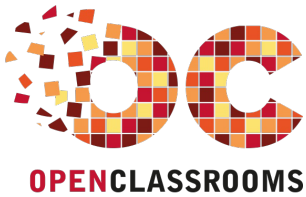
Utiliser l'API MySQL dans vos programmes

Par Henon Fabien



www.openclassrooms.com

Dernière mise à jour le 8/09/2012

Sommaire

Utiliser l'API MySQL dans vos programmes

Par  Henon Fabien

Mise à jour : 08/09/2012
Difficulté : Facile Durée d'étude : 3 heures

Tout d'abord **bonjour tout le monde** et bienvenue dans mon premier tutoriel pour le site du zéro !

Je vous apprendrai ici à utiliser l'*API MySQL* dans vos programmes pour les rendre encore plus complets et intéressants, à condition pour cela que vous compreniez bien ce que je vais vous apprendre.
A la fin de ce cours, vous devriez être capable de vous connecter à une base de données (*BDD*) et de faire des requêtes SQL depuis vos programmes *C/C++*.

Si vous êtes toujours partant, attachez vos ceintures, et préparez-vous à partir à l'aventure !
Sommaire du tutoriel :

Préface

Pré-requis

Pour pouvoir comprendre au mieux ce tutoriel, quelques connaissances sont requises.

MySQL

Vous devez savoir ce qu'est une BDD (*base de données*), ainsi que tout le vocabulaire comme : *table, champ, ligne, requête SQL, ...*
Il vous est aussi conseillé de connaître les bases du langage *SQL* pour pouvoir comprendre les requêtes de base.

S'il vous manque certaines connaissances à ce sujet je vous conseille de lire le tuto de M@teo sur le PHP, de cette manière vous serez plus en mesure de comprendre ce tutoriel.

C/C++

Il vous faudra avoir des bases en *C* (pointeurs, fonctions, conditions, boucles, installation d'une bibliothèque, etc.) pour comprendre ce tutoriel. Dans le cas échéant, je vous conseille encore une fois un cours de M@teo21 (jusqu'à la partie 3 minimum).

API MySQL, mais pour faire quoi?

Ben oui, à quoi ça nous sert d'apprendre à utiliser MySQL dans nos programmes?

Je vais répondre à cette question par un exemple simple.

Imaginons que vous vouliez faire un jeu dans lequel le joueur à un score final, alors vous allez faire un fichier score et y stocker le score. De cette manière, le joueur et ses amis pourront se concurrencer pour voir lequel est le meilleur. Mais, dans ce cas-là, pourquoi ne pas le faire à une plus grande échelle ?

En effet, vous pourriez enregistrer le score du joueur dans une *BDD* en utilisant l'*API MySQL*. Ainsi, ce n'est plus qu'avec ses amis que le joueur va concurrencer, mais avec tout le monde puisque tous les scores seront répertoriés dans cette *BDD*. Il vous suffira ensuite de faire une requête pour récupérer ces scores, pour que le joueur puisse voir s'il est le meilleur... ou pas.

Bien sûr, ce n'est qu'un exemple parmi tant d'autres. 

Portabilité

Utilisant les sockets de Windows, les exemples qui seront proposés dans ce tutoriel ne pourront être compatibles qu'avec Windows. Pour les utilisateurs UNIX, il vous faudra vous renseigner davantage (via ce cours sur les sockets par exemple).

Installation

Nous allons maintenant passer à l'installation de la librairie MySQL.

Dev-C++

- Allez tout d'abord dans **Aide** puis cliquez sur **A propos de Dev-C++**.
- Une fenêtre apparaît ; cliquez sur **Nouvelle version**.
- De là, sélectionnez le serveur **devpack.org comunity devpacks** puis cliquez sur **Check for update**.
- Dans **groups**, choisissez **databases** puis sélectionnez **libmysql**.
- Téléchargez-le.
- Allez dans les **options du projet**, onglet **paramètres**, case **éditeur de liens** et marquez **-lmysql**.

