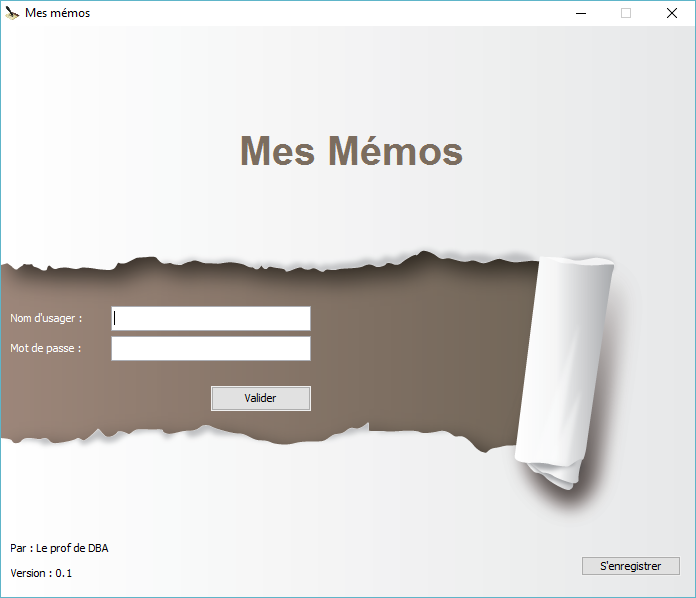
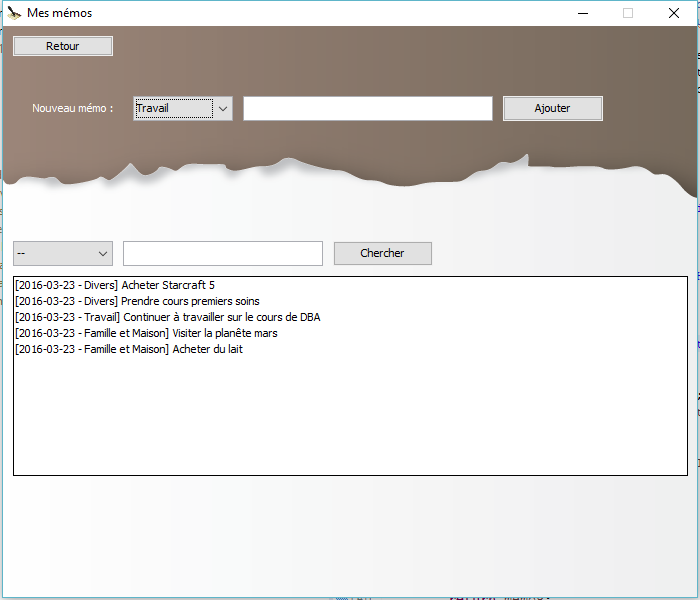
# Intégration de MySQL dans un projet existant



## L’application « Mes Mémos »

Cette application permet aux usagers de créer et de consulter une liste de mémos. Voici un exemple qui affiche tous les mémos de l’usager Fred.



## Notes

* Pour réaliser ce projet, je vous recommande FORTEMENT de suivre les étapes en ordre.
* IntelliJ a été utilisé pour créer le programme, vous devriez donc l’utiliser pour compléter ce travail.
* Si vous avez des questions, n’hésitez pas à me demander de l’aide.
* Vous devriez conserver un fichier, tables.sql, qui contiendra l’énoncé des tables que vous créerez.
* N’oubliez pas de tester chaque fonctionnalité développée afin de s’assurer qu’elle fonctionne correctement.
* **IMPORTANT** : Ne modifiez en aucun cas la classe « CorClient.java », sous peine d’avoir 0.

## Exemples d’accès à MySQL dans le programme

### Exemple 1 : Lire une liste d’items

Pour ces deux **exemples**, prenez pour acquis que userId est défini ainsi que la classe Item.

