# Introduction à la bibliothèque MySQLTcl

par Diego Alberto Arias Prad
<dariapra(at)yahoo.es>

*L´auteur:*

Je suis ingénieur en télécommunications et je vis à Lugo en Espagne. Je ne me souviens plus très bien quand j'ai utilisé Linux pour la première fois, en 1995 ou 1996 ? Avant, je ne connaissais même pas l'existence de Linux et m'en tenais à Windows. J'ai vu fonctionner Linux pour la première fois sur un ordinateur à l'université : ça a tout de suite éveillé mon attention, et très vite, je voulus l'installer sur mon propre PC à domicile. Je me souviens avoir débuté avec une Slackware.

Au cours des années, j'ai fait le tour de nombre de distributions Linux ainsi que de quelques moutures BSD, explorant les langages Java ou Tcl, les bases de données; et différents serveurs... Mais je n'utilise pas Linux que pour le "fun", j'ai eu aussi la chance de travailler sous Linux losrque j'étais chez

*Résumé*:

Nous verrons dans cet article comment installer et utiliser la bibliothèque MySQLTcl : une bibliothèque permettant d'établir des requêtes (SELECT, INSERT, DELETE...) vers une base de données MySQL dans un script Tcl. Dans le cadre de cet article, les versions respectives de Tcl, MySQL et MySQLTcl sont 8.4.2, 4.0.15 et 2.40.

Tcl signifie Tool Command Language (littéralement "langage de commande d'outil"), il a été mis au point par John Ousterhout [1]. Il est en fait constitué de deux parties : un langage de script et un interpréteur pour ce même langage. à part ça, il s'agit d'un langage de programmation structuré qui utilise trois types de données : les chaînes de caractères, les listes et les tableaux (string, list, array). Et entre autres caractéristiques, il faut souligner le support des expressions régulières, nombre de bibliothèques d'extension tierces, Tk ou encore un utilitaire pour développer des interfaces graphiques en Tcl...

MySQL est un serveur de bases de données très populaire dans la communauté du libre : est il encore nécessaire d'en faire la présentation ?

MySQLTcl est la bibliothèque Tcl permettant comme annoncé plus haut d'établir des requêtes à un serveur de bases de donnés MySQL dans un script Tcl. Les développeurs et mainteneurs actuels de cette bibliothèque sont Paolo Brutti (Paolo.Bruti at tlsoft.it), Tobias Ritzau (tobri at ida.liu.se) et Artur Trzewick (mail at xdobry.de).

Telefónica I+D.

*Traduit en Français par:*
Guillaume Baudot
<guillaume(at)linuxfocus.org>

# Installation de la bibliothèque MySQLTcl

Si votre distribution le permet, vous pouvez l'installer avec le paquetage (RPM, DEB...) ou le système de ports (Crux Linux ou FreeBSD, par exemple, ou encore Gentoo Linux) propre à votre distro et passer ce chapitre.

Mais dans le cas contraire, voire pous satisfaire votre curiosité, je vais vous présenter ici comment j'ai procédé pour l'installation manuelle (procédure qu'il faut plus voir comme un guide général que prendre à la lettre !). Avec la Linux Mandrake (version 9.2), dans un terminal sous bash :

```
$ ./configure --with-mysql-lib=/usr/lib
$ make
$ make install
```

S'il survient un problème durant la phase "configure", le message d'erreur devrait vous donner un indice : le plus souvent, le script ne parvient pas à localiser certains répertoires ou fichiers, auquel cas vous pouvez en indiquer le chemin en ajoutant au script le paramètre idoine. Comme un bon exemple vaut souvent mieux qu'un long discours, voyez les commandes que j'ai utilisées avec FreeBSD 5.0 :

```
$ export CPP=/usr/bin/cpp
$ ./configure --with-tcl=/usr/lib/local/tcl8.3
 --with-tclinclude=/usr/local/include/tcl8.3
 --with-mysql-include=/usr/local/include/mysql
 --with-mysql-lib=/usr/local/lib/mysql
$ make
$ make install
```

Comme vous aurez pu le remarquer, j'ai utilisé ici la version 8.3 de Tcl. Et pour information, les versions de MySQL et MySQLTcl étaient respectivement 3.23.54 et 2.15.

# Les notions fondamentales de Tcl

Dans ce chapitre, je vais vous présenter quelques notions basiques de Tcl, indispensables aux non initiés pour suivre cet article. Si vous programmez vous−même en Tcl, inutile de préciser que vous n'aurez pas grand chose à en apprendre.