VC

Pour VC, il vous faut télécharger les fichiers d'en-têtes et les bibliothèques ici. Par la suite :

- placez les fichiers `.h` dans **../vc/include/mysql** ;
- placez le fichier `.lib` dans **../vc/lib** ;
- ajoutez **libmysql.lib** dans les options du projet **/editeur de lien/entrée/dependence suplementaire** ;
- ajoutez **libmysql.dll** dans le dossier du projet.

Merci à dorone pour ces informations sur VC.

Code::blocks

Tout d'abord, je vous ai préparé un petit fichier compressé avec tout ce que vous allez avoir besoin pour l'installation.

Télécharger le ZIP (766 Ko)

Vous y trouverez :

- le fichier **libmysql.dll**, que vous mettrez dans le dossier de tous vos programmes utilisant l'API MySQL (comme vous le faite habituellement pour tous vos fichiers `.dll`) ;
- le fichier **libmysqlclient.a**, que vous mettrez dans le dossier **lib** du dossier **mingw32** de Code::Blocks ;
- un dossier **MYSQL** où se trouvent tous les fichiers d'en-têtes dont vous pourrez avoir besoin. Copiez ce dossier et collez-le dans le dossier **include** du dossier **mingw32** de Code::Blocks.

Maintenant vous n'avez plus qu'à créer un nouveau projet, à ajouter le fichier `.dll` dans le répertoire de celui-ci et à lier le fichier **libmysqlclient.a**. Voilà, la bibliothèque est installée ; vous allez enfin pouvoir commencer à coder !

Linux

Pour Linux, voici la commande pour installer l'API MySQL (adaptez-la à votre distribution).

Code : Console

```
$ sudo su
# apt-get install libmysqlclient15-dev
```

Merci à Xhtml_boys pour ces informations sous Linux.

Un peu de théorie

Nous arrivons maintenant à la partie « théorique » du tutoriel. Nous allons étudier les principales fonctions de l'API MySQL pour pouvoir ensuite créer un petit jeu utilisant tout ce qu'on aura appris.

Connexion/déconnexion

Tout d'abord, créez et configurez un nouveau projet en console, en utilisant les informations de la sous-partie précédente.

Pour pouvoir utiliser l'API MySQL, il faut commencer par inclure deux fichiers d'en-têtes :

Code : C

```
#include <winsock.h>
#include <MYSQL/mysql.h>
```

Vérifiez que vous avez bien mis `#include <winsock.h>` avant `#include <MYSQL/mysql.h>`, car, dans le cas contraire, vous aurez droit à une belle petite erreur.

Maintenant nous allons faire quelque chose d'indispensable pour pouvoir faire des requêtes dans votre programme : l'**initialisation**, la **connexion** et la **déconnexion**. Tout d'abord, il va falloir que nous déclarions un objet de type `MYSQL`. Ce dernier nous servira tout au long du tutoriel, afin de pouvoir utiliser la plupart des fonctions.

Code : C

```
MYSQL mysql;
```

Puis, initialisons MySQL, grâce à la fonction :

Code : C

```
mysql_init(MYSQL *mysql);
```

.

Une fois l'initialisation réalisée, il faut se connecte. Dans cette optique, nous allons utiliser deux fonctions, que voici :

Code : C

```
mysql_options(MYSQL *mysql, enum mysql_option option, const char
*arg);
mysql_real_connect(MYSQL *mysql, const char *host, const char *user,
const char *passwd, const char *db, unsigned int port, const char
*unix_socket, unsigned long client_flag);
```

Prenons-les dans l'ordre:

