Classe c_mySQL4WD()
Classe c_postgreSQL4WD()
————————
Interface de programmation

Rodolphe Jouannet

02/01/2004
Version 1.5.5.5 (MySQL)
Version 1.0.0.0 (PostgreSQL)

Table des matières

1 Conventions

Ce document utilise un certain nombre de conventions typographiques afin de permettre une meilleure compréhension des membres, méthodes et paramètres à utiliser dans la classe à étudier. Ces conventions sont les suivantes :

- **Mot en caractère gras** : nom d'un membre ou d'une méthode de la classe,
- *Mot en italique* : paramètres à passer à une méthode,
- *[Mot en italique balisé par des crochets]* : paramètres optionels à passer à la méthode.
- `Mot en police fixe` : exemple de code.

2 Introduction

La classe c_mySQL4WD() permet un acces natif à une base MySQL (version 3.22.24 ou ultérieure) depuis une application Windev 5.5 ou Windev 7. Elle est composée de deux éléments : une librairie (DLL) développée en C et une classe Windev. Son utilisation nécessite deux librairies windows : « mySQL4WD.DLL » (fournie par ce projet) et « LIBMYSQL.DLL » (fournie par mySQL).

La classe c_postgreSQL4WD() permet un acces natif à une base PostgreSQL (version 7.2 ou ultérieure - pas testée sur des version plus ancienne) depuis une application Windev 7. Elle est composée de deux éléments : une librairie (DLL) développée en C et une classe Windev. Son utilisation nécessite deux librairies windows : « postgreSQL4WD.DLL » et « LIBPQ.DLL » (fournies par ce projet).

ATTENTION : cette version hérite de la classe c_log4WD(). Vous devez donc obligatoirement installer cette classe.

3 Membres

3.1 c_mySQL4WD :mySQLEnDehors

Permet de savoir si la fin de la sélection est atteinte ou non.

3.2 c_mySQL4WD :CLIENT_NORMAL

Utiliser par la méthode **mySQLConnecte()**. C'est le mode de connection par défaut.
Ce membre est une constante, donc passée entre parenthèse (passe par valeur).

3.3 c_mySQL4WD :CLIENT_COMPRESS

Utiliser par la méthode **mySQLConnecte()** afin de permettre la compression des données entre le client et le serveur lors de l'utilisation d'une ligne à faible débit.
Ce membre est une constante, donc passée entre parenthèse (passe par valeur).

3.4 c_mySQL4WD :mySQLDebut

Utiliser par la méthode **mySQLTransaction()** afin de débuter une transaction.
Ce membre est une constante, donc passée entre parenthèse (passe par valeur).

3.5 c_mySQL4WD :mySQLFin

Utiliser par la méthode **mySQLTransaction()** afin de clore une transaction.
Ce membre est une constante, donc passée entre parenthèse (passe par valeur).

3.6 c_mySQL4WD :mySQLAnnule

Utiliser par la méthode **mySQLTransaction()** afin d'annuler une transaction.
Ce membre est une constante, donc passée entre parenthèse (passe par valeur).

3.7 c_mySQL4WD :mySQLDebugMode

Permet de créer un moniteur SQL traçant les requêtes envoyées au serveur. Les sorties générées par le moniteur sont envoyées au debugger installé sur le système (Visual C++, debugger système ou le programme DebugView que l'on trouve sur le serveur Unix).
Ce membre peut recevoir une valeur booléenne (Vrai ou Faux).

4 Méthodes

4.1 c_mySQL4WD :Constructeur()

Initialise la classe c_mySQL4WD() ou c_postgreSQL4WD(). En fait, cette méthode charge la librairie « mySQL4WD.DLL » ou « postgreSQL.DLL » en mémoire.

Cette méthode ne retourne aucune information.

4.2 c_mySQL4WD :Destructeur()

Clos l'instance de cette classe. Elle libère le fichier « mySQL4WD.DLL » ou « postgreSQL.DLL » chargé en mémoire.

Cette méthode ne retourne aucune information.

4.3 c_mySQL4WD :mySQLBloque(*requete, numRequete*)

Permet de bloquer les enregistrements retournés par la requête (lock à la ligne - ne fonctionne qu'avec le support de table InnoDB pour MySQL et fonctionne en natif avec postgreSQL). La requête passée en paramètre n'est pas fermée.

Cette méthode retourne un booléen.