À partir de maintenant, les exemples à venir sont des commandes pour le shell Tcl (tclsh).

## Les variables : commandes et substitution de variables

Les variables Tcl sont definies avec la commande *set*, comme suit :

```
% set address {Edison Avenue, 83}
Edison Avenue, 83
% set zip_code 48631
48631
```
Nous avons successivement créé ici deux variables de type chaîne, **address** et **zip_code** avec pour valeurs respectives *Edison Avenue, 83* et *48361*. Pour la définition d'**address**, j'ai placé la chaîne entre accolades, car elle contient des espaces. On utilisera la même commande pour récupérer la valeur d'un variable :

```
% set address
Edison Avenue, 83
% set zip_code
48631
```
Supposons maintenant que nous voulions afficher la valeur d'une variable à l'écran, nous utiliserons alors la commande *puts* :

```
% puts stdout [set address]
Edison Avenue, 83
```
Le paramètre **stdout** indique à la commande *puts* que l'affichage se fait sur la "sortie standard", en l'occurence l'écran. Le second paramètre, **[set address]**, correspond à ce qui sera affiché. Vous aurez certainement remarqué les crochets : ils sont là pour indiquer à l'interpréteur Tcl que le second argument est une commande Tcl qui doît être éxécutée avant l'instruction *puts*. C'est ce qu'on appelle communément un subsitution de variable. Il existe une autre méthode de substitution, que voici :

```
% puts stdout $address
Edison Avenue, 83
```
Cette commande a le même effet que la précédente, autrement dit le caractère *$* suivi d'un nom de variable indique aussi une subsitution de variable.

Nous avons vu précédemment comment initialiser une chaîne de caractères contenant des espaces, à savoir avec des accolades. Il est tout à fait possible aussi d'utiliser des guillemets. Toutefois, il est important de noter que ces deux possibiltés ne sont pas totalement équivalentes, comme vous allez pouvoir le constater :

```
% puts stdout "the zip code is [set address]"
the zip code is 48631
% puts stdout "the zip code is $address"
the zip code is 48631
% puts stdout {the zip code is [set address]}
the zip code is [set address]
% puts stdout {the zip code is $address}
the zip code is $address
```
La subsitution de variable a lieu dans une chaîne entre guillemets, alors qu'une chaîne entre accolades est interprétée telle quelle.

Enfin, on supprime une variable avec la commande *unset* :

```
% unset address
% set address
can't read "address": no such variable
% unset zip_code
% set zip_code
can't read "zip_code": no such variable
```

## Les chaînes de caractères en Tcl

Les chaînes de caractères sont un des trois types basiques définis dans Tcl. On crée une variable de type chaîne avec la commande *set*.

```
% set surname Westmoreland
Westmoreland
% set number 46.625
46.625
```

Nous avons défini ici deux variables de type chaîne : *surname* et *number*. Pour manipuler ce type de variables, nous disposons de la commande *string* dont la syntaxe est la suivante :
*string operation chaine [arguments_supplementaires]*
Une fois encore, une exemple devrait être plus parlant :

```
% string length $surname
12
% set surname [string range $surname 0 3]
West
% puts stdout $surname
West
% string tolower $surname
west
```

Contrairement à Java, Pascal ou C, Tcl n'est pas un langage fortement typé, comme vous pourrez le constater dans l'exemple suivant :

```
% set number [expr $number + 24.5]
70.125
% string range $number 2 5
.125
```

Avec la commande *expr*, on ajoute 24,5 à la valeur de **number** qui est alors traitée comme un nombre, tandis qu'avec la commande *string*, la même variable est considérée comme une chaîne et ses 4 derniers caractères sont retournés.

Il existe bien entendu de nombreuses autres fonctions de manipulation de chaînes, mais nous en savons déjà bien assez pour les besoins de cet article.

## Les listes en Tcl

Les listes Tcl sont un type spécial de chaîne dans laquelle les éléments, séparés par des espaces, donnent lieu à une interprétation

particulière.

```
% set friends_list {Fany John Lisa Jack Michael}
Fany John Lisa Jack Michael
% set friends_list [string tolower $friends_list]
fany john lisa jack michael
% set friends_list
fany john lisa jack michael
```

Un certain nombre de commandes sont à notre disposition pour manipuler les listes :

```
% lindex 2 $friends_list
lisa
```

```
% lrange $friends_list 2 4
lisa jack michael
% set friends_list [lsort -ascii $friends_list]
fany jack john lisa michael
% set friends_list
fany jack john lisa michael
% set friends_list [linsert $friends_list 2 Peter]
fany jack Peter john lisa michael
% string toupper $friends_list
FANY JACK PETER JOHN LISA MICHAEL
```

La dernière commande en particulier nous démontre que rien ne distingue les types liste et chaîne.

