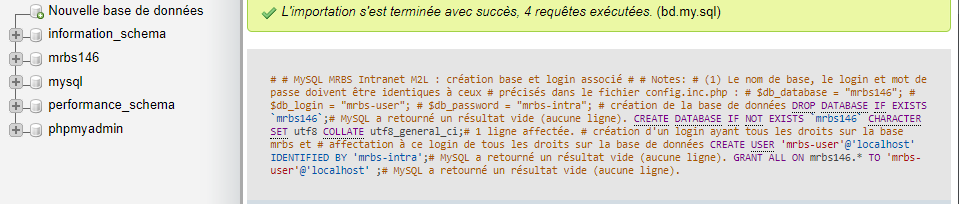
Utilisation de WinSCP et création d’un dossier mdl dans le répertoire du serveur web   


Mise des données du fichier web mrbs-1.4.6 dans le fichier mdl :

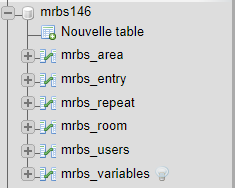


importer les fichiers SQL du répertoire *BDD* de l'archive

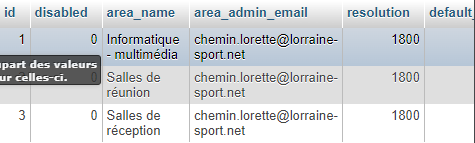
bd.my.sql : création de la base de données mrbs146 et d'un utilisateur mrbs-user qui aura tous les droits sur cette base

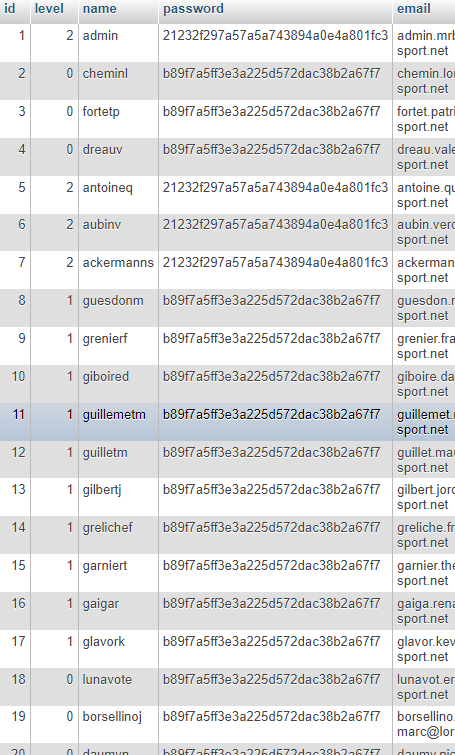


tables.my.sql : création de la structure des tables dans la base de données mrbs146



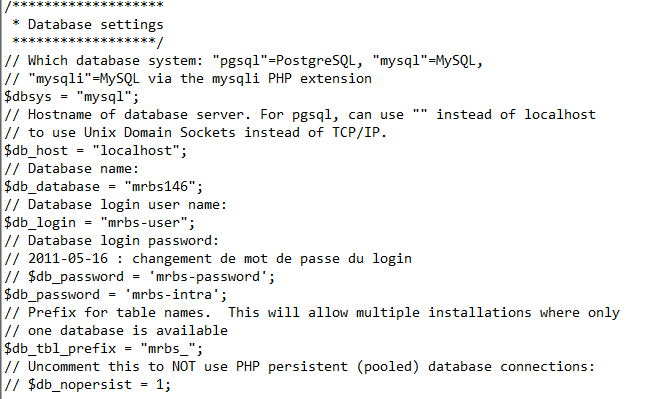
donnees.my.sql : insertion des domaines, salles ressources et utilisateurs