4.4 c_mySQL4WD :mySQLConnecte(*host, user, passwd, db, [port], [socket], [flag]*)

Permet la connection à la base de donnée. Les paramètres sont les suivants :

- *host* : défini l'adresse IP du serveur MySQL ou PostgreSQL (« localhost » par défaut),
- *user* : défini le nom de l'utilisation pour cette instance,
- *passwd* : defini le mot de passe de cette instance,
- *db* : défini le nom de la base de donnée à utiliser,
- *port* : défini le port du listener (0 par défaut),
- *unix_socket* : défini le nom fichier socket à utiliser (NULL par défaut - spécifique MySQL),
- *client_flag* : défini les capacités du client (0 par defaut - spécifique MySQL).

Cette méthode retourne un booléen.

4.5 c_mySQL4WD :mySQLDateFormat(*date*)

Formate la date passée en paramètre dans un format compatible avec SQL.

Cette méthode retourne une chaine de caractères.

4.6 c_mySQL4WD :mySQLDeconnecte()

Déconnecte la session en cours et libère l'instance du serveur SQL. Les requêtes restants en instance sont automatiquement fermées.

Cette méthode ne retourne aucune information.

4.7 c_mySQL4WD :mySQLExec(*requete, numRequete*)

Envoie la requête passée en paramètre au moteur de base de donnée SQL. La requête est alors exécutée et le résultat est retourné au client. *NumRequete* permet d'identifier la requête (attention, on ne peut pas identifier plus de 5 requêtes différentes en même temps - utiliser pour cela **mySQLFerme()** et réutilisé le numéro d'identifiant libéré). La valeur de *NumRequete* est testée dans cette fonction, elle doit être comprise entre 0 et 4 sinon, une message d'erreur est renvoyé.

Cette méthode retourne un booléen.

4.8 c_mySQL4WD :mySQLFerme(*numRequete*)

Permet de libérer les ressources utilisées lors de l'appel à la méthode **mySQLExec()**. Il est obligatoire d'appeler cette méthode après chaque **mySQLExec()** même si la requête a retournée un code d'erreur. *NumRequete* permet d'identifier la requête.

Cette méthode ne retourne aucune information.

4.9 c_mySQL4WD :mySQLFetch(*numRequete*)

Permet de se positionner sur le premier enregistrement de la requête exécutée par **mySQLExec()**. Cette fonction ne récupère qu'un enregistrement (tuple) à la fois, ce qui évite l'utilisation des ressources machine. Cependant, il n'est pas conseillé d'utiliser **mySQLFetch()** pour des traitements longs sur les enregistrements (en effet, les enregistrements liés à la requête sont lockés même s'ils ne sont pas encore en mémoire). *NumRequete* permet d'identifier la requête.

Cette méthode retourne un booléen tant qu'il reste des enregistrements à lire.

4.10 c_mySQL4WD :mySQLGetErrorMessage()

Permet de récupérer le message d'erreur SQL.

Cette méthode retourne une chaine de caractères.

4.11 c_mySQL4WD :mySQLGetNumRows(*numRequete*)

Permet de récuper le nombre d'enregistrements retourné par une requête. Cette fonction ne s'utilise qu'après un **mySQLPremier()** et ne peut s'appliquer à une requête de type fetch. *NumRequete* permet d'identifier la requête.

Cette méthode retourne un entier long.

4.12 c_mySQL4WD :mySQLLitCol(*numRequete, numChamps*)

Permet de récupérer la valeur du numéro de champs passé en paramètre (correspond au numéro d'ordre dans la requête SQL). Les champs traités par cette méthode ne doivent pas retourner plus de 255 caractères. *NumRequete* permet d'identifier la requête[1].

Cette méthode retourne une chaine de caractères.

4.13 c_mySQL4WD :mySQLLitColLong(*numRequete, numChamps*)

Permet de récupérer la valeur du numéro de champs passé en paramètre (correspond au numéro d'ordre dans la requête SQL). Les champs traités par cette méthode ne doivent pas retourner plus de 64 KO de caractères. *NumRequete* permet d'identifier la requête[2].

Cette méthode retourne une chaine de caractères.

4.14 c_mySQL4WD :mySQLLitColParNom(*numRequete, nomChamps*)

Permet de récupérer la valeur du champs passé en paramètre (correspond au nom du champs renseigné dans la requête SQL). Les champs traités par cette méthode ne doivent pas retourner plus de 255 caractères. *NumRequete* permet d'identifier la requête.

Cette méthode retourne une chaine de caractères.

4.15 c_mySQL4WD :mySQLLitColParNomLong(*numRequete, nomChamps*)

Permet de récupérer la valeur du champs passé en paramètre (correspond au nom du champs renseigné dans la requête SQL). Les champs traités par cette méthode ne doivent pas retourner plus de 64 KO de caractères. *NumRequete* permet d'identifier la requête.

