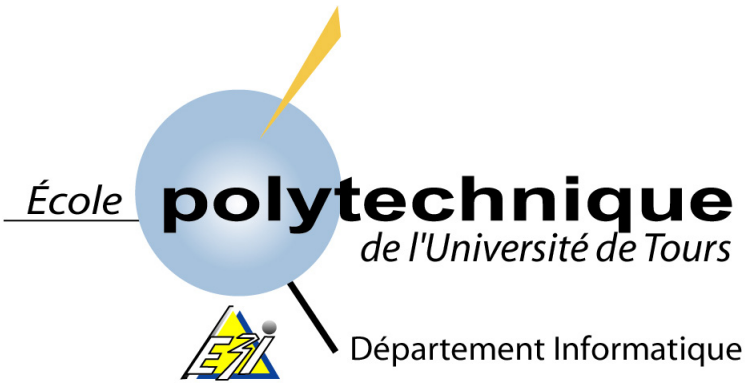
UNIVERSITE François Rabelais TOURS
Polytech'Tours-Département Informatique
64, Avenue Jean Portalis
37200 TOURS



Projet de Base de Données : Création d'une base de programmation de DAO en VB, C++, C#, MySQL et ASP

Encadré par :

H. MARTEAU

Présenté par :

Florian AGEN, Julien MICHOT
Promotion 2004-2007

Projet de Base de données :
Création d'une base de programmation de DAO
en VB, C++, C#, MySQL et ASP

Florian AGEN - Julien MICHOT

Juin 2005

Table des matières

Introduction

L'utilisation d'une base de données dans un programme informatique est aujourd'hui une chose assez courante. En effet, plusieurs services sont ainsi accessibles au travers de l'environnement Access, comme par exemple l'executeur de requêtes SQL.

De plus, il existe de nos jours une multitude de langages de programmation, comme le C++, le Visual Basic ou encore le C#, et qui ne possèdent généralement pas les mêmes caractéristiques. Aussi, la gestion d'une base de données diffère suivant le langage étudié.

La librairie de DAO de Microsoft (Microsoft DAO Library 3.6), que nous avons utilisé au cours de notre projet, offre une interface générique entre le langage de programmation choisi par le développeur d'application, et la base de données Access. Elle est donc le point commum entre ces différents langages de programmation.

Notre but dans le cadre de ce projet de base de données sera d'expliciter comment dialoguer avec une base de données suivant le langage informatique considéré. Nous allons entreprendre une comparaison des langages VB, C++, C#, MySQL et ASP au travers de différentes fonctions de base de gestion d'une base de données. Un exemple de programme sera présenté dans une dernière partie.

Chapitre 1

Généralités sur la DAO

La librairie de développement de DAO de Microsoft (Microsoft Library DAO 3.6) peut être considérée comme une interface de liaison entre le langage de programmation utilisé par le développeur, et une base de données (généralement une base de données Access). Elle contient de nombreuses fonctions de gestion, ainsi que différentes constantes nécessaires à la manipulation de la base de données.

Cette librairie est comme beaucoup de librairies, contenue dans un fichier .DLL, ce qui permet de l'inclure dans un projet quelque soit le langage utilisé (orienté objet tout de même). Le principe des DLL est justement de pouvoir faire appel à des fonctions, indépendamment du langage. On pourrait dès lors penser que gérer une base de données en VB, C++ ou C#, reviendrait à faire de la recopie !

Or, si les fonctions sont à peu près les "mêmes", les paramètres à fournir sont totalement dépendants du langage. Nous devons donc adapter notre code pour chaque langage.

1.1 La DAO suivant le langage

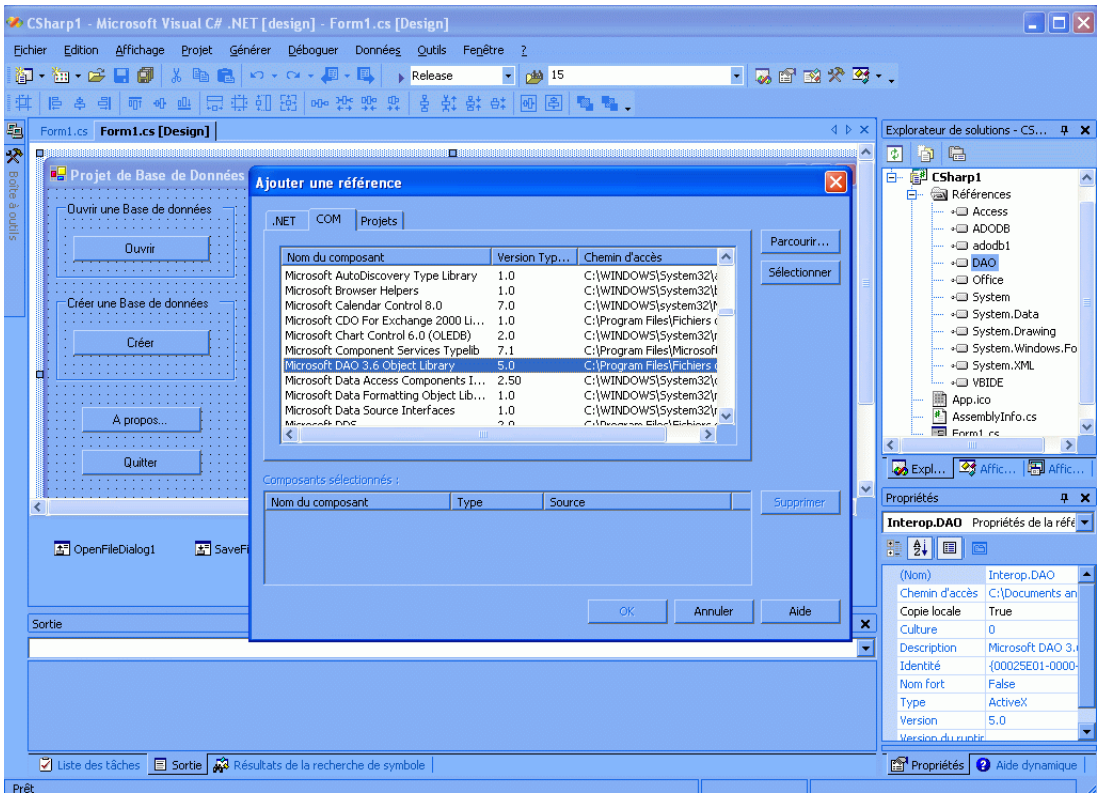
Lors de la réalisation de notre projet, nous avons commencé par créer un programme simple, qui nous permettra de montrer les fonctions de base de gestion d'une base de données, comme nous le verrons dans la deuxième et troisième partie.

*** Visual Basic :**

Le premier langage utilisé fut le Visual Basic, le langage le plus naturel pour gérer une base de données Access.

