Développement professionnel avec Access

par Papy Turbo

*Un grand merci à **Maxence Hubiche** pour sa relecture approfondie, ses nombreuses et pertinentes remarques, à **Etienne Bar** pour les points forts correspondant à son expérience, et à **Pomalaix,** avec qui je partage une haine prononcée pour les franglicismes de tout poil, pour ses corrections pointues.*

Attachez vos ceintures

J'ai passé mon permis poids lourds, vers 73, sur des GMCs rescapés du débarquement et des Berliet KTs à 12 vitesses avant et 4 arrière, si on tient compte du crabot. Quel bonheur de maîtriser plus de dix tonnes, assis un mètre au dessus du toit des voitures ! Et la vue plongeante, l'été, sur les décapotables…

Et puis, 10 ans plus tard, j'ai eu l'unique occasion de faire 150 km au volant d'une Ferrari (Dino ? sais plus ! Rouge, en tout cas). Vingt-quatre heures pour décrocher de mes oreilles l'espèce de sourire béat du gamin qui vient de découvrir une grande sensation…

Dans chaque cas, de (très) grandes sensations. Mais aucun rapport entre l'une et l'autre !

Autrement dit, pour en venir à la question essentielle, et en dehors de toute considération de pollution ou de danger de la route :
— Quand est-ce que je vais prendre le semi-remorque, quand la 2 CV et quand la Ferrari ?

ou, plus précisément, en ce qui nous concerne ici,
— Que peut faire Access, quelles sont ses limites, quelles sont les possibilités qu'il est le seul à offrir ?

Dans le domaine des PME/PMI d'abord, où, apparemment, voir Forum Access : Arguments pour et contre Access ?, un consensus semble s'établir pour « accorder » une niche à Access.

Dans le domaine « réservé » au client/serveur également, j'ai nommé les grands comptes ou grandes entreprises :

D'abord, parce que je ne suis pas le seul à constater, parmi les offres d'emploi en ligne, les nombreuses recherches de développeurs Access et/ou Excel, en salles de marché, assurances...

Ensuite, parce qu'avec une architecture décentralisée, on peut réaliser des applications avec une interface Access et une base Jet ou MSDE en local, synchronisée avec la base centrale de type client/serveur. Pour une ergonomie et des performances hors du commun.

Enfin j'évoquerai l'utilisation d'Access, plus probablement sous forme de projet (.adp) avec base MSDE, comme **prototype** de mise au point d'une application lourde. Sa souplesse et sa réactivité permettent, pendant l'analyse d'un projet complexe, des économies et une rapidité de mise au point exceptionnelles.

1ère partie : les éléments principaux d'Access

Access est si riche qu'on a du mal à classer ses avantages et ses points faibles. Je me baserai donc sur la première recommandation de Microsoft concernant l'architecture d'une application Access :

Bien qu'il soit possible d'enregistrer la totalité d'une application Access dans un seul fichier de type « *.mdb », il est hautement recommandé de la scinder en deux ensembles :

1. Un fichier contenant (presque) exclusivement les tables et leurs relations : **la base Jet**, que l'on peut remplacer par une base MSDE ou client/Serveur,

2. Un fichier contenant tout le reste : attaches des tables, requêtes, formulaires, états, macros, modules de code et modules de classe…, **l'application**.

Les avantages de cette séparation ont déjà été exposés, tant pour la sauvegarde des données que pour le partage en réseau.

En suivant cette séparation, j'évoquerai donc DAO (Data Access Objects), ADO (ActiveX Data Objects) et autres techniques d'accès aux données dans la section de l'application traitant du langage de programmation : VBA (Visual Basic for Applications) est l'outil qui utilise ces objets et leurs méthodes.

Les bases de données Jet / MSDE

On ne devrait pas dire « base de données Access », mais « base de données **Jet** ». **Jet est le moteur de base de données d'Access**.

Une base de données Jet n'est PAS une base client/serveur, mais un simple fichier contenant des tables, avec intégrité référentielle (**SGBDR** = Système de Gestion de Base de Données Relationnelle).