- `mysql_options` :
  - Le premier argument est un pointeur de structure, que nous avons vu juste avant.
  - Le deuxième argument est l'option que nous voulons configurer, nous utiliserons toujours `MYSQL_READ_DEFAULT_GROUP`, qui lie les options spécifiées dans le fichier `my.cnf`.
  - Enfin, le troisième argument est la valeur de cette option ; vous pouvez mettre le nom que vous souhaitez.
- `mysql_real_connect` :
  - Le premier argument est toujours un pointeur vers votre structure.
  - Le deuxième argument est le nom de domaine ou l'adresse de votre hébergeur (dans mon cas, c'est www.goldzoneweb.info). S'il est marqué « localhost », je vous conseille de faire comme moi, et de mettre l'adresse de votre hébergeur car l'utilisation de « localhost » peut engendrer des erreurs.
  - Le troisième argument est votre identifiant de connexion.
  - Le quatrième argument est votre mot de passe.
  - Le cinquième argument est le nom de votre base de données.
  - Le sixième argument est le port, je vous conseille de mettre 0 pour éviter les erreurs.
  - Le septième argument est le socket à utiliser, je vous conseille de mettre `NULL` ici.
  - Et le huitième argument est le *flag*, je vous conseille de mettre 0.

Pour travailler en local (avec EasyPHP par exemple), il faut mettre **localhost** comme 2e argument, puis root en identifiant, et il faut laisser vide pour le mot de passe. (information de Arabesque356)

Certains hébergeurs bloquent le port pour la connexion, il vous est donc impossible d'utiliser l'API avec ces hébergeurs. (information de Arabesque356)

Comme vous l'aurez sûrement compris, `mysql_options` sert à spécifier des options de connexion, et `mysql_real_connect` sert à se connecter à une base de données. Pour terminer, il ne vous reste plus qu'à fermer la connexion MySQL à l'aide de la fonction :

Code : C

```
mysql_close(MYSQL *mysql);
```

.

Code : C

```
#include <stdio.h>
#include <stdlib.h>
#include <winsock.h>
#include <MYSQL/mysql.h>
```

```
int main(void)
{
    MYSQL mysql;
    mysql_init(&mysql);
    mysql_options(&mysql,MYSQL_READ_DEFAULT_GROUP,"option");


if(mysql_real_connect(&mysql,"www.goldzoneweb.info","mon_pseudo","******","ma_ba
    {
        mysql_close(&mysql);
    }
    else
    {
        printf("Une erreur s'est produite lors de la connexion à la BDD!");
    }

    return 0;
}
```

Voilà je ne crois pas avoir besoin d'expliquer ce code, si ce n'est que, lorsque la connexion plante, `mysql_real_connect` renvoie `NULL`, ce qui nous permet de gérer les erreurs.

Les requêtes n'attendant aucun jeu de résultats

Hein, c'est quoi que ça, un nouveau type de requêtes ?

Non non, rassurez-vous ! En fait, vous les connaissez déjà : **`INSERT`**, **`DELETE`**, **`CREATE`**, **`UPDATE`** par exemple n'attendent aucune réponse. Au contraire, des requêtes comme **`SELECT`** ou **`SHOW`** attendent une réponse. Voyons maintenant la fonction permettant de faire des requêtes MySQL :

Code : C

```
mysql_query(MYSQL *mysql, const char *query);
```

Il vous faudra spécifier pour cette fonction le pointeur de structures ainsi que la requête. Voici un exemple de son utilisation:

Code : C

```
mysql_query(&mysql, "INSERT INTO ma_table VALUES('valeur 1', 'valeur
2', 'etc')");
```

Comme vous pouvez le voir, cette requête va enregistrer dans la table `ma_table` les valeurs que j'ai mises dans l'exemple. Maintenant, si vous voulez savoir combien de lignes ont été affectées par votre requête, il suffit d'appeler la fonction :

Code : C

```
mysql_affected_rows(MYSQL *mysql);
```

Les requêtes attendant un jeu de résultats

Ce type de requête n'est pas plus compliqué que l'autre, si ce n'est que, comme on attend un résultat, il faut faire des opérations en plus. Pour la requête en elle même ça ne change pas, il faut toujours utiliser la fonction `mysql_query`. Cependant, pour récupérer le résultat il y a deux façons de faire, que nous allons étudier ensemble.