Voici la marche à suivre pour inclure et ainsi pouvoir manipuler la DAO sous VB :
- Dans l'onglet "Projet", cliquez sur "Ajouter une référence...",
- Puis sous l'onglet "COM", sélectionnez la librairie "Microsoft DAO 3.6 Object Library"
- Enfin, cliquez sur "OK"

Voici la boîte de dialogue d'ajout d'une référence (dans la version Microsoft Visual .NET) :



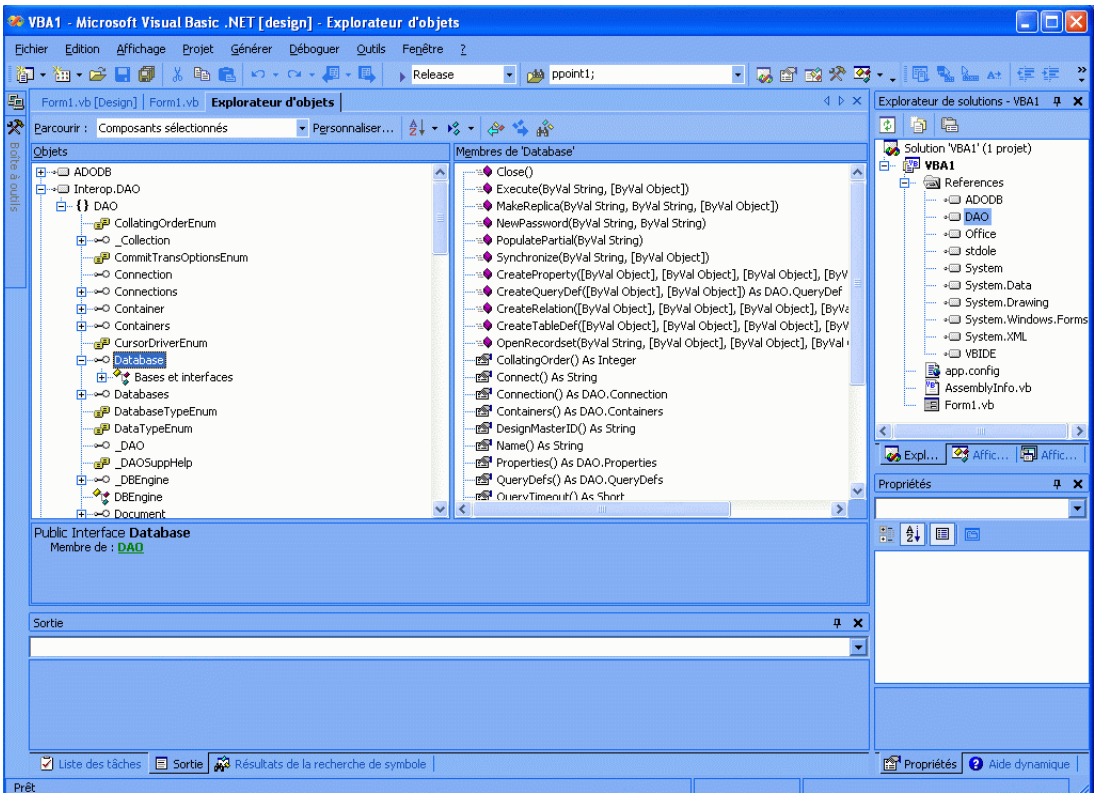
* C# :

Pour le langage C#, la procédure d'inclusion de la DAO est identique à celle du Visual Basic.

Une fois que la référence Microsoft DAO est insérée, vous avez la possibilité de consulter les différentes fonctions disponibles de la DAO dans le langage considéré, suivant la procédure suivante :

– Dans "L'explorateur de projet", cliquez sur "References",
– Puis double-cliquez sur "DAO" (ou "Interop.DAO").

Vous obtiendrez ainsi toutes les fonctions (les prototypes), classes et autres constantes définies dans la librairie.



*** C++ :**

Le langage C++ a posé beaucoup plus de problèmes que pour les langages VB ou C#. En effet, en c++, nous avons la possibilité d'inclure la librairie Microsoft DAO, mais nous ne pouvons pas accéder aux différents éléments de celle-ci, directement ! (comme notamment l'objet DataBase) Aussi, Microsoft a développé une interface appelée AFXDAO, qui redéfinit toutes les fonctions de la librairie DAO, dans un fichier nommé afxdao.h.

Donc pour faire de la gestion de base de données en c++, il est nécessaire d'inclure en tête de fichier, le header "afxdao.h" (Ce fichier est fourni par Microsoft avec Visual).

*** MySQL :**

Le langage MySQL est quand à lui plus particulier.
Premièrement, le MySQL ne peut pas vraiment être considéré comme un langage à part entière. Il n'a ni variable, si instructions de conditions ou de bouble (si, tant que tec...). Il n'est composé que de fonctions prédéfinies, commençant par "mysql_". Le MySQL doit être plutôt considéré comme une interface entre le langage SQL et l'environnement dans lequel il est utilisé. Aujourd'hui, le MySQL est souvent utilisé dans un environnement PHP, et quelque fois en c++.

Le programme de demonstration que nous avons entrepris, a été conçu sous le langage PHP, pour la partie traitement, et HTML pour la partie interface graphique.Par conséquent, pour lancer le "programme", il faut un serveur PHP (comme Apache par exemple).

De plus, le MySQL ne gère que des base de données hébergées par un serveur, en SQL. Il ne peut en aucun cas ouvrir une connexion vers un fichier ".mdb" !

*** ASP :**

L'ASP permet quand à lui de gérer une base de données Access. Il suffit pour cela de définir plusieurs variables correctement, comme nous le verrons dans la deuxième partie.

Aucun fichier n'est à inclure.

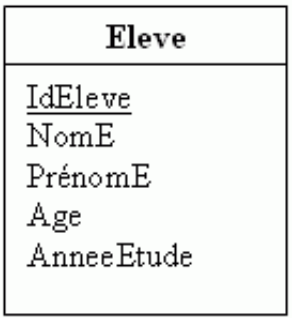
*** C :**

Aucune librairie ne permet la gestion d'une base de données en langage C. La DAO est composée de classes, elles-même composées de multiples méthodes, ce qui n'est reconnu que par un langage orienté objet, comme notamment le C++.

Le langage C, seul, ne peut donc pas manipuler une base de données Access.

1.2 La base de données

Afin d'illustrer les fonctions de gestion d'une base de données, nous avons choisis de travailler sur une base de données contenant une seule table nommée Eleve, qui possède différents champs.