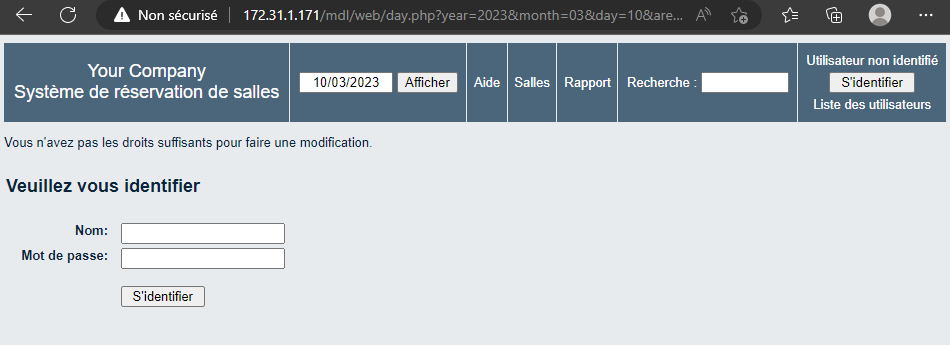


…

Mise à jour du driver d’accès :



Connexion au mdl :



Analysez les fichiers SQL fournis :

* A quoi sert le fichier tables.my.sql ?
  + Il sert à mettre en place toutes les tables dans la base de données
* A quoi sert le fichier donnees.my.sql ?
  + A instancier toutes les données dans chaque tables
* Dans le fichier bd.my.sql, que font les instructions suivantes :

DROP DATABASE IF EXISTS `mrbs146`;

CREATE DATABASE IF NOT EXISTS `mrbs146`

CHARACTER SET utf8 COLLATE utf8\_general\_ci;

**La requête crée une base de donnée nommée ‘mrbs146’**

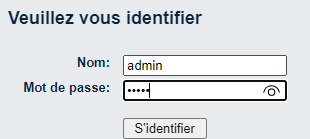
CREATE USER 'mrbs-user'@'localhost' IDENTIFIED BY 'mrbs-intra';

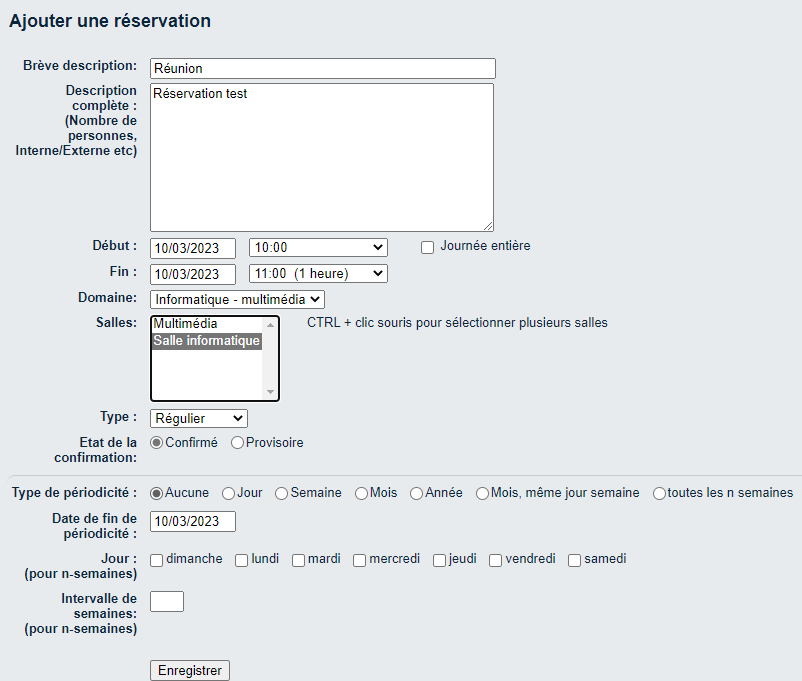
**La requête crée un utilisateur de nom ‘mrbs-intra’**

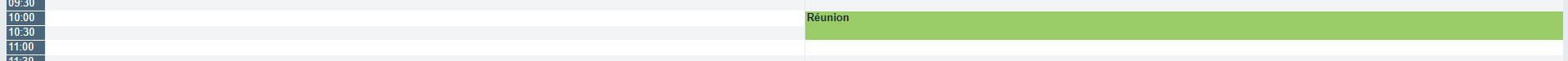
GRANT ALL ON mrbs146.\* TO 'mrbs-user'@'localhost' ;

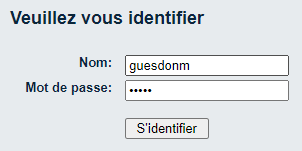
**La requête donne tous les droits à cet utilisateur**