Depuis Access 2000, il est également possible de développer sous Access avec une **base MSDE** (Microsoft SQL Server Desktop Edition). Les nouveaux projets (fichiers .adp) sont conçus pour accéder directement à une base MSDE ou SQL server (Standard Query Language, langage standard de requêtes).

Les limites

Jet et MSDE sont limités en taille : 2 Go maxi. Au-delà, utiliser une base client/serveur proprement dite.

Jet, avec une limite théorique de 255 connexions simultanées, marque une **baisse de performances à partir de 5 connexions**, et n'est **pas recommandé au-delà de 10**. **MSDE est optimisé pour 5 requêtes concomitantes**, y compris les procédures stockées. Au-delà, les performances se dégradent rapidement.

Il y a pire : Jet stocke les données dans des **Pages**. La page est un bloc de données qui contient généralement plusieurs enregistrements. Dans le cas d'ajouts intensifs et concurrentiels en particulier, l'expérience montre des blocages qui n'auraient pas lieu si Jet ne bloquait qu'un seul enregistrement à la fois.

Jet n'a pas de déclencheurs (triggers). On utilise à la place les évènements de formulaire, principalement : `Form_BeforeUpdate`, `Form_BeforeInsert`, `Form_Delete`, ... Dans ce cas, il est impératif de protéger l'accès aux bases Jet si on veut empêcher qu'un utilisateur ne modifie les données directement dans les tables, et ne leur fasse perdre leur cohérence. Ou d'utiliser une base MSDE, avec de vrais déclencheurs.

Jet n'a pas de procédures stockées. Je ne savais pas que c'était un problème. Toute **manipulation de données ou de leur structure s'effectue parfaitement en** DAO, accessoirement en ADO. Avec, par rapport au langage SQL, l'avantage considérable d'un véritable éditeur syntaxique, du débogage…

Il y a d'autres problèmes, tels que la gestion des GUIDs (Globally Unique IDentifiers), qui ne sont pas propres à Access, ni à SQL Server, mais à la différence entre la manière dont l'un et l'autre gèrent ces données. Ces problèmes, résolubles par tâtonnements, ne concernent donc que les échanges de données entre bases différentes.

Jet n'est pas un moteur client/serveur. Il s'agit d'un moteur de base de données fichier. Il est donc plus lent que MSDE ou autre base client/serveur, et plus gourmand en ressources réseau, pour exécuter la même requête, dans les mêmes conditions d'environnement réseau et machines. Mais d'une part, on ne l'utilisera pas dans les mêmes conditions, et d'autre part, la souplesse d'un simple fichier, plus le mécanisme d'attache des tables donnent des possibilités que ne fournissent pas les applications client/serveur, même avec MSDE.

Enfin, **la stabilité des fichiers Access** (fichiers *.mdb) **n'est pas garantie**. Il arrive fréquemment qu'un fichier soit endommagé et, malgré les outils de réparation, ne puisse être récupéré. **Il est donc impératif de soigner tout particulièrement sa politique de sauvegarde.**

Les avantages d'une base « fichier »

Les **sauvegardes** sont particulièrement simples à organiser.

L'anneau de vitesse : la vitesse d'exécution d'une application mono poste, dont la base est sur le même poste que l'application, **est sans commune mesure** avec celle d'une application connectée à travers un réseau Ethernet, quel que soit le langage ou le type de base utilisés. C'est le domaine privilégié des bases Jet.

Le circuit : rappelons aussi qu'Access, dans une petite entreprise de quelques postes, sans serveur de domaine ni de courriers…, sera pratiquement **la seule raison d'utiliser le réseau**. On constate alors que, le réseau n'étant pas ou peu occupé, on obtient de **remarquables temps de réponse**.

Bien sûr, dans les deux cas, **MSDE donnera de meilleures performances**. Ne serait-ce que du fait qu'un formulaire contenant des données importantes s'ouvrira plus vite : MSDE, comme SQL Serveur importe les données de la base en mode asynchrone, c'est-à-dire en tâche de fond, **pendant** que le formulaire s'affiche et que l'utilisateur commence à agir sur les contrôles.