Les champs sont définis ainsi :
- IdEleve : entier (clé, identifiant de l'élève)
- NomE : chaîne de caractères (nom de l'élève)
- PrénomE : chaîne de caractères (prénom de l'élève)
- Age : entier (âge de l'élève)
- AnneeEtude : entier (année d'étude de l'élève)

Le programme, duliqué dans différents langages, sera donc une gestion d'élèves, du département Informatique de Tours par exemple. L'insersion, la suppression, et la mise à jour des Ages et des Années d'étude seront implémentées.

1.3 Les types de données

Lors de la réalisation du programme, nous avons choisis d'utiliser des variables globales pour définir tous les éléments nécessaires à la gestion d'une base de données. Cela nous permet ainsi, de pouvoir comparer directement et rapidement les différences entre les déclarations.

Nous mentionnons ci-après, comment définir les objets tels que RecordSet ou DataBase, suivant le langage utilisé.

*** Visual Basic :**

Variables globales du programme en VB :

```
'la base de données
Dim db As DAO.Database
'DBEngine permet d'ouvrir et de créer une base de données
Dim DBEngine As New DAO.DBEngine
'un recordset
Dim rst As DAO.Recordset
'Table est utilisée lors de la création de la Table Eleve
Dim Table As DAO.TableDef
'déclaration des champs de la table Eleve
Dim IdEleve As DAO.Field
Dim NomE As DAO.Field
Dim PrenomE As DAO.Field
Dim Age As DAO.Field
Dim AnneeEtude As DAO.Field
```

En Visual Basic, dans la version .NET, nous sommes contraint de préciser l'appartenance des objets Database, RecordSet etc... Il faut donc définir tous les éléments de la base de données en mentionnant un DAO.LeTypeDeLObjet.

La collection DBEngine est très importante, car elle contient l'ensemble des classes utiles (Connections, Database, ...). Si l'on veut ouvrir ou créer une base de données, nous devons passer par le DBEngine.

*** C++ :**

Variables globales du programme en C++ :

```
//la base de données
CDaoDatabase * db;
//un recordset
CDaoRecordset *rst;
//Table est utilisée lors de la création de la Table Eleve
CDaoTableDef * Table;
//déclaration des champs de la table Eleve
CDaoIndexInfo IdEleve;
CDaoIndexInfo NomE;
```

CDaoIndexInfo PrenomE ;
CDaoIndexInfo Age ;
CDaoIndexInfo AnneeEtude ;

Comme nous l'avons vu précédemment, le C++ utilise la bibliothèque "afxdao" pour gérer la connexion à une base de données. Voilà pourquoi les différentes variables sont des classes, dont le nom commence par "CDao..." (Class Data Access Object).

Ces variables étant déclarées en globales (sauf les CDaoIndexInfo), nous ne pouvons pas déclarer directement des classes, mais seulement leur pointeur (ce qui est une caractéristique du C++ : pas d'allocation lors de la déclaration dans une classe). Par conséquent, une fonction d'allocation a été conçue (appelée "Initialisation"), et sera lancée une seule fois au lancement du programme.

De plus, il faut remarquer qu'aucun DBEngine n'est nécessaire, la connexion se fait directement au travers de la classe CDaoDatabase.

*** C# :**

Variables globales du programme en C# :

//Declaration d'un environnement Access
Access.Application oAccess ;
//la base de données
DAO.Database db ;
//DBEngine permet d'ouvrir et de créer une base de données
DAO._DBEngine DbEngin ;
//un recordset
DAO.Recordset rst ;
//Table est utilisée lors de la création de la Table Eleve
DAO.TableDef Table ;
//déclaration des champs de la table Eleve
DAO.Field IdEleve ;
DAO.Field NomE ;
DAO.Field PrenomE ;
DAO.Field Age ;
DAO.Field AnneeEtude ;

Les déclarations des variables en C# rappelent fortement les déclarations en VB. Nous devons ici aussi préciser le "DAO.", pour définir l'origine des classes.

La spécificité du C# réside dans le fait que nous devons aussi définir un environnement Access. Dans tous les autres langages, cette déclaration n'est pas à réaliser. Mais en C# elle est obligatoire, car c'est elle qui définira le DBEngine (utilisé pour ouvrir ou créer une base de données).
Remarque : une fonction d'initialisation d'Access est aussi nécessaire (à réaliser une seule fois) :

oAccess=new Access.ApplicationClass() ;
DbEngin=oAccess.DBEngine ;

*** MySQL :**

Variables globales du programme en MySQL :

//la base de données
global ***$db ;***
//le nom du serveur (ex :localhost)
global ***$Serveur ;***
//Nom de connexion
global ***$LogDB ;***
//Mot de passe de connexion
global ***$PassDB ;***
//Nom de la base de données
global ***$NomDB*** *;*

Pour le langages MySQL, les variables sont toutes de type identique. Il n'est donc pas nécessaire de définir un type spécialcomme pour les langages précédents.

*** ASP :**

Variables globales du programme en ASP :

'la connexion à une base de données
Dim Connexion
'un recordset
Dim MonRs

De même qu'en MySQL, l'ASP n'oblige en rien à définir le type des variables.

Après avoir défini les différentes variables globales utilisées dans nos langages, nous pouvons établir les différentes fonctions de base de la gestion d'une base de données :

- L'ouverture d'une base de données
- La fermeture d'une base de données
- La création d'une base de données
- L'insertion d'un enregistrement
- La modification d'un enregistrement
- La suppression d'un enregistrement

Chapitre 2

Programmation des fonctions de base

Nous allons définir dans cette deuxième partie, les principales fonctions de gestion d'une base de données. Chaque fonction sera de plus, codée dans 5 langages différents (VB, C++, C#, MySQL et ASP).

La grande majorité des variables qui seront mentionnées tout au long de ce chapitre sont explicitées dans la partie 1.3. Une variable appelée **fichier** contient l'adresse et le nom de la base de données à ouvrir (ou créer). **fichier** est de type *String* quelque soit le langage utilisé.

Nous mentionnerons dans ce rapport, uniquement les paramètres obligatoires des fonctions pour que la communication à une base de données s'établisse correctement. Les nombreux paramètres optionnels offerts par la librairie ne seront donc pas explicités, ce sera aux développeurs d'adapter les fonctions de base auc spécificités de leurs programmes.

2.1 Ouverture et fermeture d'une base de données

La gestion d'une base de données passe bien entendu par l'ouverture et la fermeture du fichier.

2.1.1 L'ouverture de la base de données

*** Visual Basic :**

db = DBEngine.OpenDatabase(fichier)

En Visual Basic, la connexion à une base de données s'effectue au travers de la collection DBEngine. Seule, l'adresse du fichier est nécessaire.

*** C++ :**

db → Open (fichier) ;

En C++, l'ouverture de la base de données se fait directement en appelant une méthode de la classe CDaoDataBase, et en lui fournissant simplement l'adresse du fichier à ouvrir.