*mysql_use_result(MYSQL *mysql)*

Cette fonction récupère le jeu de résultat, mais ne l'enregistre pas dans le client. Son principal avantage réside dans sa rapidité et dans sa faible utilisation de la mémoire. Toutefois, vous ne pourrez pas faire d'opérations comme revenir au résultat n°2, etc. Vous devrez stocker la valeur de retour de cette fonction dans un pointeur de structure de type `MYSQL_RES`, qui est fait spécialement pour stocker le jeu de résultats.

Stocker un jeu de résultat ne suffit pas pour le lire ; pour cela, vous devrez appeler une autre fonction :

Code : C

```
mysql_fetch_row(MYSQL_RES *result);
```

La valeur de retour de cette fonction devra quant à elle être stockée dans un objet de type `MYSQL_ROW` (correspondant généralement à un tableau de chaînes de caractères) `mysql_fetch_row` stocke les valeurs de la ligne suivante dans le jeu de résultats. Il est nécessaire, de libérer la mémoire allouée par cette fonction.

Code : C

```
mysql_free_result(MYSQL_RES *result);
```

Pensez à vérifier que `mysql_use_result` renvoie quelque chose.

Voici un exemple récapitulatif.

Code : C

```
#include <stdio.h>
#include <stdlib.h>
#include <winsock.h>
#include <MYSQL/mysql.h>



int main(void)
{
    //Déclaration du pointeur de structure de type MYSQL
    MYSQL mysql;
    //Initialisation de MySQL
    mysql_init(&mysql);
    //Options de connexion
    mysql_options(&mysql,MYSQL_READ_DEFAULT_GROUP,"option");

    //Si la connexion réussie...

if(mysql_real_connect(&mysql,"www.goldzoneweb.info","mon_pseudo","*****","ma_bas
    {
        //Requête qui sélectionne tout dans ma table scores
        mysql_query(&mysql, "SELECT * FROM scores");
        //Déclaration des objets
        MYSQL_RES *result = NULL;
        MYSQL_ROW row;
```

```
            int i = 1;

            //On met le jeu de résultat dans le pointeur result
            result = mysql_use_result(&mysql);
            //Tant qu'il y a encore un résultat ...
            while ((row = mysql_fetch_row(result)))
            {
               printf("Resultat %ld\n", i);
               i++;
            }
            //Libération du jeu de résultat
            mysql_free_result(result);

            //Fermeture de MySQL
            mysql_close(&mysql);
        }
        else  //Sinon ...
        {
            printf("Une erreur s'est produite lors de la connexion à la BDD!");
        }

        return 0;
    }
```

Code : Console

```
Resultat 1
Resultat 2
Resultat 3
Resultat 4
Resultat 5
Resultat 6

Press any key to continue.
```

Le jeu de résultats est mis dans `result`, puis on fait une boucle pour lister toutes les lignes. Mais ça n'indique toujours pas la *valeur* de la ligne : deux nouvelles fonctions vont nous le permettre.

Code : C

```
mysql_num_fields(MYSQL_RES *result);
```

Elle retourne un entier non signé qui correspond au nombre de champs de la table sélectionnée. Donc, pour pouvoir avoir les valeurs de notre requête, il va falloir stocker dans une variable le nombre de champs, puis faire une boucle pour avoir chaque valeur, une par une.