L'autoroute, avec ses bouchons, ses engorgements : toute application critique devant utiliser le réseau d'entreprise ou, a fortiori, l'Internet pour accéder aux données, sera réservée aux bases Client/Serveur, MSDE compris.

Enfin, il importe de détailler les coûts, les possibilités d'architecture distribuée, les bases multiples, les tables jusque dans l'application elle-même…

Les coûts

J'aimerais pouvoir indiquer un ordre de grandeur des **coûts d'installation, de création et de maintenance d'une base client/serveur** : combien de personnes à plein temps, y compris concepteurs, techniciens de maintenance de la base, des serveurs, sans parler des spécialistes de réplication et autres. Rappelons qu'à part MSDE, une base client/serveur ne fonctionne pas sans une machine dédiée, dite « serveur » et qu'un serveur, avec sa maintenance, coûte cher.

Pour Access, c'est clair : **une personne dont c'est le métier à plein temps, peut parfaitement gérer plusieurs applications simultanément**. Par « gérer une application », j'entends non seulement analyse, création et maintenance de la base de données, mais aussi programmation de l'application (accès aux données + objets métier + interface utilisateur + routines système), documentation, formation, débogage, suivi de l'évolution… c'est-à-dire **tout**.

De même, sur le plan du matériel, quelques machines en réseau poste à poste. Une personne dont ce n'est pas la fonction principale, suffit, avec un minimum de formation, pour assurer les sauvegardes, contrôler les vidanges, changer une roue…

Il est clair que peu de PME/PMI, en tout cas en dessous de 10 employés, auraient les moyens d'entrer dans le club du « vrai » client/serveur.

Mais aussi : pourquoi faudrait-il que les grands comptes soient limités exclusivement aux outils plus lourds, forcément beaucoup plus chers et moins flexibles ? Parce qu'ils sont grands, donc riches ?

Architecture distribuée, avec base tampon synchronisée

Bon nombre d'applications de gestion, parmi celles qui s'appuient sur une base Client/Serveur, n'ont pas des temps de réponse critiques. Cela concerne bien évidemment toutes les applications en lecture seule telles que diffusion de catalogues, de tarifs, d'annuaires… mais aussi nombre d'applications prévisionnelles ou autres. Celles-ci se contentent parfaitement d'une mise à jour de la base de données toutes les heures, voire une fois par jour ou par semaine, ou plus simplement, à la demande.

On utilisera une base tampon, contenant un extrait de la base centrale de type client/serveur, pour permettre aux utilisateurs de

- travailler **sans aucune contrainte de réseau** : l'accès aux données, stockées dans une copie locale de la base, profitera de performances et d'une ergonomie optimales.
- travailler **hors connexion** : commercial en déplacement, travail à domicile…
- créer **des scénarios complexes** ayant des répercussions sur l'ensemble des enregistrements, **sans risque pour les données** de la base : le responsable des Ventes peut préparer une modification d'ensemble des tarifs sur plusieurs milliers de produits. Les scénarios les plus complexes peuvent être créés, annulés (« rollback »), stockés dans une ou des copies de la base tampon, et ne seront renvoyés dans la base centrale que lorsque tous les aspects auront été analysés et approuvés.
- **faire circuler** une copie des données, dans le cadre d'un travail de groupe, etc.

Un **moteur**, écrit en VBA, avec requêtes Jet, assurera la **synchronisation**, depuis et vers le Serveur. Un tel moteur, à ne pas confondre avec la réplication, est réutilisable et relativement simple. Il transfère les données très rapidement. Typiquement, 5 Mo sur un réseau très encombré, en 2 à 3 minutes. Des mises à jour ultérieures en moins d'une minute.