*** C# :**

```
//Ouverture de la base de données
db = DbEngin.OpenDatabase(fichier,false,false,"");
```

L'ouverture en C# est déjà plus compliquée, puisque 4 paramètres sont obligatoires :
- Le premier définit l'adresse du fichier à ouvrir.
- Le deuxième définit les options d'ouverture (ici aucune).
- Le troisième décrit le caractère ReadOnly (true = ouvert en lecture seule).
- Enfin le quatrième paramètre représente la connexion.
  (par exemple :"ODBC ;DSN=Eleve ;DATABASE=Eleve ;UID=Loggin ;PWD=MotDePasse ;", ou simplement "" !)

De nombreux paramètres peuvent ainsi être spécifiés lors de l'ouverture. Nous avons réalisé notre programme avec les paramètres par défaut, comme mentionné dans l'exemple. Pour connaître chaque fonctionnalité, consultez la documentation MSDN, en cherchant...sous Visual Basic, car il n'y a aucune documentation de la DAO sous C# !

*** MySQL :**

```
//Connexion au serveur MySQL
$db=mysql_connect($Serveur,$LogDB,$PassDB);
//Ouverture de la base de données
mysql_select_db($NomDB,$db);
```

Dans le MySQL, il est obligatoire d'établir une connexion à un serveur, définit dans la variable **$Serveur**, pour accéder à une base de données (ex : "localhost"). Lorsqu'une authentification est demandée, le Login et le mot de passe se passent aussi dans la fonction mysql_connect. Tous ces champs peuvent être omis.

Il faut ensuite sélectionner la base de données (contenue dans la variable **$NomDB** ici)que l'on désire consulter (à l'aide de la fonction **mysql_select_db**).

*** ASP :**

```
'Definition de la source
DSN_BASE = "QDB="&Server.Mappath(fichier)&";
Driver={Microsoft Access Driver (*.mdb)};DriverId=25"
'Création de la connexion
SET Connexion = Serveur.CreateObject("ADODB.Connection")
'Ouverture de la base de données
Connexion.open DSN_BASE
'Ouverture d'une session
SET Session("NomSession") = Connexion
```

En ASP, nous devons créer plusieurs objets afin de pouvoir se connecter à une base de données.

- Il faut tout d'abord créer une connexion ADODB.

– Puis, le type de connexion est à définir par le développeur. Ici, on se connecte à une base de données à l'adresse **fichier**, en précisant le type de driver : driver Access.
– Enfin, nous devons déclarer une session dans la connexion.

Tous ces évènements étaient complètement cachés dans les autres langages vus précédemment.

2.1.2 La fermeture de la base de données

Voici comment fermer correctement une base de données.

*** Visual Basic :**

db.Close()

*** C++ :**

db → Close() ;

*** C# :**

db.Close() ;

*** MySQL :**

mysql_close() ;

*** ASP :**

Connexion.Close
SET Connexion = nothing

Remarque :

La fermerture est une action vraiment simple à réaliser quelque soit le langage de programmation.

Pour supprimer la communication AFXDAO, en C++, lors de la fermeture du programme, la fonction **AfxDaoTerm()** doit absolument être appelée ! Dans le cas échéant, le programme entrainera une erreur. Pour les autres langages, aucune fonction n'est indispensable loas de la fermeture du programme.

2.2 Création d'une base de données

La création d'une base de données fait aussi partie des fonctions principales pour la gestion d'une base de données.

*** Visual Basic :**

db = DBEngine.CreateDatabase(fichier,
DAO.LanguageConstants.dbLangGeneral)

En Visual Basic, lorsque l'on créer une base de données, nous devons préciser notamment la langue de la base de données. En général, il s'agit de l'alphabet courant (ASCII), on mentionne donc **dbLangGeneral**. Mais il existe bien d'autres alphabets (**dbLangCyrillic**, **dbLangGreek** etc...).

*** C++ :**

db → Create(fichier, dbLangGeneral) ;

Pour le C++, il faut aussi décrire le type de caractères qui seront dans la base de données.

*** C# :**

db = DbEngin.CreateDatabase(fichier,
DAO.LanguageConstants.dbLangGeneral,DAO.DatabaseTypeEnum.dbVersion40) ;

Dans le cas du C#, il faut aussi définir la version de la base de données à générer. La dernière version des fichiers .mdb d'Access étant la version 40. Nous avons de plus la possibilité de créer une base de données cryptée (en précisant : **DAO.DatabaseTypeEnum.dbEncrypt** ou **dbDecrypt**).

*** MySQL :**

mysql_create_db ($NomDB) ;

Le MySQL possède aussi une fonction de création d'une base de données (dont le nom est dans la variable **$NomDB**.

*** ASP :**

Aucune fonction ne permet de créer directement une base de données Access en ASP. Peut être qu'il existe des astuces, mais elles restent à définir.

2.3 Création d'une table et de ses champs

La création d'une table et de ses différents champs sont des actions qui peuvent être effectuées séparément ou en une seule requête, suivant le langage.

2.3.1 La table

En Visual Basic et en C++, créer une table dans une base de données revient simplement à appeler une seule fonction, en précisant le nom de la table.

*** Visual Basic :**

Table = db.CreateTableDef("Eleve")

*** C++ :**

Table → Create("Eleve");

*** C# :**

Table = db.CreateTableDef("Eleve",
DAO.TableDefAttributeEnum.dbAttachExclusive,"Eleve","");

Pour le C#, 3 paramètres supplémentaires sont à définir. Il y a tout d'abord le nom de la table (Eleve), les attributs de celle-ci (**dbAttachExclusive** pour signifier que l'on utilise la "Microsoft Jet database engine" (par défaut)), la source de la table (le nom de la table, encore) et la connexion (si elle existe).

*** MySQL :**

```
//Création de la requête
$query="CREATE TABLE `Eleve` (";
$query.="`IdEleve` INT NOT NULL ,";
$query.="`NomE` CHAR( 255 ) NOT NULL ,";
$query.="`PrénomE` VARCHAR( 255 ) NOT NULL ,";
$query.="`Age` INT NOT NULL ,";
$query.="`AnneeEtude` INT NOT NULL ,";
$query.="INDEX ( `IdEleve` )";
$query.=");";

//On lance la requête
$resultat=mysql_query($query,$db);
```

*** ASP :**

```
'Création d'un RecordSet
SET MonRs = Serveur.CreateObject("ADODB.Recordset");
'Création de la requête
query="CREATE TABLE `Eleve` ("
query&="`IdEleve` INT NOT NULL ,"
query&="`NomE` CHAR( 255 ) NOT NULL ,"
query&="`PrénomE` VARCHAR( 255 ) NOT NULL ,"
query&="`Age` INT NOT NULL ,"
query&="`AnneeEtude` INT NOT NULL ,"
query&="INDEX ( `IdEleve` )"
query&=");"
'Execution du RecordSet
MonRs.Open query,Connexion
```

Les langages MySQL et ASP sont plus particuliers. En effet, il n'existe pas de fonction en MySQL ou en ASP de création de table. Nous devons dès lors passer par le SQL pour générer des tables. Il faut pour les deux langages, créer la requête SQL, lancer celle-ci. Pour le MySQL,

il s'agit ici d'appeler simplement la fonction **mysql_query** qui envoit au serveur la requête. Pour l'ASP, il est nécessaire de passer par un *RecordSet*, que l'on déclare (**CreateObject**), et que l'on execute (**MonRs.Open**).