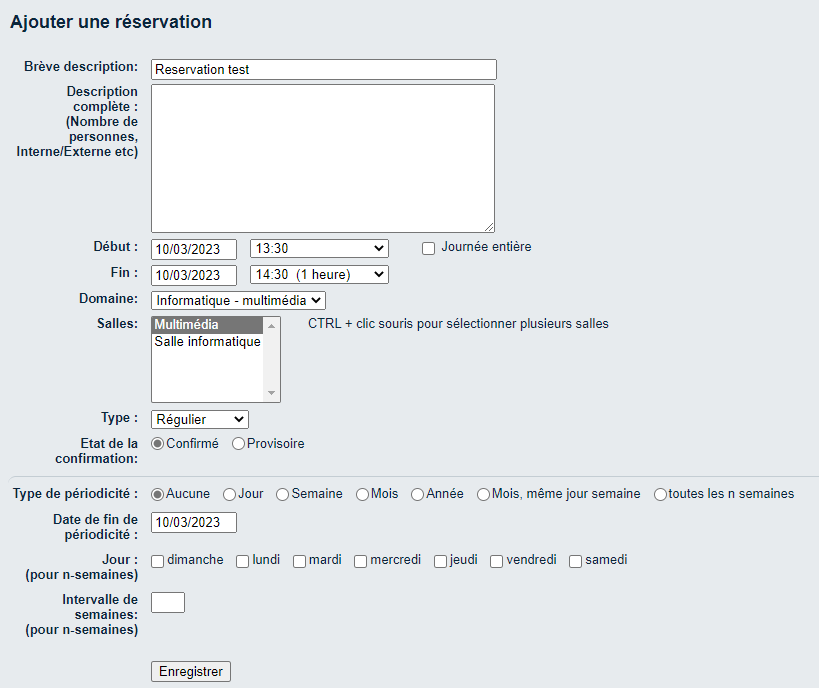
Vérifier que votre service de réservation de salles est maintenant fonctionnel en créant une nouvelle réservation avec un utilisateur de type admin et une autre avec un utilisateur de type usager.













Ou retrouve t’on les informations de ces réservations dans la base de données ?

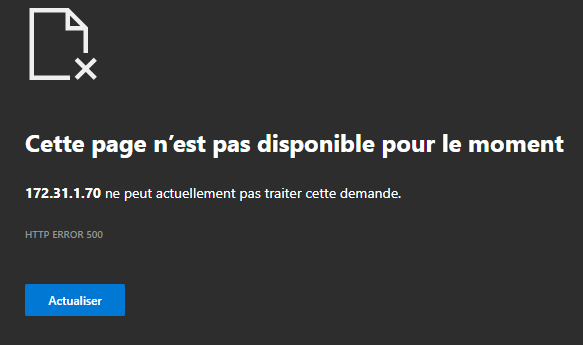
**Les données des réservations sont dans la table mrbs\_entry**

# Etape 2 : installation de cette même version sur le nouveau serveur (buster)

Installation de la version actuelle de l'application MRBS (mrbs-1.4.6) sur le serveur Buster en utilisant la même procédure que précédemment.

Réalisation des mêmes étapes puis affichage de la page

**Problèmes rencontrés ?**



Spoiler : Le code de l’ancienne application ne peut pas être exécuté sur le nouveau serveur pour des pbs d’écriture…

# Etape 3 : Installation de la version 1.9.2 de l’application MRBS sur le buster

De toute façon le projet était de profiter de la migration pour mettre à jour le service (c’est mieux en termes de sécurité).

Téléchargez la version 1.9.2 de la solution logicielle MRBS (mrbs-1.9.2) sur <https://sourceforge.net/projects/mrbs/files/> et vérifiez les conditions d'utilisation de l’application dans le fichier README pour valider que la configuration du serveur est suffisante :

|  |  |  |
| --- | --- | --- |
|  | Version minimum demandée | Version du serveur Buster |
| Version de PHP | 5.4.0 | 7.3.19 |
| Version de la base de données | 5.5.3 | 10.3.27 |
| Version du serveur web | ALL | 2.4.38 |