List<Item> items = new ArrayList<Item>();

Connection connection = DBConnection.getConnection();

try {

PreparedStatement statement = connection.prepareStatement(

"SELECT \* FROM items WHERE id\_user = ? ");

statement.setInt(1, userId);

ResultSet result = statement.executeQuery();

while(result.next()) {

items.add(new Item(result.getInt("id"), result.getString("name")));

}

statement.close();

} catch (SQLException e) {

e.printStackTrace();

}

return items;

### Exemple 2 : Supprimer un item

boolean success = false;

Connection connection = DBConnection.getConnection();

PreparedStatement statement;

try {

statement = connection.prepareStatement("DELETE FROM items WHERE id = ?");

statement.setInt(1, id);

statement.execute();

statement.close();

success = true;

} catch (SQLException e) {

e.printStackTrace();

}

return success;

## Les tâches à réaliser

Ces tâches doivent être réalisées en ordre, autrement il sera difficile de tester que tout fonctionne.

### Tâche #0 : S’assurer que le TimeZone de MySQL est défini

Dans MySQL Workbench, écrire ceci :

SET GLOBAL time\_zone = '-5:00';

### Tâche #1 : Création de la base de données MySQL (dans MySQL Workbench)

Cette base de données MySQL contiendra les tables de l’application. Pour cette question, il s’agit de créer la base de données (create database …), pas les tables.

#### Notes/Énoncés SQL:

CREATE DATABASE mes\_memos\_db CHARACTER SET utf8 COLLATE utf8\_general\_ci;

### Tâche #2 : Avoir une connexion à la base de données valide (dans MySQL Workbench)

Afin d’avoir accès à la base de données, il est nécessaire de se créer un usager MySQL et d’y donner les droits nécessaires.

#### Notes/Énoncés SQL:

CREATE USER 'mes\_memos\_user'@'localhost' identified by "AAAaaa111";

GRANT ALL ON mes\_memos\_db.\* TO 'mes\_memos\_user'@'localhost';

### Tâche #3 : Spécifier les informations de connexion à la base de données MySQL

Pour que le programme puisse avoir accès à la base de données, il faut spécifier les informations de connexion nécessaires. Pour ce faire, allez dans la classe dao/DBConnexion.java et modifiez les informations qui se trouvent dans la méthode « getConnection() ».

Si, lorsque vous exécutez le programme Java, une erreur qui est affichée, c’est parce que les informations de connexion sont erronées.

#### Notes:

dataSource.setUser("mes\_memos\_user");

dataSource.setPassword("AAAaaa111");

dataSource.setServerName("localhost");

dataSource.setDatabaseName("mes\_memos\_db");

### Tâche #4 : Modifier le nom du créateur du projet

La première tâche concrète s’agit de modifier le nom du créateur affiché sur la page de connexion de l’application. Je vous laisse le soin de fouiller dans le code du programme afin de voir où faire la modification. Le nouveau nom du créateur devrait être le vôtre.



Le but de cette tâche est de vous familiariser avec la structure du code, il n’y a donc pas d’accès à la base de données.

#### Notes:

super.addLabel("Par : Ahmed Sadek", 10, 510, 200, LINE\_HEIGHT);

### Tâche #5 : Terminer le module d’enregistrement

Afin d’avoir accès au système (comme usager), il est nécessaire de pouvoir s’y enregistrer. Pour ce faire, il y a un bouton en bas à droite de la page de connexion.

Tout ce qu’il reste à faire est de créer la table « users » et de **compléter** la méthode *UserDAO.register().*

Les informations liées à un usager sont :

* id : clé primaire de la table
* username : chaîne de caractères de 40 caractères maximum , qui doit être unique
* password : chaîne de caractères de 255 caractères maximum

Note : Le mot de passe DOIT ÊTRE « haché » avant d’être inséré dans la base de données! Pour ce faire, utilisez cette ligne dans la méthode *UserDAO.register*():

Note 2 : Pour ce laboratoire, le nom de vos tables doit nécessairement être en minuscule.