## Les tableaux en Tcl

Un tableau peut être considéré comme une liste indexée où les index sont eux−mêmes des chaînes. On créera un tableau comme suit :

```
% set phone_numbers(fany) 629
629
% set phone_numbers(michael) 513
513
% set phone_numbers(john) 286
286
```

Les valeurs d'un tableau se récupèrent avec la commande *set* et en utilisant la subsitution de variable. NB : les valeurs entre parenthèses sont les index.

```
% set $phone_numers(michael)
513
```

La commande *array* renvoie des informations sur une variable de type tableau, comme l'illustre cet exemple :

```
% array size phone_numbers
3
% array names phone_numbers
fany john michael
```

# Connexion à la base de données

Avant toute requête, il est indispensable d'établir une connection au serveur de bases de données. Nous allons voir ici comment faire, et comment gérer les erreurs susceptible d'advenir.

## Établissement de la connexion

Voici un exemple sommaire de script Tcl qui ouvre une connexion à un serveur de bases de données MySQL, puis la ferme aussitôt.

```
0:  #!/usr/bin/tclsh8.4
1:
2:  package require mysqltcl
3:  global mysqlstatus
4:
```

```
5:  set port {3306}
6:  set host {127.0.0.1}
7:  set user {john_smith}
8:  set password {jsmith_password}
9:
10: set mysql_handler [mysqlconnect -host $host
    -port $port -user $user -password $password]
11:
12: mysqlclose $mysql_handler
```

Attention, les numéros de ligne (suivis de leurs deux−points) ont été ajoutés par mes soins, il ne faut pas les reproduire dans vos scripts. En outre, vous aurez probablement à modifier la ligne #0 pour indiquer une chemin valide vers le shell Tcl présent sur votre machine.

La ligne #0 indique que le fichier est un script Tcl et précise le chemin vers l'interpréteur idoine.

La ligne 2 informe l'interpréteur qu'il doit charger la bibliothèque MySQLTcl pour éxécuter les commandes de ce script. Par exemple, on peut voir en ligne #10 un appel à la commande *mysqlconnect* : il va sans dire que sans notre ligne #2, l'interpréteur renverrait une erreur du genre "commande inconnue".

Les lignes #5 et #6 indiquent le numéro de port et l'adresse de l'hôte du serveur MySQL auquel on veut se connecter. 3306 est le port par défaut pour MySQL, et l'hôte est 127.0.0.1 : le serveur MySQL et le script Tcl se trouvent sur la même machine.

Lignes #7 et #8, indiquez les "login" et "mot de passe" d'un compte valide sur le serveur MySQL.

C'est à la ligne #10 qu'on établit la connexion. Vous remarquerez que je stocke la valeur de retour de la commande *mysqlconnect* dans une variable, **mysql_handler**. C'est cette valeur qui nous permet ensuite d'identifier la connexion établie, que ce soit pour soumettre des requêtes ou pour fermer la connexion, comme le montre la ligne #12.

## Gérer les erreurs

L'éxécution de commandes de la bibliothèque MySQLTcl est susceptible de provoquer une erreur (comme celle de toute commande au demeurant). Lorsqu'une erreur survient, si elle n'est pas prise en charge par le script, ce dernier s'interrompt prématurément. Aussi, pour introduire la gestion d'erreur, je vous propose cette variante du premier script :

```
10: if [catch {mysqlconnect -host $host -port $port
    -user $user -password $password} mysql_handler] {
11:     puts stderr {error, the database connection could not be established}
12:     exit
13: } else {
14:     mysqlclose mysql_handler
15: }
```

La commande *catch* éxécute la commande entre accolades (**set mysql_handler [mysqlconnect −host $host...**) et stocke son résultat dans **mysql_handler**. La commande *catch* renvoie 0 quand tout va bien et 1 si l'éxécution de la commande passée en premier paramètre génère une erreur.

Dans cet exemple, si la commande *mysqlconnect...* échoue, on affiche un message sur la sortie d'erreur standard (stderr), normalement l'écran. De nombreuses raisons peuvent être à l'origine d'une erreur de connexion à une base MySQL : mot de passe erroné, hôte invalide, mauvais numéro de port. Vous vous doutez bien qu'il va rapidement être question de messages d'erreur un peu plus éloquents qu'un simple "erreur, la connexion à la bases de données n'a pu se faire".