Cette méthode retourne une chaine de caractères.

4.16 c_mySQL4WD :mySQLLitLigne(*numRequete*)

Permet de récupérer une chaine de caractères constitué des différents champs passés à une clause SELECT, champs séparés par un caractère de tabulation. *NumRequete* permet d'identifier la requête.

Cette méthode retourne une chaine de caractères.

4.17 c_mySQL4WD :mySQLLitMemo(*numRequete, numChamps, nomChamps*)

Permet de récupérer le contenu d'un champs BLOB (binaire ou texte) dans le champs passé en paramètre (en général un champs de type image, le nom d'un bouton, d'un onglet, ...). *NumRequete* permet d'identifier la requête. *NumChamps* permet d'identifer le numéro du champs BLOB concerné. *NomChamps* permet d'identifier le champs Windev à renseigner.

[1] Voir les contraintes liées à l'identifiant au chapitre concernant **mySQLExec()**.

[2] Voir les contraintes liées à l'identifiant au chapitre concernant **mySQLExec()**.

Cette méthode ne retourne aucune information.

4.18 c_mySQL4WD :mySQLPremier(*numRequete*)

Permet de se positionner sur le premier enregistrement de la requête exécutée par **mySQLExec()**. S'il n'y a pas d'enregistrement, le membre **mySQLEnDehors** est positionné à vrai. *NumRequete* permet d'identifier la requête.

Cette méthode ne retourne aucune information.

4.19 c_mySQL4WD :mySQLPrepareBulkInsert(*nomTable, nomColonne(s)*)

Permet de préparer l'insertion de masse (bulk insert). *NomTable* permet d'identifier la table à mettre à jour. *NomColonne(s)* permet d'identifier le ou les colonnes à insérer. Si *nomColonne(s)* n'est pas passé en paramètre, la classe mySQL4WD considère que tous les champs de la table sont à insérer.

Cette méthode ne retourne aucune information.

4.20 c_mySQL4WD :mySQLBulkInsert(*buffer, numRequete*)

Permet d'envoyer à la classe d'acces natif les enregistrements à insérer dans la table. *Buffer* permet de renseigner les différents champs de la table (en relation directe avec la méthode **mySQLPrepare-BulkInsert()**). *NumRequete* permet d'identifier la requête.

Cette méthode retourne un booléen.

4.21 c_mySQL4WD :mySQLExecuteBulkInsert(*numRequete*)

Permet d'envoyer au serveur MySQL tous les enregistrements à insérer. Cette méthode doit obligatoirement être appelée en dernier. *NumRequete* permet d'identifier la requête.

Cette méthode renvoie un booléen.

4.22 c_mySQL4WD :mySQLQuoteString(*chaine*)

Renvoie la chaine de caractère passée en paramètre entre simple quote. Cette fonction doit être utilisée lors de l'utilisation de la clause WHERE.

Cette méthode retourne une chaine de caractères.

4.23 c_mySQL4WD :SetAutoCommit(*autoCommitMode, numRequete*)

Permet d'activer le mode AUTOCOMMIT (*autoCommitMode* = vrai) ou de le désactiver (*auCommitMode* = faux). Cette méthode n'est utilisée que dans le cadre du mode transactionnel. *NumRequete* permet d'identifier la requête.

Cette méthode ne retourne aucune information.

4.24 c_mySQL4WD :mySQLSuivant(*numRequete*)

Permet de se positionner sur l'enregistrement suivant. Si la fin de la sélection est atteinte, le membre **mySQLEnDehors** passe à vrai sinon il passe à faux. *NumRequete* permet d'identifier la requête.

Cette méthode ne retourne aucune information.

4.25 c_mySQL4WD :mySQLTable(*numRequete, nomTable*)

Permet de renseigner le contenu de la table *nomTable* avec le contenu de la requête identifiée par *numRequete*. Il est nécessaire que la table *nomTable* contienne le même nombre de colonne que la requête ne retourne de champs (elle peut aussi en contenir plus). *NumRequete* permet d'identifier la requête.

Cette méthode ne retourne aucune information.

4.26 c_mySQL4WD :mySQLTransaction(*mode, numRequete*)

Permet de débuter (membre **mySQLDebut**), de clore (membre **mySQLFin**) ou d'annuler (membre **mySQLAnnule**) une transaction. Cette méthode ne fonctionne qu'avec le support de table InnoDB pour MySQL et fonctionne en natif pour PostgreSQL. *NumRequete* permet d'identifier la requête.

Cette méthode retourne un booléen.