Code : C

```
mysql_fetch_lengths(MYSQL_RES *result);
```

Elle retourne un tableau d'entiers non signés qui correspondent à la taille de la valeur de chaque champ du résultat. Avec ces informations, voici un exemple de requête complète ::

Code : C

```
#include <stdio.h>
#include <stdlib.h>
```

```c
#include <winsock.h>
#include <MYSQL/mysql.h>



int main(void)
{
    //Déclaration du pointeur de structure de type MYSQL
    MYSQL mysql;
    //Initialisation de MySQL
    mysql_init(&mysql);
    //Options de connexion
    mysql_options(&mysql,MYSQL_READ_DEFAULT_GROUP,"option");

    //Si la connexion réussie...

if(mysql_real_connect(&mysql,"www.goldzoneweb.info","mon_pseudo","*****","ma_bas
    {
        //Requête qui sélectionne tout dans ma table scores
        mysql_query(&mysql, "SELECT * FROM scores");

        //Déclaration des objets
        MYSQL_RES *result = NULL;
        MYSQL_ROW row;

        unsigned int i = 0;
        unsigned int num_champs = 0;

        //On met le jeu de résultat dans le pointeur result
        result = mysql_use_result(&mysql);

        //On récupère le nombre de champs
        num_champs = mysql_num_fields(result);

        //Tant qu'il y a encore un résultat ...
        while ((row = mysql_fetch_row(result)))
        {
            //On déclare un pointeur long non signé pour y stocker la taille des
            unsigned long *lengths;

            //On stocke ces tailles dans le pointeur
            lengths = mysql_fetch_lengths(result);

            //On fait une boucle pour avoir la valeur de chaque champs
            for(i = 0; i < num_champs; i++)
            {
               //On ecrit toutes les valeurs
               printf("[%.*s] ", (int) lengths[i], row[i] ? row[i] : "NULL");
            }
            printf("\n");
        }

        //Libération du jeu de résultat
        mysql_free_result(result);

        //Fermeture de MySQL
        mysql_close(&mysql);

    }
    else  //Sinon ...
    {
        printf("Une erreur s'est produite lors de la connexion à la BDD!");
    }

    return 0;
}
```

Code : Console

```
[1] [The BasheR] [130000]
[2] [Rom] [10005120]
[3] [Tib] [51203]
[4] [Joe] [78491]
[5] [Fab] [12054]
[6] [Nic] [1102350]
```

La seule ligne qui peut vous poser problème est la suivante :

Code : C

```
printf("[%.*s] ", (int) lengths[i], row[i] ? row[i] : "NULL");
```

D'abord, la chaîne de conversion de `printf`, `%.*s`, est assez austère, il faut l'avouer. Cependant, en prenant les éléments un par un, on arrive à la comprendre sans problème. Tout d'abord, le `%` indique le début d'un format. Ensuite vient le point, qui sert à séparer un nombre éventuel indiquant la largeur du champ et la précision désirés. Par exemple, avec `%10.20s`, on aura une largeur de champ de 10 et on écrira au maximum 20 caractères. Enfin, on trouve l'étoile, qui permet, en réalité, de spécifier la largeur ou la précision à l'aide d'une variable du programme. Celle-ci doit être de type `int` (d'où le transtypage).

Ainsi, l'appel à `printf` suivant (tiré du K&R) signifiera : « imprimer sur `stdout` au plus `max` caractères de la chaîne `s` ».

Code : C

```
printf("%.*s", max, s);
```

Ensuite, la condition ternaire. Voici son équivalent sous forme classique :

Code : C

```
if (row[i])
    printf("%.*s", (int) lengths[i], row[i]);
else
    printf("NULL");
```

Aussi la syntaxe est-elle la suivante :

Code : Autre

```
(condition) ? (a_faire_si_vrai) : (a_faire_si_faux)
```

Merci à mleg pour l'explication de ce point.

*mysql_store_result(MYSQL *mysql)*

Le principe est le même, à la différence que cette fonction stocke le jeu de résultat dans une mémoire tampon, ce qui rend

possible des éventuelles opérations à partir de ce jeu. Notez néanmoins que la fonction peut se révéler plus lente que `mysql_use_result`.

Pour réaliser une requête similaire à celle de l'exemple précédent il suffit juste de changer `mysql_use_result` par `mysql_store_result`.