J'avais soigneusement évité jusqu'ici d'évoquer la ligne #3, mais il est temps d'y venir. On y déclare une variable globale, *mysqlstatus* : une variable globale est une variable qui est accessible depuis n'importe quelle partie d'un script Tcl, par opposition à une variable locale qui n'est visible que là où elle est déclarée (dans une fonction, une bibliothèque...). Ces notions de portée des variables dépassant largement le cadre de cet article, vous retiendrez simplement que la bibliothèque MySQLTcl définit (et utilise) une variable globale nommée *mysqlstatus*, qui est un tableau constitué des éléments suivants :

| élément | signification |
|---|---|
| code | La valeur de *mysqlstatus(code)* est 0 s'il n'y a pas eu d'erreur. Et en cas d'erreur, elle est égale au code d'erreur du serveur MySQL. Enfin, si l'erreur est indépendante du serveur, la valeur est −1.<br><br>La valeur de *mysqlstatus(code)* est mise à jour après toute éxécution d'une commande de la bibliothèque MySQLTcl. |
| command | La dernière commande de la bibliothèque MySQLTcl qui ait provoqué une erreur est enregistrée dans *mysqlstatus(command)*.<br><br>Attention, cette valeur est mise à jour **uniquement** lorsqu'une erreur est survenue ! |
| message | Comme pour *mysqlstatus(command)*, cette valeur n'est modifiée qu'en cas d'erreur. Vous trouverez dans *mysqlstatus(message)* le message d'erreur associé à la dernière commande infructueuse. |

Il existe encore un élément du tableau *mysqlstatus*, mais qui est sans rapport avec la gestion d'erreurs.

| élément | signification |
|---|---|
| nullvalue | *mysqlstatus(nullvalue)* est utilisé pour stocker la chaîne qui représente la valeur NULL dans le résultat de requêtes SQL. Par défaut, c'est une chaîne vide, mais vous pouvez à loisir utiliser n'importe quelle chaîne. |

Ainsi, pour une gestion d'erreurs plus efficace et avec l'aide du tableau *mysqlstatus*, je vous propose cette autre variante du script de connexion :

```
10: catch {mysqlconnect -host $host -port $port
    -user $user -password $password} mysql_handler
11: if {$mysqltatus(code) != 0} {
12:     puts stderr $mysqlstatus(message)
13: } else {
14:     mysqlclose mysql_handler
15: }
```

# Les commandes essentielles de la bibliothèque MySQLTcl

Dans ce dernier chapitre, nous allons voir les quelques commandes les plus indispensables de la bibliothèque MySQLTcl, ainsi que des exemples d'utilisation. Pour une liste détaillée, reportez vous à la page de manuel.

Les commandes que nous aborderons ici sont listées dans le tableau ci−dessous : pour légende, les paramètres sont soulignés, un paramètre entre deux points d'interrrogation est optionnel et la barre verticale "|" signifie "ou".

| commande | description succinte |
|---|---|
| mysqlconnect ?option value ...? | |

| | |
|---|---|
| | Établit la connexion à la base de données ; renvoie un identifiant de connexion qui sera par la suite utilisé par les autres commandes. |
| mysqluse handle dbname | Associe l'identifiant de connexion avec la base spécifiée en second paramètre (en gros, sélectionne la base voulue). |
| mysqsel handle sql_statement ?–list | –flatlist? | Soumet une requête SQL de type **SELECT** à la base de données. |
| mysqlexec handle sql_statement | Soumet une requête d'un type autre que **SELECT** (soit INSERT, DELETE, UPDATE...) à la base. |
| mysqlclose handle | Referme la connexion à la base. |

## mysqlconnect

Nous avons déjà abordé cette commande un peu plus haut, mais j'y reviens pour préciser qu'elle accepte un paramètre supplémentaire : **–db**. Ce dernier permet de fixer à l'établissement de la connexion la base de données vers laquelle se feront les requêtes, comme vous pouvez le voir dans cet exemple :

```
% set mysql_handler [mysqlconnect -h 127.0.0.1 -p 3306 \\
  -user john_smith -password jsmith_password -db jsmith_database]
```

La base de données ciblée est nommée *jsmith_database* : toute commande utilisant *mysql_handler* sera destinée à cette même base.

Attention : les "antislash" ne devraient pas apparaître dans la commande, je les ai mis pour indiquer que c'est la même commande qui continue sur la ligne suivante.