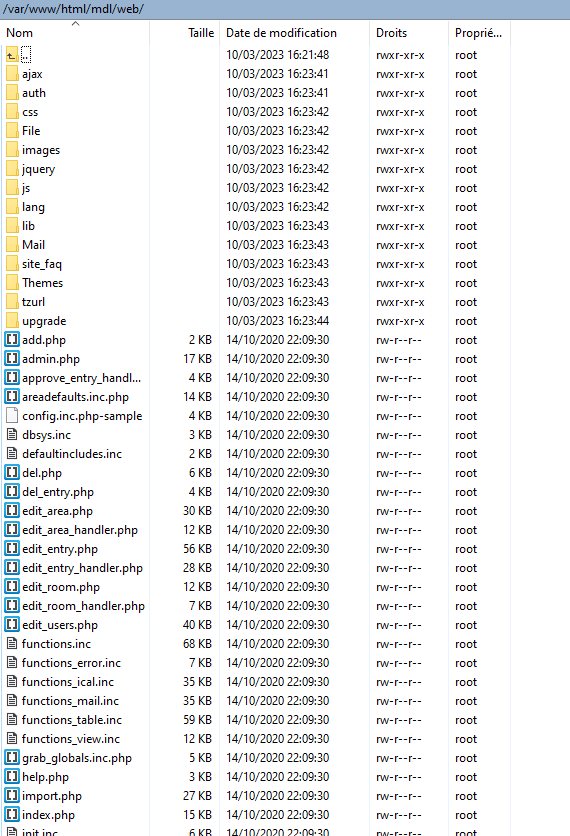
## Installation de la version mrbs-1.9.2 sur le serveur buster

Pour procéder à l’installation du service, vous n’avez pas de mode opératoire cette fois-ci…

Bien lire le fichier INSTALL dans l’archive que vous avez récupéré puis installez l’application et **complétez votre rapport avec un petit** **mode opératoire de l’installation de cette nouvelle version** en français.

Rapport mode opératoire :

1. Installer les fichiers MRBS sur le serveur web
   1. Installez le fichier mrbs1.9.2.tar.gz et le copier dans le serveur web en prenant soin de garder les anciens fichiers dans un dossier temporaire



* 1. Copier le contenu du fichier config.inc.php-sample sur config.inc.php tout en le configurant

On mettra le site dans un dossier mrbs192 et créera les tables dans une base mrbs192. Attention, pensez à donner les droits sur les tables de cette base à votre utilisateur mrbs-user créé à l’étape précédente. Cette fois on va restreindre ses droits aux opérations sur les données : lecture, modification, insertion et suppression.

**Instructions pour créer la base mrbs192 et donner les droits sur les tables de cette base à l’utilisateur mrbs-user existant :**

- Créer une base de donnée nomée mrbs192 puis importer toutes les tables/données  
- GRANT SELECT, INSERT, UPDATE, DELETE ON mrbs192.\* TO 'mrbs-user'@'localhost';

Vérification que le service de réservation de salle est accessible. Si vous avez oublié de le faire avant, certains paramètres sont à modifier dans le fichier de configuration config.inc.php…

Quand votre serveur sera opérationnel, vous devez arriver sur la page de configuration du service de réservation. Ce n’était pas le cas lors de de l’étape 1, vous aviez accès aux utilisateurs et il la visualisation des salles et des réservations.

**Pourquoi ?** (piste : allez voir ce qu’il y a dans la bdd pour mrbs192 comparé à mrbs146).

# Etape 4 : Migration des données entre les 2 serveurs

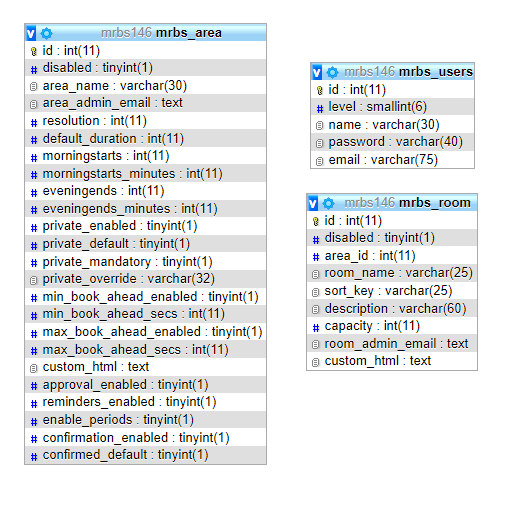
## Migration manuelle des données (Et là c’est le drame !)