Pour ces deux langages, les champs sont aussi définis en SQL, en même temps que la table. La création de la table et de ses champs sont font par conséquents en même temps, il s'agit donc d'une seule et même action.

2.3.2 Les champs

Afin d'alléger le rapport, un seul exemple de création d'un champs sera illustré pour chaque langage.

*** Visual Basic :**

```
IdEleve = Table.CreateField("IdEleve", DAO.DataTypeEnum.dbInteger)
'Mise à jour de la Table dans la base de données
db.TableDefs.Append(Table)
```

Pour le VB, ainsi que pour tous les autres langages, le type de chaque champs doit être indiqué. Il existe de nombreux types différents, avec entre autres : dbInteger,dbDecimal,dbText,dbTime etc...).

*** C++ :**

```
Table → CreateField("IdEleve",dbInteger,10,0);
//Mise à jour de la Table dans la base de données
Table → Append();
```

En C++, la taille du champs (ici 10 (bits ou caractères)) ainsi que l'attribut sont à définir.

*** C# :**

```
IdEleve = Table.CreateField("IdEleve",
DAO.DataTypeEnum.dbInteger,10);
//Mise à jour de la Table dans la base de données
db.TableDefs.Append(Table);
```

Le C# permet aussi de préciser la taille de l'objet.

*** MySQL et ASP :**

Les différents champs de la table Eleve sont générés lors de la création de la table en elle-même. Ce référer à la création d'une table.

2.4 Insertion d'un enregistrement

La fonction la plus couramment manipuler dans une base de données est sans aucun doute l'insertion d'un enregistrement.

Par un souci de clarté, nous avons décidé de ne prendre en compte que 2 champs dans la table (IdEleve qui est un nombre, et NomE qui est une chaîne de caractères). La procédure à suivre pour les autres champs (PrénomE, Age et AnneeEtude) est identiques.

Dans cette fonction, deux variables locales sont utilisées :
- **nb**, qui est un entier, et
- **EntreeNom**, un *TextBox* dont la valeur est accessible dpar l'intermédiaire de son champ *Text*.

*** Visual Basic :**

```
    'ouvre un recordset dans la table Eleve
rst = db.OpenRecordset("Eleve")
    'création d'un nouvel enregistrement
rst.AddNew()
    'ajout des valeurs de champ
rst("IdEleve").Value = nb
rst("NomE").Value = EntreeNom.Text
    'mise à jour de la table
rst.Update()
    'fermeture du recordset
rst.Close()
```

Le VB donne la possibilité aux développeurs d'inserer un enregistrement à l'aide d'un Record-Set, ouvert sur la table Eleve. Il faut ensuite ajouter un nouvel enregistrement (**AddNew**), et définir ses valeurs de champs. Enfin, il est absolument indispensable d'enregsitrer dans la table les modifications (**Update**).

*** C++ :**

```
//ouvre un recordset dans la table Eleve
rst → Open( dbOpenDynaset, "SELECT * FROM Eleve" );
//création d'un nouvel enregistrement
rst → AddNew();
//ajout des valeurs de champ

var.Clear();
itoa(nb,str2,10);
var.SetString(str2,VT_BSTRT);
rst → SetFieldValue("IdEleve",var);

var.Clear();
```

```
strcpy( str2,StringToChar(EntreeNom → Text));
var.SetString(str2,VT_BSTRT);
rst → SetFieldValue("NomE",var);

//mise à jour de la table
rst → Update();
//fermeture du recordset
rst → Close();
```

Dans le cas du C++, plusieurs remarques sont à faire.
- Il faut premièrement fixer le type d'ouverture du RecordSet : **dbOpenDynaset** (permet de modifier la base de données).
- De plus, nous devons sélectionner la table concernée (ici, en sélectionnant tous les enregistrements de la table Eleve). *Nous avons essayé de mettre seulement "Eleve", mais le recordSet ne fonctionnait plus.*
- Ensuite, nous réalisons les mêmes actions que pour le VB, nous déclarons un nouvel enregistrement, que l'on initialise et que l'on ajoute réellement.

Mais la principale difficulté résidait dans les conversions des variables. En effet, la fonction **SetFieldValue** prend comme paramètre un type peu commun en C++ : le *COleVariant*. Nous avons donc déclarer une variable locale de type *COleVariant* : **COleVariant var;**. Ce type est en réalité le Variant du VB, il permet de transformer n'importe quel type en un autre type. Nous avons de plus utiliser une variable (str2) de type *CHAR* (tableau de chars) pour réaliser une première conversion du type *INT* vers *CHAR ** (avec le **str2**), puis du type *char ** vers *BSTRT* pour enfin envoyer la variable var dans la fonction **SetFieldValue** ! La conversion peut sembler compliquée, elle est néanmoins obligatoire.

*** C# :**

```
//ouvre un recordset dans la table Eleve
rst = db.OpenTable("Eleve",
DAO.RecordsetOptionEnum.dbConsistent);
//création d'un nouvel enregistrement
rst.AddNew();

//ajout des valeurs de champ
rst.set_Collect("IdEleve",nb);
rst.set_Collect("NomE",EntreeNom.Text);
//mise à jour de la table
rst.Update(1,false);
//fermeture du recordset
rst.Close();
```

Pour le C#, l'idée reste identique : ouverture d'un RecordSet, création d'un enregistrement, affectation des valeurs et sauvegarde des modification.

Une petite différence cependant, la fonction **OpenTable** doit avoir un paramètre *Options*, que nous avons défini comme **dbConsistent**, c'est à dire que les modifications que nous allons entreprendre doivent être prise en compte dans la base de données.

Avec le C#, le problème des conversions est totalement effacé, la fonction **set_Collect** gère très bien tous les types de données.

*** MySQL :**

```
//Création de la requête
$query="INSERT INTO Eleve ( IdEleve, NomE, PrénomE,Age
,AnneeEtude) VALUES(etc...) ";
//On lance la requête
$resultat=mysql_query($query,$db);
```

*** ASP :**

```
'Création d'un RecordSet
SET MonRs = Serveur.CreateObject("ADODB.Recordset");
'Initialisation du RecordSet
MonRs.Open "INSERT INTO Eleve ( IdEleve, NomE, PrénomE
,Age,AnneeEtude) VALUES(etc...)",Connexion
```

Pour les langages MyQL et ASP, l'insertion d'un enregistrement s'effectue en SQL, dans une requête pour le MySQL, et dans un RecordSet pour l'ASP.