## mysqluse

Cette commande sert à changer la base de données associée à l'identifiant de connexion.

## mysqlsel

Cette commande envoie à la base une requête de type **SELECT**, elle provoquera une erreur pour tout autre type de requête.

Un troisième paramètre peut être passé à la commande : *list* ou *flat_list*. Pour comprendre l'incidence de ce paramètre sur le résultat obtenu, je vous propose un nouvel exemple. Supposons que nous disposons dans notre base de données d'une table nommée *people* avec un contenu similaire à celui de ce tableau :

| id | first_name | last_ name | phone |
|---|---|---|---|
| 26 | Karl | Bauer | 8245 |
| 47 | James | Brooks | 1093 |
| 61 | Roberto | Castro Portela | 6507 |

Nous allons utiliser *mysqlsel* pour notre requête :

```
% mysqlsel $mysql_handler {select first_name, last_name from people order by id asc} -list
```

```
{Karl Bauer} {James Brooks} {Roberto {Castro Portela}}
% mysqlsel $mysql_handler {select first_name, last_name from people order by id asc} -flatlist
Karl Bauer James Brooks Roberto {Castro Portela}
```

Dans le premier cas (paramètre *−list*), la commande renvoie une liste dont tous les éléments sont eux−mêmes des listes. Et dans le second (paramètre *−flat_list*), la commande retourne une liste simple où les éléments ont été concaténés.

Mais alors, qu'advient−il quand on utilise la commande sans le troisième paramètre ? Et bien la commande renvoie simplement le nombre de lignes retournées par la requête (le nombre d'éléments, si vous préférez).

```
% mysqlsel $mysql_handler {select first_name, last_name from people order by id asc}
3
```

## mysqlexec

Cette commande soumet une requête d'un autre type que **SELECT** à la base de données. Si la requête est de type **SELECT**, cela ne provoque pas d'erreur : elle sera simplement ignorée.

Pour illustration, basons notre exemple sur la même table que précédemment, à savoir *people*.

```
% mysqlexec $mysql_handler {delete from people where id=26}
1
% mysqlsel $mysql_handler {select first_name, last_name, phone from people order by id asc} -list
{James Brooks 1093} {Roberto {Castro Portela} 6507}
% mysqlexec $mysql_handler \\
  {insert into people (id, first_name, last_name, phone) values (58, "Carla", "di Bella", 8925)}
1
% mysqlsel $mysql_handler {select first_name, last_name, phone from people order by id asc} -list
{James Brooks 1093} {Carla {di Bella} 8925} {Roberto {Castro Portela} 6507}
```

L'on peut bien sûr utiliser d'autres requêtes que **INSERT** ou **DELETE**. Ainsi, pour mettre à jour une entrée de la table :

```
% mysqlexec $mysql_handler {update people set phone=3749 where first_name="James"}
1
% mysqlsel $mysql_handler {select first_name, last_name, phone from people order by id asc} -list
{James Brooks 3749} {Carla {di Bella} 8925} {Roberto {Castro Portela} 6507}
```

Apprenez enfin que, éxécutée ave succès, cette commande renvoie le nombre de lignes affectées par la requête.

## mysqlclose

Comme nous l'avons vu auparavant, la commande *mysqlclose* sert à clore la connexion au serveur MySQL : elle prend paramétre un identifiant de connexion (ce sera le plus souvent l'identifiant obtenu par *mysqlconnect* au début du même script).

## Les autres commandes de la bibliothèque MySQLTcl

Je ne saurais conclure cet article sans vous rappeler que la bibliothèque propose un jeu de commande largement plus étendu que les cinq dont je vous ai fait part. Elles permettent entre autres d'obtenir des

informations sur les bases, de modifier des chaînes de caractères (par exemple pour insérer des caractères d'échappement, comme "\"...), ou encore faire des requêtes imbriquées... L'une des meilleures sources d'information à ce sujet est la page de manuel, qui fait partie intégrante de l'installation de MySQLTcl.

## Liens utiles

[1] Petite biographie de John K. Ousterhout, l'auteur de Tcl :
http://www.softpanorama.org/People/Ousterhout/index.shtml

[2] Tutoriel sur les expressions régulières en Tcl :
http://www.mapfree.com/sbf/tcl/book/select/Html/7.html

TclTutor est une application interactive et libre pour apprendre Tcl :
http://www.msen.com/~clif/TclTutor.html

La documentation de référence de MySQL, disponible sous divers formats (html, pdf...) :
http://www.mysql.com/documentation/index.html

---