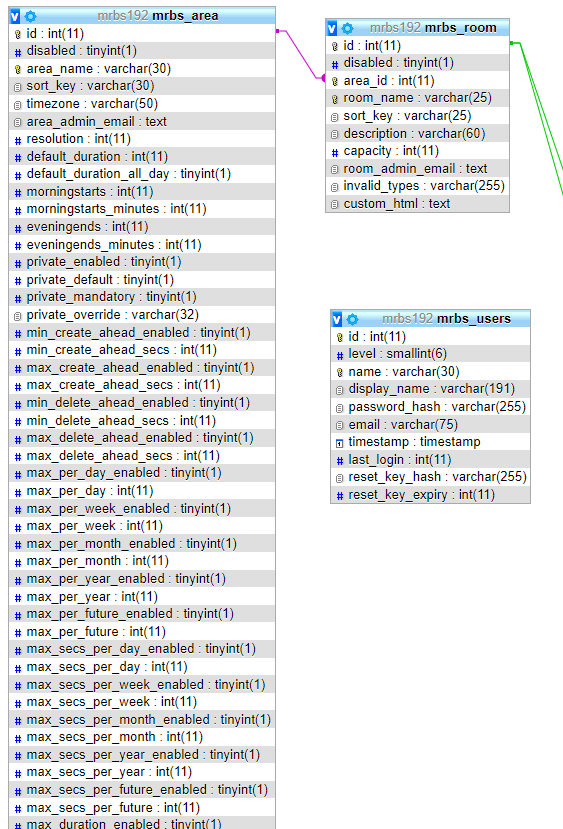
On voudrait bien récupérer les données du jeu d’essai fourni pour l’étape 1 (donnees.my.sql) mais les structures des tables ont évolué entre les 2 versions de l’application rendant l’opération impossible.

Vous pouvez essayer et constater les erreurs SQL (**copie d’écran**)

Pour rappel, ce script insérait des données dans les tables area, room et user. Observez les différences entre les 2 schémas de base de données pour ces trois tables qui expliquent ces erreurs :

**Faites une capture du modèle relationnel de ces 3 tables pour chaque base (avec le concepteur)**

Mrbs-146 : 

Mrbs-192 : 

## Migration manuelle des données (Et là c’est le drame !) (PARTIE SLAM)

L’idée donc d’adapter les instructions du script donnees.my.sql fourni pour l’ancienne version de l’application en l’adaptant à la nouvelle structure. **Le script d’insertion des données dans la nouvelle table area mrbs\_area\_nouvelle\_version.sql vous est fourni dans les ressources, il faut l’importer dans votre base mrbs2..**

Vous allez étudier comment on pourrait faire évoluer les données du script pour **les 2 autres tables restantes**. Pour cela, vous devez identifier les différences entre les tables qui existent dans les deux modèles.

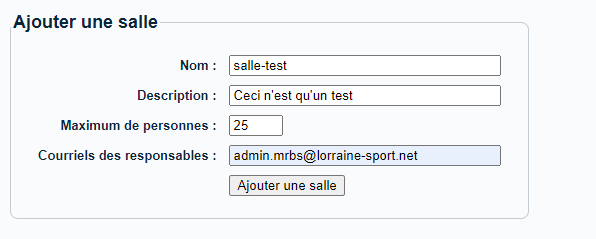
Pour vous aider, créez manuellement sur le nouveau site un utilisateur admin (nom : admin, mail : [admin.mrbs@lorraine-sport.net](mailto:admin.mrbs@lorraine-sport.net), mdp : admin, nom : admin)++



INSERT INTO `mrbs\_users` (`id`, `level`, `name`, `display\_name`, `password\_hash`, `email`, `timestamp`, `last\_login`, `reset\_key\_hash`, `reset\_key\_expiry`) VALUES

(1, 2, 'admin', 'admin', '$2y$10$bpJmBLadLT2Qjtl64iFn3uMU/enyfZDAs7pqIOv2sdrf0MwRQTQKu', 'admin.mrbs@lorraine-sport.net', '2023-03-17 14:06:35', 1679062002, '', 0);