String motDePasse = BCrypt.hashpw(String.valueOf(password), BCrypt.gensalt());

#### Notes/Énoncés SQL:

USE mes\_memos\_db;

CREATE TABLE user (

id INT NOT NULL AUTO\_INCREMENT,

username VARCHAR(40) NOT NULL UNIQUE,

password VARCHAR(255) NOT NULL,

PRIMARY KEY pk\_user(id)

) ENGINE = innoDB CHARACTER SET utf8 COLLATE utf8\_general\_ci;

        try {

            PreparedStatement statement  = connection.prepareStatement("INSERT INTO user(username, password) VALUES(?, ?)");

            statement.setString(1, username);

            String motDePasse = BCrypt.hashpw(String.valueOf(password), BCrypt.gensalt());

            statement.setInt(2, motDePasse);

            statement.execute();

            statement.close();

            success = true;

        } catch (SQLException e) {

            e.printStackTrace();

        }

### Tâche #6 : Terminer le module de connexion

Puisqu’il est maintenant possible de s’enregistrer, il s’agit maintenant de pouvoir s’y connecter.

Il faut donc terminer la méthode *UserDAO.login()*.

Étant donné que le mot de passe est haché, il est nécessaire d’implémenter la méthode en suivant les étapes suivantes :

1. Aller chercher l’enregistrement de la table « users » dont le *username* correspond à celui spécifié par l’usager (faire un SELECT … WHERE username = ?).
2. Si l’usager existe, faire la vérification du mot de passe de la façon suivante :

String pwd = … // Mot de passe haché qui vient de la base de données

if (BCrypt.checkpw(String.valueOf(password), pwd)) {

// Succès

}

else {

// Mauvais mot de passe

}

1. Si le mot de passe est bon, alors construire un objet « User » (new User(…)), qui doit être retourné par la méthode. Autrement, il faut retourner NULL.

#### Notes/Énoncés SQL:

try {  
 PreparedStatement statement = connection.prepareStatement("SELECT \* FROM user WHERE username = ?");  
 statement.setString(1, username);  
 ResultSet result = statement.executeQuery();  
  
 if(result.next()) {  
 if (BCrypt.checkpw(String.valueOf(password), result.getNString("password"))) {  
 user = new User(result.getInt("id"), username);  
 }  
 else {  
 user = null;  
 }  
 }  
  
 statement.close();  
} catch (SQLException e) {  
 e.printStackTrace();  
}

### Tâche #7 : Terminer la création d’une catégorie

Lorsque vous êtes en mesure de vous connecter, il est temps de pouvoir ajouter des catégories de mémos. La méthode *CategoryDAO.addCategory()* doit donc être terminée. Important : le nom d’une catégorie est unique, c’est-à-dire que pour un usager, il ne doit pas être possible d’en créer deux avec le même nom. Vous devez donc créer un index unique en conséquence.

Les catégories contiennent :

* id : identifiant de la table
* id\_user : identifiant de l’usager
* name : Nom de la catégorie (255 car. max)

#### Notes/Énoncés SQL:

CREATE TABLE category (

id INT NOT NULL AUTO\_INCREMENT,

id\_user INT NOT NULL,

name VARCHAR(255) NOT NULL,

PRIMARY KEY pk\_category(id)

) ENGINE = innoDB CHARACTER SET utf8 COLLATE utf8\_general\_ci;

CREATE INDEX idx\_category ON category(id\_user, name);

try {  
 PreparedStatement statement = connection.prepareStatement("INSERT INTO category(id\_user, name) VALUES(?, ?)");  
 statement.setInt(1, userId);  
 statement.setString(2, name);  
 statement.execute();  
 statement.close();  
 success = true;  
} catch (SQLException e) {  
 e.printStackTrace();  
}

### Tâche #8 : Terminer l’affichage des catégories

L’insertion étant terminé, il serait pratique de pouvoir les consulter dans le programme. Il faut donc terminer la méthode *CategoryDAO.getCategoryList()*.

N’oubliez pas que les catégories/mémos sont par usager! Deux usagers ne se partagent pas les mêmes mémos/catégories.