Aucune mise à jour n'est nécessaire, la fonction INSERT de SQL le fait automatiquement.

2.5 Consultation des enregistrements d'une Table

Nous allons maintenant définir la procédure à suivre afin d'accéder aux différents enregistrements d'une table donnée.

Seulement deux champs seront évoqués dans cette partie (IdEleve et NomE, un entier et une chaîne). Pour le reste des champs la marche à suivre reste identique.

De plus, nous travaillerons sur un seul enregistrement. Une boucle de type *while* sera donc à ajouter pour consulter tous les enregistrements retournés, comme dans la fonction précédente.

*** Visual Basic :**

```
'ouvre un recordset sélectionnant les enregistrements que l'on désire consulter.
rst = db.OpenRecordset("SELECT * FROM Eleve WHERE ...")
'Récupération des valeurs du premier enregistrement
num = rst("IdEleve").Value
Nom = rst("NomE").Value
```

En Visual Basic, comme dans les langages C++, C# et ASP, nous devons tout d'abord appeler une requête SQL à l'aide d'un *RecordSet*. Les différents enregistrements sont alors retournés dans celui-ci.

En VB, les champs sont accessibles avec la syntaxe suivante :
NotreRecordSet("NomDuChamp").Value . Les types des données sont très bien gérés puisque qu'aucune conversion supplémentaire n'est à ajouter.

*** C++ :**

```
//ouvre un recordset dans la table Eleve
rst → Open( dbOpenDynaset, "SELECT * FROM Eleve WHERE ..." );
//Récupération des valeurs du premier enregistrement
var = rst → GetFieldValue("IdEleve");
var = rst → GetFieldValue("NomE");
```

A la différence du VB, avec la C++, il faut lancer la fonction *GetFieldValue("NomDuChamp")* du *RecordSet* pour obtenir la valeur du champ *NomDuChamp*.

La variable **var** est du type *COleVariant* (que nous avons explicitée antérieurement), donc pour avoir la valeur par exemple de l'entier **IdEleve**, il faut ecrire : **num = (int) var.intVal ;**

*** C# :**

```
//ouvre un recordset
rst = db.OpenRecordset("SELECT * FROM Eleve WHERE ...",
DAO.RecordsetTypeEnum.dbOpenDynaset,
DAO.RecordsetOptionEnum.dbConsistent,
DAO.LockTypeEnum.dbOptimistic);
//Récupération des valeurs du premier enregistrement
num = (int) rst.get_Collect("IdEleve");
nom = rst.get_Collect("NomE");
```

Le C# convertit quand à lui aisément les différents types de données. Nous utilisons en C# la méthode *get_Collect("NomDuChamp")* qui retourne la valeur du champ *NomDuChamp*, de l'enregistrement en cours.

*** MySQL :**

```
//Création de la requête
$query="SELECT * FROM Eleve WHERE ... ";
//On lance la requête
$resultat=mysql_query($query,$db);
//Acces aux valeurs :
$Eleve=mysql_fetch_array($resultat)
$num = Eleve["IdEleve"];
$nom = Eleve["NomE"];
```

Dans le cas du MySQL, les résultats peuvent être regroupés dans un tableau. C'est ce que fait la fonction **mysql_fetch_array**. Il ne reste plus qu'à accéder aux champs dont les valeurs sont dans le tableau à l'indice *"NomDuChamp"* (propriété des tableaux en PHP : on peut affecter des noms aux *cases* d'un tableau).

*** ASP :**

```
'Création d'un RecordSet
```

```
SET MonRs = Serveur.CreateObject("ADODB.Recordset");
'Execution du RecordSet
MonRs.Open "SELECT * FROM Eleve WHERE ... ",Connexion

//Acces aux valeurs :
num = MonRs("IdEleve")
nom = MonRs("NomE")
```

L'ASP se rapproche fortement du Visual Basic pour l'acces aux données : il suffit d'appeler le RecordSet en précisant le nom du champs.

2.6 Edition (mise à jour) d'un enregistrement

Pour modifier les valeurs de champs d'un enregistrement, deux méthodes existent. Soit le langage nous permet de changer les valeurs avec ses propres fonctions, comme le permmettent le VB, le C++ ou le C#. Soit nous devons construire et executer une requête SQL du type 'UPDATE ...', comme le proposent le MySQL ainsi que l'ASP.

*** Visual Basic :**

```
'ouvre un recordset
rst = db.OpenRecordset("SELECT * FROM Eleve WHERE ...")
'ouvre l'édition de l'enregistrement
rst.Edit()
'met à jour les valeurs Age et AnneeEtude
rst("Age").Value = EntreeAge.Value
rst("AnneeEtude").Value = EntreeAnneeEtude.SelectedIndex + 1
'active ces modifications
rst.Update()
```

Avec le VB, le C++ et le C#, la procédure est identique, seule la syntaxe change.

Nous devons tout d'abord créer et lancer un RecordSet sélectionnant les enregistrements que l'on désire modifier. Puis nous appelons la méthode **rst.Edit** qui précise que l'on va modier les valeurs de champs. Enfin, pour garder en mémoire les modifications, le fonction **rst.Update** doit être lancée.

Comme nous l'avons vu pour la fonction de consultation d'une base de données, les valeurs des champs d'un enregistrement sont directement accéssibles par le **RecordSet**. Il suffit donc d'affecter ces valeurs pour modifier celles-ci.

*** C++ :**

```
//ouvre un recordset dans la table Eleve
rst → Open( dbOpenDynaset, "SELECT * FROM Eleve WHERE ..." );
//ouvre l'édition de l'enregistrement
```

```
rst → Edit();
//met à jour les valeurs Age et AnneeEtude
var.Clear();
itoa((int)EntreeAge → Value,str2,10);
var.SetString(str2,VT_BSTRT);
rst → SetFieldValue("Age",var);

var.Clear();
itoa(EntreeAnneeEtude → SelectedIndex + 1,str2,10);
var.SetString(str2,VT_BSTRT);
rst → SetFieldValue("AnneeEtude",var);
//active ces modifications
rst → Update();
```

Pour le C++, l'opération de modification est plus austère. En effet, nous retrouvons le problème des conversions des variables (détaillée dans la partie Insertion d'un enregistrement).