Code : C

```c
#include <stdio.h>
#include <stdlib.h>
#include <winsock.h>
#include <MYSQL/mysql.h>



int main(void)
{
    //Déclaration du pointeur de structure de type MYSQL
    MYSQL mysql;
    //Initialisation de MySQL
    mysql_init(&mysql);
    //Options de connexion
    mysql_options(&mysql,MYSQL_READ_DEFAULT_GROUP,"option");

    //Si la connexion réussie...

if(mysql_real_connect(&mysql,"www.goldzoneweb.info","mon_pseudo","*****","ma_bas
    {
        //Requête qui sélectionne tout dans ma table scores
        mysql_query(&mysql, "SELECT * FROM scores");

        //Déclaration des objets
        MYSQL_RES *result = NULL;
        MYSQL_ROW row;

        unsigned int i = 0;
        unsigned int num_champs = 0;

        //On met le jeu de résultat dans le pointeur result (maintenant on util
mysql_store_result
        result = mysql_store_result(&mysql);

        //On récupère le nombre de champs
        num_champs = mysql_num_fields(result);

        //Tant qu'il y a encore un résultat ...
        while ((row = mysql_fetch_row(result)))
        {
            //On déclare un pointeur long non signé pour y stocker la taille des
            unsigned long *lengths;

            //On stocke cette taille dans le pointeur
            lengths = mysql_fetch_lengths(result);

            //On fait une boucle pour avoir la valeur de chaque champs
            for(i = 0; i < num_champs; i++)
            {
                //On ecrit toutes les valeurs
                printf("[%.*s] ", (int) lengths[i], row[i] ? row[i] : "NULL");
            }
            printf("\n");
        }

        //Libération du jeu de résultat
        mysql_free_result(result);

        //Fermeture de MySQL
        mysql_close(&mysql);
```

```
        }
        else  //Sinon ...
        {
            printf("Une erreur s'est produite lors de la connexion à la BDD!");
        }

        return 0;
    }
```

Mais alors pourquoi utiliser cette fonction si elle fait la même chose, en plus de temps ? 

Eh bien, c'est vrai qu'à première vue ça ne sert pas à grand chose, mais en fait l'intérêt de cette fonction est qu'on peut « voyager » dans le jeu de résultat, c'est-à-dire qu'on peut demander à avoir seulement le résultat n°2, ou n°3, etc. Pour cela, il existe une fonction que voici :

Code : C

```
mysql_data_seek(MYSQL_RES *result, my_ulonglong offset);
```

Elle prend en argument le jeu de résultat ainsi que le numéro de la ligne qu'on veut obtenir.

Code : C

```
//Requête qui sélectionne tout dans ma table scores
            mysql_query(&mysql, "SELECT * FROM livreor");

            //Déclaration des variables
            MYSQL_RES *result = NULL;
            MYSQL_ROW row;

            unsigned int i = 0;
            unsigned int num_champs = 0;

            //On met le jeu de résultats dans le pointeur result.
            result = mysql_store_result(&mysql);

            //On choisit une ligne.
            mysql_data_seek(result, 4);

            //On récupère le nombre de champs
            num_champs = mysql_num_fields(result);

            //On stocke les valeurs de la ligne choisie.
            row = mysql_fetch_row(result);

            //On déclare un pointeur long non signé pour y stocker
la taille des valeurs.
            unsigned long *lengths;

            //On stocke cette taille dans le pointeur.
            lengths = mysql_fetch_lengths(result);

            //On fait une boucle pour avoir la valeur de chaque
champ.
            for(i = 0; i < num_champs; i++)
            {
                //On ecrit toutes les valeurs
                printf("[%.*s] ", (int) lengths[i], row[i] ? row[i]
: "NULL");
```

```
                    }
```

Code : Console

```
[4] [Joe] [78491]
```

Voilà, nous en avons donc terminé avec la théorie, nous allons maintenant passer à la pratique dans un petit TP, que de réjouissances en perspective. 

La pratique

Nous y voilà enfin ! Après toute cette attente nous allons enfin pouvoir pratiquer !  Ce petit TP ne sera pas bien compliqué et utilisera ce que vous venez d'apprendre.