4.27 c_mySQL4WD :SetAutoCommit(*mode, numRequete*)

Permet de gérer le mode transactionnel pour les tables de type InnoDB pour MySQL et fonctionne en natif pour PostgreSQL. Si *mode* est à faux, la variable AUTOCOMMIT = 0. Si *mode* est à vrai, la variable AUTOMMIT = 1. *NumRequete* permet d'identifier la requête.

Cette méthode ne retourne aucune information.

4.28 c_mySQL4WD :mySQLPing()

Permet de tester la connection au serveur MySQL. Si la connection est tombée, **mySQLPing()** tente de reconnecter le poste client. Cette méthode n'est pas utilisée par PostgreSQL.

Cette méthode ne retourne aucune information.

4.29 c_mySQL4WD :mySQLMsgBox(*chaine*)

Permet d'afficher un message d'erreur clair. Il allie le message d'erreur passé dans *chaine* et le message d'erreur renvoyé par le serveur SQL.

Cette méthode ne retourne aucune information.

4.30 c_mySQL4WD :mySQLDernier(*numRequete*)

Permet de se positionner sur le dernier enregistrement de la requête exécutée par **mySQLExec()**. S'il n'y a pas d'enregistrement, le membre **mySQLEnDehors** est positionné à vrai. *NumRequete* permet d'identifier la requête.

Cette méthode ne retourne aucune information.

4.31 c_mySQL4WD :mySQLPrecedent(*numRequete*)

Permet de se positionner sur l'enregistrement précédent. Si le début de la sélection est atteinte, le membre **mySQLEnDehors** passe à vrai sinon il passe à faux. *NumRequete* permet d'identifier la requête.

Cette méthode ne retourne aucune information.

4.32 c_mySQL4WD :mySQLEscapeString(*chaine*)

Formate la chaine de caractères passée en paramètre afin d'être compatible avec le format des chaines de caractères du moteur SQL (en fait, elle double les simples quotes incluses dans la chaine, et insère des caractères d'échappement en cas de besoin. Elle retourne la chaine ainsi constituée encadrée par des simples quotes). Cette fonction est obligatoire lors d'appel aux commande INSERT et UPDATE.

Cette méthode retourne une chaine de caractères.

5 Exemples

5.1 Exemple de requête simple

```
Local
    retCode is boolean
    mySQL is c_mySQL4WD()

mySQL :mySQLConnecte(« localhost », « root », « mdp », « essai »)
retCode = mySQL :mySQLExec(« SELECT id, nom FROM client », 0)
if (retCode) then
    mySQL :mySQLTable(0, « TABLE1 »)
end

mySQL :mySQLFerme(0)
mySQL :mySQLDeconnecte()
```

5.2 Exemple de requête fetch

```
Local
    retCode is boolean
    mySQL is c_mySQL4WD()
    client_id, client_nom are strings

mySQL :mySQLConnecte(« localhost », « root », « mdp », « essai »)
retCode = mySQL :mySQLExec(« SELECT id, nom FROM client », 0)
if (retCode) then
    while (mySQL :mySQLFetch(0))
        client_id = mySQL :mySQLLitCol(0, 1)
        client_nom = mySQL :mySQLLitCol(0, 2)
        tableajoute(« TABLE1 », client_id + TAB + client_nom)
    end
end

mySQL :mySQLFerme(0)
mySQL :mySQLDeconnecte()
```

5.3 Exemple d’insertion de masse

```
Local
```

```
    index is int
    retCode is boolean
    customName is string

mySQL4WD :mySQLPrepareBulkInsert(« customer »)
for index = 1 to 10000
    customName = « Client n° » + NumeriqueVersChaine(index, « 06d »)
    retCode = mySQL4WD :mySQLBulkInsert(index + « ,
» + mySQL4WD :mySQLEscapeString(customName), 0)
    if (retCode = false) then
        erreur(mySQL4WD :mySQLGetErrorMessage())
        Sortir
    end
end
mySQL4WD :mySQLExecuteBulkInsert(0)
```

5.4 Exemple d'utilisation de la méthode mySQLLitLigne()

```
Local
    retCode is boolean

TableSupprimeTout(« TABLE1 »)
retCode = mySQL4WD :mySQLExec(«
SELECT id, name, create_date FROM customer », 0)
if (retCode) then
    mySQL4WD :mySQLPremier(0)
    while (not mySQL4WD :mySQLEnDehors)
        TableAjoute(« TABLE1 », mySQL4WD :mySQLLitLigne(0))
        mySQL4WD :mySQLSuivant(0)
    end
else
    Erreur(« Erreur n°
» + mySQL4WD :mySQLErreur, mySQL4WD :mySQLGetErrorMessage())
end
mySQL4WD :mySQLFerme(0)
```