Créez ensuite une nouvelle salle pour ce lieu (nom : salle-test, max personne : 25)



INSERT INTO `mrbs\_room` (`id`, `disabled`, `area\_id`, `room\_name`, `sort\_key`, `description`, `capacity`, `room\_admin\_email`, `invalid\_types`, `custom\_html`) VALUES

(1, 0, 1, 'salle-test', 'salle-test', 'Ceci n\'est qu\'un test', 25, 'admin.mrbs@lorraine-sport.net', NULL, NULL);

Faites un export des données des 2 tables mrbs\_room et mrbs\_users pour voir les instructions SQL INSERT générées et les comparer à celles du script donnees.my.sql.

En confrontant ces deux versions des requêtes (et avec la structure des nouvelles tables), vous devriez maintenant pouvoir réécrire les instructions pour créer quelques utilisateurs et les salles de l’ancien jeu de données (partagez-vous le travail). Exécutez vos instructions pour les valider.

**Instructions SQL :**

**INSERT INTO `mrbs\_users` (`id`, `level`, `name`, `display\_name`, `password\_hash`, `email`, `timestamp`, `last\_login`, `reset\_key\_hash`, `reset\_key\_expiry`) VALUES**

**(2, 0, 'cheminl', 'cheminl', 'b89f7a5ff3e3a225d572dac38b2a67f7', 'chemin.lorette@lorraine-sport.net', null, 0, '',0),**

**(3, 0, 'fortetp', 'fortetp', 'b89f7a5ff3e3a225d572dac38b2a67f7', 'fortet.patrick@lorraine-sport.net', null, 0, '',0),**

**(4, 0, 'dreauv', 'dreauv', 'b89f7a5ff3e3a225d572dac38b2a67f7', 'dreau.valerie@lorraine-sport.net’, null, 0, '',0),**

**(5, 2, 'antoineq', 'antoineq', '21232f297a57a5a743894a0e4a801fc3', 'antoine.quentin@lorraine-sport.net', null, 0, '',0),**

**(6, 2, 'aubinv', 'aubinv', '21232f297a57a5a743894a0e4a801fc3', 'aubin.veronique@lorraine-sport.net', null, 0, '',0),**

**(7, 2, 'ackermanns', 'ackermanns', '21232f297a57a5a743894a0e4a801fc3', 'ackermann.solange@lorraine-sport.net', null, 0, '',0),**

**(8, 1, 'guesdonm', 'guesdonm', 'b89f7a5ff3e3a225d572dac38b2a67f7', 'guesdon.martin@lorraine-sport.net', null, 0, '',0),**

**(9, 1, 'grenierf', 'grenierf', 'b89f7a5ff3e3a225d572dac38b2a67f7', 'grenier.francoise@lorraine-sport.net', null, 0, '',0)**

**INSERT INTO `mrbs\_room` (`id`, `disabled`, `area\_id`, `room\_name`, `sort\_key`, `description`, `capacity`, `room\_admin\_email`, `invalid\_types`, `custom\_html`) VALUES**

**(2, 0, 2, 'Corbin', 'Corbin', '', 15, '', NULL, NULL),**

**(3, 0, 2, 'Baccarat', 'Baccarat', '', 20, '', NULL, NULL),**

**(4, 0, 2, 'Longwy', 'Longwy', '', 12, '', NULL, NULL),**

**(5, 0, 1, 'Multimédia', 'Multimédia', '', 25, '', NULL, NULL),**

**(6, 0, 3, 'Amphithéâtre', 'Amphithéâtre', '', 200, '', NULL, NULL),**

**(7, 0, 2, 'Lamour', 'Lamour', '', 30, '', NULL, NULL),**

**(8, 0, 2, 'Grüber', 'Grüber', '', 15, '', NULL, NULL),**

**(9, 0, 2, 'Majorelle', 'Majorelle', '', 40, '', NULL, NULL),**

**(10, 0, 2, 'Daum', 'Daum', '', 15, '', NULL, NULL)**

Si vous n’avez pas rencontré d’erreurs, vous devriez maintenant avoir une version fonctionnelle du serveur de réservation de salle avec une partie de l’ancien jeu d’essai ! Cependant cette stratégie est fastidieuse !!!