Concept

Le concept de ce jeu est assez simple, nous allons reprendre le jeu du plus ou moins en console, et y apporter quelques modifications. Tout d'abord, il va nous falloir une table pour stocker les scores, dont voici le code :

Code : SQL

```
CREATE TABLE `scores` (
   `id` int(11) NOT NULL AUTO_INCREMENT,
   `pseudo` varchar(20) NOT NULL,
   `score` int(11) NOT NULL,
  KEY `id` (`id`)
);
```

Au début, il y aura un menu demandant si on jouer ou voir les scores.

Jouer

Si le joueur choisit de jouer, alors on entre dans le jeu du plus ou moins : un nombre est généré aléatoirement, l'utilisateur doit essayer de le trouver. Une fois cela fait, on indique au joueur le nombre de coups qu'il a fait et on lui demande son pseudo. Il suffit ensuite d'enregistrer le tout dans la table « scores ».

Scores

Pour afficher les scores, nous utiliserons le format suivant, avec les scores dans l'ordre croissant :

Code : Autre

```
1- pseudo1 score1 pts

2- pseudo2 score2 pts

...

...
```

Correction

Secret (cliquez pour afficher)

Code : C

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <winsock.h>
#include <MYSQL/mysql.h>



int main(void)
{
    long nombreMystere = 0, nombreEntre = 0, choix = 0;
    const long MAX = 100, MIN = 1;
    char pseudo[20] = "";

    // Génération du nombre aléatoire

    srand(time(NULL));

    //Choix de voir les scores ou de jouer
    printf("Bienvenue dans mon jeu de plus ou moins!\n\n");

    printf("1- Jouer\n");
    printf("2- Scores\n\n");

    scanf("%ld", &choix);


    switch (choix)
    {
        //Si on veut jouer
        case 1:
            //On génére le nombre aléatoire
            nombreMystere = (rand() % (MAX - MIN + 1)) + MIN;

            int i = 1;

            do
            {
                // On demande le nombre
                printf("Quel est le nombre ? ");
                scanf("%ld", &nombreEntre);

                // On compare le nombre entré avec le nombre mystère

                if (nombreMystere > nombreEntre)
                    printf("C'est plus !\n\n");
                else if (nombreMystere < nombreEntre)
                    printf("C'est moins !\n\n");
                else
                    printf ("Bravo, vous avez trouve le nombre mystere !!!\n

                i++;
            } while (nombreEntre != nombreMystere);

            printf("Votre score est de: %ld\n\n", i);
            printf("Veuillez entrer votre pseudo\n");
            scanf("%s", pseudo);

            //Déclaration de l'objet de type MYSQL
            MYSQL mysql;
            //Initialisation de MySQL
            mysql_init(&mysql);
```

```
                //Options de connexion
                mysql_options(&mysql,MYSQL_READ_DEFAULT_GROUP,"option");

                //Si la connexion réussie...

    if(mysql_real_connect(&mysql,"www.votrehebergeur.com","mon_pseudo","*****","
                {
                    //On déclare un tableau de char pour y stocker la requete
                    char requete[150] = "";

                    //On stock la requete dans notre tableau de char
                    sprintf(requete, "INSERT INTO scores VALUES('', '%s', '%ld')


                    //On execute la requete
                    mysql_query(&mysql, requete);


                    //Fermeture de MySQL
                    mysql_close(&mysql);
                }
                else
                {
                    printf("Une erreur s'est produite lors de la connexion a la
                }


                break;

            //On veut voir les scores
            case 2:
                printf("Liste des scores du jeu: \n\n");

                //Déclaration de l'objet de type MYSQL
                MYSQL mysql2;
                    //Initialisation de MySQL
                    mysql_init(&mysql2);
                    //Options de connexion
                    mysql_options(&mysql2,MYSQL_READ_DEFAULT_GROUP,"option2");

                    //Si la connexion réussie...