*** C# :**

```
//ouvre un recordset
rst = db.OpenRecordset("SELECT * FROM Eleve WHERE ...",
DAO.RecordsetTypeEnum.dbOpenDynaset,
DAO.RecordsetOptionEnum.dbConsistent,
DAO.LockTypeEnum.dbOptimistic);
//ouvre l'édition de l'enregistrement
rst.Edit();

//met à jour les valeurs Age et AnneeEtude
rst.set_Collect("Age",EntreeAge.Value);
rst.set_Collect("AnneeEtude",EntreeAnneeEtude.SelectedIndex + 1);
//active ces modifications
rst.Update(1,false);
```

En C#, nous devons, comme pour tout appel de *RecordSet*, spécifier les options de celui-ci (dbOpenDynaset,dbConsistent...).

De plus, la fonction de mise à jour **Update** en C# prend 2 paramètres : le premier définit le type d'update à effectuer (de type *System.Int32*), le deuxième paramètre étant un booléen (*true* = forcer la mise à jour, *false* = na pas forcer).

*** MySQL :**

```
//Création de la requête
$query="UPDATE Eleve SET Age= ".$Age.",
AnneeEtude = ".$AnneeEtude." WHERE ... ";
//On lance la requête
$resultat=mysql_query($query,$db);
```

*** ASP :**

'Création d'un RecordSet
SET MonRs = Serveur.CreateObject("ADODB.Recordset");
'Initialisation du RecordSet
MonRs.Open "UPDATE Eleve SET Age= "&Age&",
AnneeEtude = "&AnneeEtude&" WHERE ... ",Connexion

En ce qui concerne les langages MySQL et ASP, la mise à jour des champs d'un ou de plusieurs enregistrements se fait par la procédure UPDATE du langage SQL. Aucune fonction d'écriture dans une base de données n'existe réellement en MySQL et en ASP, on utilise donc les fonction SQL.

2.7 Suppression d'un enregistrement

Enfin, la dernière fonction de base pour la gestion d'une base de données, évoquée dans ce rapport, est la suppression d'un enregistrement.

Dans tous les langages étudiés, une requête SQL doit être définie afin de sélectionner les enregistrements à supprimer. Pour alléger les explications, nous allons simplement voir comment supprimer un élément. Par extension, la suppression de tous les enregistrements retournés par une requête se fera par une boucle du type *while*, qui dépent du langage considéré (ce référer aux codes sources).

*** Visual Basic :**

'ouvre un recordset
rst = db.OpenRecordset("SELECT * FROM Eleve WHERE ...")
'Suppression de l'enregistrement
rst.Delete()

Pour supprimer un enregistrement en VB, nous ouvrons tout d'abord un RecordSet, qui sélectionne par une requête SQL les enregistrements à enlever, puis on appele la fonction **Delete**. Cette fonction supprime directement (pas besion d'**update**) l'élément en cours, et ne supprime pas tous les éléments. Il faudra donc faire une boucle (**Do While Not rst.EOF**), sans oublier de passer à l'enregistrement suivant (**rst.MoveNext()**).

*** C++ :**

//ouvre un recordset dans la table Eleve
rst → Open(dbOpenDynaset, "SELECT * FROM Eleve WHERE ...");
//Suppression de l'enregistrement
rst → Delete();

De même qu'en VB, **Delete** efface simplement le premier enregistrement. Pour tous les effacer, une boucle (**while(!rst → IsEOF())**) doit être implémentée (avec **rst → MoveNext() ;**).

*** C# :**

```
//ouvre un recordset
rst = db.OpenRecordset("SELECT * FROM Eleve WHERE ...",
DAO.RecordsetTypeEnum.dbOpenDynaset,
DAO.RecordsetOptionEnum.dbConsistent,
DAO.LockTypeEnum.dbOptimistic);
//Suppression de l'enregistrement
rst.Delete();
```

La suppression en C# est tout à fait similaire aux précédentes. Il faut néanmois préciser obligatoirement le type et les options du RecordSet, ainsi qu'un paramètre nommé LockEdit. *LockEdit* permet de définir comment les mises à jour de la table sont effectuées (lorsque plusieurs utilisateurs se connectent : **dbOptimistic**=non bloqué ou **dbPessimistic**=bloqué).

*** MySQL :**

```
//Création de la requête
$query="DELETE FROM Eleve WHERE ... ";
//On lance la requête
$resultat=mysql_query($query,$db);
```

*** ASP :**

```
'Création d'un RecordSet
SET MonRs = Serveur.CreateObject("ADODB.Recordset");
'Initialisation du RecordSet
MonRs.Open " DELETE FROM Eleve WHERE ...",Connexion
```

Les langages MySQL et ASP utilisent tous les deux la requête **DELETE** du langage SQL. Ils ne possèdent donc pas leur propre fonction de suppression.

2.8 Remarques

Il y a quelques remarques a faire sur les langages évoqués dans ce rapport.

Premièrement, il est à noter qu'il existe une librairie appelée *libmySQL*, qui permet de faire du MySQL dans un environnement C++. Le MySQL n'est donc pas cantonné à l'environnement PHP.

Deuxièmement, il est possible que l'on désire ajouter plusieurs options pour les *RecordSet* ou par les bases de données lors de leur création. Cette fonctionnalité est possible ne insérant un | entre chaque option. Voici un exemple :

dbAttachExclusive | dbAttachedODBC

Enfin, la traduction du programme de gestion d'une base de données en C# fut une opération difficile. En effet, il n'existe aucun programme écrit en C# manipulant les fonctions de DAO. Il n'existe pas non plus de documentation de la DAO en C# !

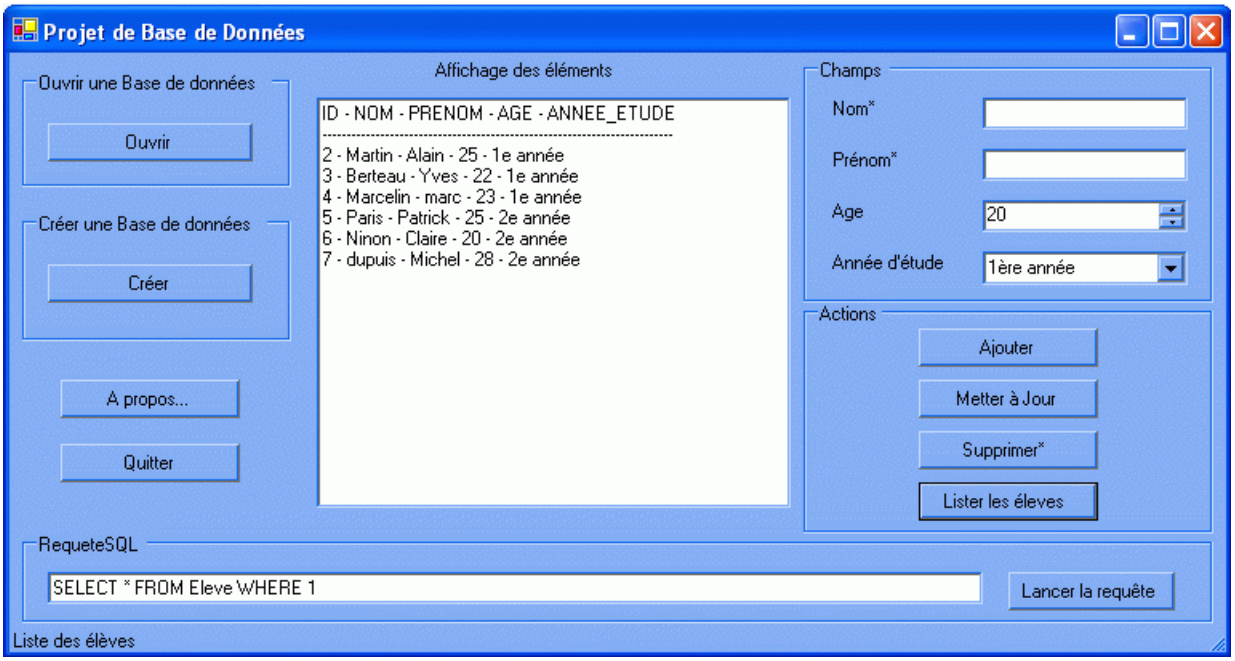
Aujourd'hui, ce n'est plus la librairie DAO qui domine le marcher, mais la librairie ADO.NET, commune à différents langages .NET. Il serait donc préférable de passer à cette librairie, qui est beaucoup plus ouverte et simple à employer.