#### Notes/Énoncés SQL:

try {  
 PreparedStatement statement = connection.prepareStatement("SELECT \* FROM category WHERE id\_user = ?");  
 statement.setInt(1, userId);  
 ResultSet result = statement.executeQuery();  
  
 while(result.next()) {  
 categories.add(new Category(result.getInt("id"), userId, result.getNString("name")));  
 }  
  
 statement.close();  
} catch (SQLException e) {  
 e.printStackTrace();  
}

### Tâche #9 : Terminer la suppression d’une catégorie

Il ne reste plus qu’à terminer la méthode de suppression de catégorie, via la méthode *CategoryDAO.deleteCategory()*.

#### Notes/Énoncés SQL:

try {  
 PreparedStatement statement = connection.prepareStatement("DELETE FROM category WHERE id = ?");  
 statement.setInt(1, id);  
 statement.execute();  
 statement.close();  
 success = true;  
} catch (SQLException e) {  
 e.printStackTrace();  
}

### Tâche #10 : Terminer l’ajout d’un mémo

Il est temps d’attaquer le dernier module du programme, soit les mémos. Dans cette tâche, il faut compléter l’ajout des mémos via la méthode *MemoDAO.addMemo()*.

Un mémo contient :

* id : clé primaire de la table
* id\_category : id de la catégorie
* memo : champs texte (« TEXT »).
* created : champ de type « timestamp »

Pour l’insertion de la date du mémo, utilisez une logique similaire à cet exemple :

INSERT INTO products (name, created) VALUES (?, NOW()) -- NOW() est une fonction qui retourne la

date actuelle

#### Notes/Énoncés SQL:

CREATE TABLE memo (

id INT NOT NULL AUTO\_INCREMENT,

id\_category INT NOT NULL,

memo TEXT NOT NULL,

created TIMESTAMP,

PRIMARY KEY pk\_memo(id)

) ENGINE = innoDB CHARACTER SET utf8 COLLATE utf8\_general\_ci;

try {  
 PreparedStatement statement = connection.prepareStatement("INSERT INTO memo(id\_category, memo, created) VALUES(?, ?, NOW())");  
 statement.setInt(1, categoryId);  
 statement.setString(2, memo);  
 statement.execute();  
 statement.close();  
 success = true;  
} catch (SQLException e) {  
 e.printStackTrace();  
}

### Tâche #11 : Terminer l’affichage des mémos

Cette fonction permet de retourner la liste de mémos de l’usager. Comme vous pourrez le constater, la méthode *MemoDAO.getMemoList*() prend en paramètre l’id de l’usager. Il faudra donc passer par la table « categories » afin de retrouver les mémos de l’usager (puisque la table mémo ne possède pas la colonne id\_user).

INNER JOIN, ça vous dit quelque chose? Voici un exemple simple :

SELECT

products.\*,

users.username

FROM

users,

products

WHERE

products.id\_user = users.id

**ou une autre version :**

SELECT

products.\*,

users.username

FROM

users JOIN products ON items.id = products.id\_user;

Note : Les mémos doivent être affichés en ordre décroissant selon leur date de création.

#### Notes/Énoncés SQL:

try {  
 PreparedStatement statement = connection.prepareStatement("SELECT \* FROM memo JOIN category ON memo.id\_category = category.id WHERE id\_user = ?");  
 statement.setInt(1, userId);  
 ResultSet result = statement.executeQuery();  
  
 while(result.next()) {  
 memos.add(new Memo(result.getInt("id"), result.getNString("name"), result.getNString("memo"), result.getDate("created")));  
 }  
  
 statement.close();  
} catch (SQLException e) {  
 e.printStackTrace();  
}

### Tâche #12 : Terminer la suppression d’un mémo

Cette tâche consiste à permettre la suppression d’un mémo. Pour ce faire, complétez la méthode *MemoDAO.deleteMemo().*

#### Notes/Énoncés SQL:

try {  
 PreparedStatement statement = connection.prepareStatement("DELETE FROM memo WHERE id = ?");  
 statement.setInt(1, id);  
 statement.execute();  
 statement.close();  
 success = true;  
} catch (SQLException e) {  
 e.printStackTrace();  
}