**Indiquez dans votre rapport la stratégie utilisée pour adapter les données au nouveau modèle.**

**Pour adapter les données au nouveau modèle, j’ai ajouté depuis le site des utilisateurs et des salles pour voir le code utilisé puis j’ai ajouté certaines données**

## Migration automatique

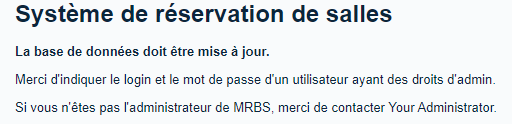
L’application mrbs prévoit une migration automatique des tables entre les versions, à partir de la version 1.2.x.

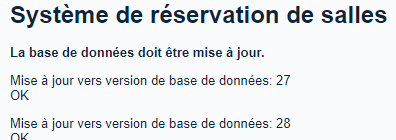
Modifiez le fichier de configuration config.inc.php de votre site mrbs2 pour lier l’application à l’ancienne base de données mrbs (qui contenait donc l’ensemble des anciennes données du script donnees.my.sql).

Allez sur le site qui va constater que votre base nécessite une mise à jour et renseignez les données d’un utilisateur ayant les droits sur la base de données mrbs pour permettre la migration (votre utilisateur mrbs fera l’affaire).

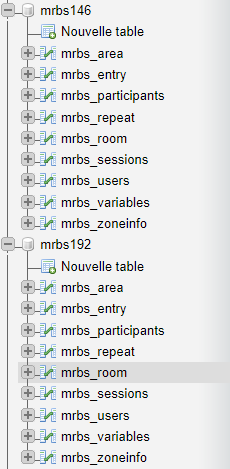
**Qu’est-ce que le site vous affiche une fois les migrations effectuées :**





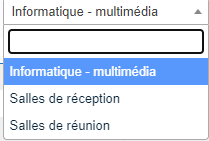


Constatez dans phpmyadmin l’évolution de la structure de la base de données mrbs qui doit maintenant être identique à mrbs2.



Vérifiez en utilisant l’application qu’on a bien récupéré les domaines, les salles et les utilisateurs de l’ancienne application.

Il y a bien les mêmes salles ET LES BONS UTILISATEURS





## Observation des fichiers php de migration (partie SLAM)

L’application a pu faire évoluer la bdd en opérant un ensemble de modifications SQL depuis le code php. Les éléments de code concernés sont repris en annexe au TP.

Le fichier defaultincludes/inc à la racine du site contient l’instruction require\_once 'upgrade.inc'; qui charge le fichier en question sur les pages du site. Ce fichier contient lui-même ligne 80 un appel à la fonction do\_upgrade() présente dans le même fichier à la ligne 221.

**Analyser le code de la fonction do\_upgrade($upgrade\_handle) pour indiquer ce qu’elle fait.**

Elle est composée d’une condition (si la base de donnée n’existe pas, cela ne va pas marcher)

Sinon elle va récupérer les versions des schémas de base de données puis si la version est inférieure à celle demandée elle va effectuer les modifications de MRBS puis les modifications au niveau local

SI la version actuelle de la BDD est inférieure à celle demandée par l’application, cette fonction réalise un appel à la fonction upgrade\_database() présente également dans ce fichier à la ligne 141.

**Analyser le code de la fonction function upgrade\_database($local, $from, $to, DB $upgrade\_handle) pour indiquer ce qu’elle fait.**

Cette fonction va mettre à jour toutes les tables de la base de données en comparant avec l’ancienne version

Lors de la migration précédemment effectuée, le site vous avait affichée des numéros de mise à jour successives.

**Quelles lignes dans le code de la fonction correspondent à ces affichages ?**

function upgrade\_echo($message)

{

echo $message;

// Flush the message, so that there's some progress information

// output to the browser even when the upgrade is taking a while

if (ob\_get\_length() !== false)

{

ob\_flush();

}

flush();

}

Pour chaque évolution de version, cette fonction a donc exécuté chacune des instructions de chacun des scripts présents dans chaque sous dossier de numéro de version du dossier upgrade du site.