Chapitre 3

Présentation du programme

Afin d'illustrer les différentes fonctions de gestion d'une base de données vues dans la partie précédente, nous avons entrepris la réalisation d'un même programme sous différents langages.

Voici un aperçu du programme que nous avons réalisé (en VB, C++, C# et MySQL) :



Toutes les fonctions décritent dans la partie 2 de ce rapport sont accessibles directement dans l'interface.

Voici une description des fonctionnalités du programme.

* **Ouvrir** : permet de sélectionner une base de données et lance la fonction d'ouverture.

* **Créer** : génère une base de données dont le nom est à définir, et lance la fonction d'ouverture.

*** Quitter** : ferme la base de données, et quitte le programme.

*** Lancer la requête** : permet de lancer une requête SQL, définie dans le champs texte nommé RequeteSQL. Le résultat s'affichera dans la zone d'affichage. Attention à la syntaxe !

*** Ajouter** : Ajoute un enregistrement dans la table Eleve, avec les paramètres définis par l'utilisateur (Nom, Prénom, Age et année d'étude). L'identifiant IdEleve sera égale au nombre d'enregistrements déjà présents +1.

*** Metter à Jour** : Permet de faire une mise à jour de 2 valeurs (Age et Année d'étude) d'un élève défini par l'utilisateur (champs NomE et PrénomE).

*** Supprime** : Efface un (ou des) élève(s) défini(s) par l'utilisateur (champs NomE et PrénomE), de la table Eleve.

*** Lister les Elèves** : Lit tous les enregistrements de la Table Eleve, et les affiche dans la zone de Texte. Cette opération est appelée à chaque modification de la table.

Remarques sur le programme :

Peu de controls ont été implémentés. L'objectif du projet étant la gestion de la base de données dans différents langages, la vérification de la syntaxe des champs rentrés par l'utilisateur et la gestion de l'intégrité de la base de données seront à inclure par les développeurs.

Ce programme a été conçu pour être le plus simple et le plus compréhensible que possible. La création de tables par exemple est réalisée automatiquement lors de la création de la base de données, mais elle pourrait très bien être aussi personnalisée par l'utilisateur, étant donnée que le code est fournit.

Cas du MySQL :

Pour le langage MySQL, nous avons recréé l'interface du programme sous HTML (avec du Javscript).

Pour manipuler le MySQL, nous avons utilisé comme nous le mentionnions précédemment, le langage PHP, qui gère très bien le MySQL. Un serveur PHP est par conséquent nécessaire, pour "executer" le script. Le PHP étant un langage dynamique, l'ouverture et la fermeture de la base de données sont appelées à chaque action de l'utilisateur, puisque nous ne pouvons pas laisser une base de données ouverte.

Cas de l'ASP :

Aucun exemple "physique" en ASP, n'a été conçu. Nous n'avons pas eut le temps nécessaire pour entreprendre une programmation ASP. Cependant, ce document contient tous les éléments essentiels à la création d'un projet de DAO sous ASP puisque toutes les fonctions mentionnées dans la partie 2 ont été traduites aussi en ASP.

Conclusion

Nous avons vu tout au long de ce rapport que la gestion d'une base de données était en grande partie dépendante du langage de programmation utilisé. La librairie Microsoft DAO est reconnue par différents langages, mais les prototypes des fonctions changent pour chaque langage, et il est par conséquent nécessaire de réaliser une adaptation des paramètres. Cela inclus donc une perte de temps pour la recherche de paramètres efficaces.

Ce rapport apporte donc toutes les informations essentielles à la gestion d'une base de données pour les langages de programmation VB, C++, C#, MySQL et ASP.

Les programmes fournis réalisent exactement les mêmes fonctions, et pourront donc servir de modèles pour de futurs projets de base de données.

Il est à noter que les inconvénients de la librairie Microsoft DAO sont aujourd'hui résolus par la librairie ADO.NET qui apporte de nombreuses nouvelles fonctionnalitées, et qui est beaucoup plus appropriée aux langages .NET.

Références

Cours de DAO - Visual Basic Access, années 2004-2005, Hubert Marteau.

Cours d'ASP - VBScript, années 2004-2005, Hubert Marteau.

Documentation MSDN, Microsoft - http ://msdn.microsoft.com/

Support Microsoft pour DAO - http ://support.microsoft.com/default.aspx ?scid=kb ;en-us ;164481

Résumé

La gestion d'une base de données est une opération totalement dépendante du langage de programmation utilisé. La librairie Microsoft DAO réalise l'interface de dialogue entre le langage et la base de données. Elle possède de nombreuses fonctions de gestion, mais les paramètres à gérer diffèrent suivant le langage. Ce rapport évitera donc ce travail fastidieux de recherche.

Les programmes fournis sont composés de fonctions génériques, traduites dans 4 langages de programmations.

Mots clés

Base de données, Base de programmation, Visual Basic, C++, C#, MySQL, ASP, Microsoft DAO Library.

Abstract

The database management is a totally dependent operation of the programming language used to achieve the software. The Microsoft DAO library makes the connection interface between the language and the database. The tool is full of management functions, in spite of the fact that parameters differ depending on the language. This report wants to spare users a tedious work of research.

All the software we provide are made up of generic functions, translated in four programming languages.

Keywords

Database, Visual Basic, C++, C#, MySQL, ASP, Microsoft DAO Library.