    if(mysql_real_connect(&mysql2,"www.votrehebergeur.com","mon_pseudo","*****",
                    {
                        //Requête qui sélectionne tout dans ma table scores
                        mysql_query(&mysql2, "SELECT * FROM scores ORDER BY scor

                        //Déclaration des pointeurs de structure
                        MYSQL_RES *result = NULL;
                        MYSQL_ROW row;

                        unsigned int i = 0;
                        unsigned int num_champs = 0;
                        int j =1;

                        //On met le jeu de résultat dans le pointeur result
                        result = mysql_use_result(&mysql2);

                        //On récupère le nombre de champs
                        num_champs = mysql_num_fields(result);

                        //on stock les valeurs de la ligne choisie
                        while ((row = mysql_fetch_row(result)))
                        {

                            //On déclare un pointeur long non signé pour y stock
    valeurs
                            unsigned long *lengths;

                            //On stocke ces tailles dans le pointeur
```

```
                            lengths = mysql_fetch_lengths(result);

                            //On affiche la position du joueur
                            printf("%ld- ", j);

                            //On fait une boucle pour avoir la valeur de chaque
                           for(i = 1; i < num_champs; i++)
                           {

                               //On ecrit toutes les valeurs
                               printf("%.*s ", (int) lengths[i], row[i] ? row[i]
                           }
                           printf("pts\n");

                           //On incrémente la position du joueur
                           j++;

                       }

                       //Libération du jeu de résultat
                       mysql_free_result(result);


                       //Fermeture de MySQL
                       mysql_close(&mysql2);

                   }
                   else  //Sinon ...
                   {
                       printf("Une erreur s'est produite lors de la connexion a
                   }

               break;

           //Sinon
           default:
               printf("Le choix que vous avez entre n'est pas valide!");
               break;


       }



       return 0;
   }
```

Q.C.M.

Le premier QCM de ce cours vous est offert en libre accès.
Pour accéder aux suivants

Connectez-vous Inscrivez-vous

Quel est le bon ordre?

- ◯ initialisation/connexion/options/fermeture
- ◯ initialisation/options/connexion/fermeture
- ◯ fermeture/connexion/initialisation/options
- ◯ connexion/options/initialisation/fermeture

Quelle structure permet de stocker le résultat d'une ligne?

- ◯ MYSQL_ROW
- ◯ MYSQL
- ◯ MYSQL_RES

Quelles sont les différences entre mysql_use_result et mysql_store_result?

- ◯ mysql_use_result est plus rapide mais ne permet pas de faire des opérations sur le jeu de résultats
- ◯ mysql_use_result est plus lent et ne permet pas les opérations sur le jeu de résultats
- ◯ mysql_use_result est plus lent mais permet les opérations sur le jeu de résultats
- ◯ mysq_use_result est plus rapide et permet les opérations sur le jeu de résultats

A quoi sert mysql_data_seek?

- ◯ A bouger le curseur dans les différents champs
- ◯ A choisir la ligne que l'on veut dans le jeu de résultats
- ◯ A rendre les données inutilisables

Correction !

Statistiques de réponses au QCM

Voilà, ce cours sur l'*API MySQL* est maintenant fini, j'espère qu'il a été assez clair et que vous avez bien tout compris.

Voici quelques idées d'amélioration du mini-jeu que l'on vient de faire, que l'on réaliser au niveau de MySQL :

- limiter le nombre de scores à afficher à 10 pour éviter que ça prenne trop de place ;
- prévenir le joueur que l'enregistrement de son score s'est bien déroulé.

De plus, vous pouvez laisser libre cours à votre imagination. 

Nous pouvons donc nous quitter maintenant (j'en vois qui se réjouissent dans le fond ! ). Si vous voulez en apprendre encore plus sur cette bibliothèque, je vous conseille cette page : c'est la liste de toutes les fonctions de l'*API MySQL* avec leur description, en français !

Sur ce, au revoir tout le monde, et à bientôt pour de nouvelles aventures !