Observez les scripts des dossiers 28 à 82 pour comprendre les instructions SQL que l’application a effectué.

# ANNEXES

## Code de la fonction do\_upgrade($upgrade\_handle)

|  |
| --- |
| // Upgrades the database tables defined by the current config context  function do\_upgrade($upgrade\_handle)  {    $result = true;    if (!db()->table\_exists(\_tbl('entry')))    {      upgrade\_echo("<p>" . htmlspecialchars(get\_vocab('no\_tables\_found')) . "</p>\n");    }    else    {      $current\_db\_schema\_version = db\_schema\_version();      $current\_db\_schema\_version\_local = db\_schema\_version\_local();      // Do any MRBS upgrades first      if ($result && ($current\_db\_schema\_version < DB::DB\_SCHEMA\_VERSION))      {        $result = $result && upgrade\_database(false, $current\_db\_schema\_version, DB::DB\_SCHEMA\_VERSION, $upgrade\_handle);      }      // Then any local upgrades      if ($result && ($current\_db\_schema\_version\_local < DB::DB\_SCHEMA\_VERSION\_LOCAL))      {        $result = $result && upgrade\_database(true, $current\_db\_schema\_version\_local, DB::DB\_SCHEMA\_VERSION\_LOCAL, $upgrade\_handle);      }    }    return $result;  } |

## Code de la fonction upgrade\_database($local, $from, $to, DB $upgrade\_handle)

|  |
| --- |
| // Upgrade between database schema versions.  // Returns FALSE on error, TRUE is successful  function upgrade\_database($local, $from, $to, DB $upgrade\_handle)  {    // $local is a boolean specifying whether the upgrades are global MRBS ones ($local === false)    // or local upgrades ($local === true);    // $upgrade\_handle is the database handle to use for the upgrade.   It will typically    // have admin rights (eg CREATE and ALTER)    global $dbsys;    $sql\_type = $dbsys;    if ($sql\_type == 'mysqli')    {      $sql\_type = 'mysql';    }    for ($ver = ($from+1); $ver <= $to; $ver++)    {      upgrade\_echo('<p>' .                   (($local) ? get\_vocab('upgrade\_to\_local\_version') : get\_vocab('upgrade\_to\_version')) .                   ": $ver");      if ($local)      {        $filename = "upgrade/local/$ver/$sql\_type.sql";        $php\_filename = "upgrade/local/$ver/post.inc";      }      else      {        $filename = "upgrade/$ver/$sql\_type.sql";        $php\_filename = "upgrade/$ver/post.inc";      }      $handle = fopen($filename, 'r');      if (!$handle)      {        // No need to localise, should never happen!        upgrade\_echo("Fatal error: Failed to open '$filename' for reading.\n");        return false;      }      $file\_size = filesize($filename);      $sql = (!empty($file\_size)) ? fread($handle, filesize($filename)) : '';      fclose($handle);      // PostgreSQL databases can have multiple schemas and so need a qualified      // table name when referring to the table name.  However the table prefix is also      // used to make, for example, constraint, index and trigger names unique.  These      // are unique within the schema and do not need to be schema-qualified.  Also, when      // the tables were created from tables.pg.sql they would just have had "mrbs\_"      // replaced by the unqualified table prefix.      $sql = str\_replace('%DB\_TBL\_PREFIX%', \_tbl('', true), $sql);      $sql = str\_replace('%DB\_TBL\_PREFIX\_SHORT%', \_tbl('', false), $sql);      foreach (explode(";", $sql) as $query)      {        // Skip any empty query (so that last semi-colon doesn't run        // an empty query)        if (preg\_match("/\S/", $query))        {          $res = $upgrade\_handle->query($query);        }      }      upgrade\_echo('<br>' . get\_vocab('ok'));      if ($ver > 1)      {        $variable\_name = ($local) ? "local\_db\_version" : "db\_version";        $upgrade\_handle->command("UPDATE " . \_tbl('variables') . " SET variable\_content = ? ".                                 "WHERE variable\_name = ?", array($ver, $variable\_name));      }      upgrade\_echo("</p>\n");      // Now execute the PHP file if there is one      if (is\_readable($php\_filename))      {        include($php\_filename);      }    }    return true;  } |