Des bases, des bases, oui, mais des bases Jet

En utilisant une **base Jet**, facile à copier, à sauvegarder…, il est possible de

- créer une mini-**base cryptée** pour stocker, par exemple, les innombrables codes et mots de passe requis sur Internet, les groupes et les codes utilisés pour sécuriser vos applications, les tarifs confidentiels sur un ordinateur portable…
- stocker dans le dossier « Documents and Settings*utilisateur*\Application Data\ » une base contenant les **préférences de chaque utilisateur** : dernier formulaire ouvert, taille et position des formulaires, options diverses…
- créer une copie de la base de données, pour permettre à un utilisateur d'y faire des **tests sur des données réelles**, sans pour autant affecter la vraie base,
- dans le cadre de la **formation** des nouveaux employés, créer, pour chaque nouvel élève, une copie fraîche d'une **base Exemple**, comme support de formation,
- etc. etc. etc.

Des tables DANS l'application

J'adore proposer l'inverse d'un grand principe que je viens d'énoncer solennellement : Oui, toutes les tables de données doivent être stockées dans un fichier de base de données **séparé de l'application**.

Cela dit, rien ne m'empêche de stocker **dans** l'application :

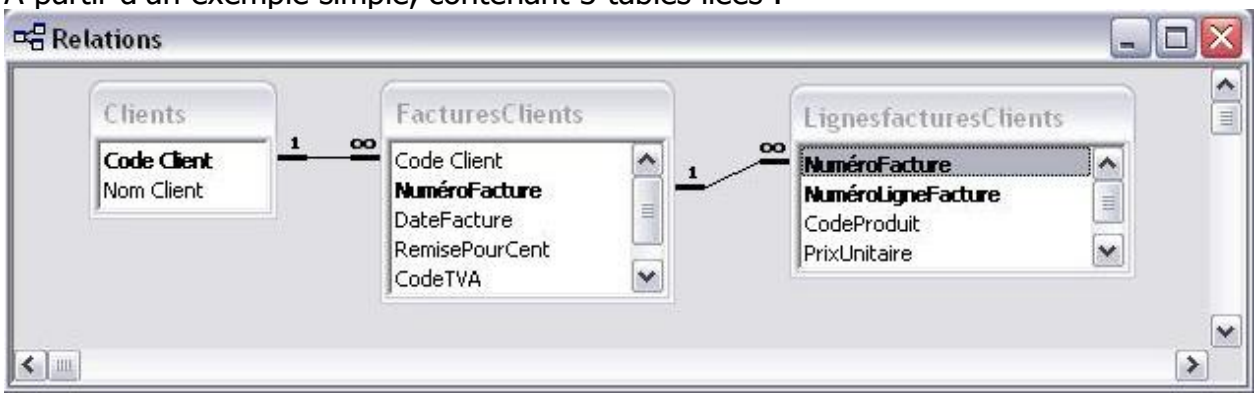
- une ou des tables de **paramètres** : les paramètres communs **à tous les utilisateurs** d'un poste y trouveront leur place. Le paramétrage des applications est un vaste sujet qui nous entraînerait trop loin (dans la base de données, dans le registre Windows, dans le dossier « Application Data »…). Mais rien n'est aussi souple et puissant que quelques tables Jet, juste là où on en a besoin.

- une table contenant **la liste des attaches de tables** à (ré-)attacher, en cas d'incident réseau, de changement du chemin d'accès ou de réinstallation…
- une table permettant de stocker les clés des **enregistrements sélectionnés** par l'utilisateur (voir La recherche Multi-critères avec Access). Cette table **doit** être locale, et **surtout pas** dans la base partagée avec les autres utilisateurs. Cette liste de clés permettra
  - d'inclure une case à cocher de **sélection individuelle** des enregistrements : l'utilisateur pourra sélectionner ainsi des enregistrements **non contigus**,
  - d'enregistrer, à l'aide d'un index, l'**ordre de tri** choisi par l'utilisateur,
  - **d'imprimer un ou des états** reflétant exactement les sélections et les tris opérés à l'écran par l'utilisateur,
  - etc. etc. etc.
- Dès que l'on essaye d'exécuter une requête mise à jour (**UPDATE** pour les fans du SQL) trop complexe, Access envoie un message d'erreur : « **Requête non modifiable** ». **Solution** : Exporter, par une requête 'Création de table', les données concernées dans une **table temporaire, locale** pour éviter tout conflit, et faire la mise à jour à partir de cette table.
- etc., etc., etc.

Hiérarchies

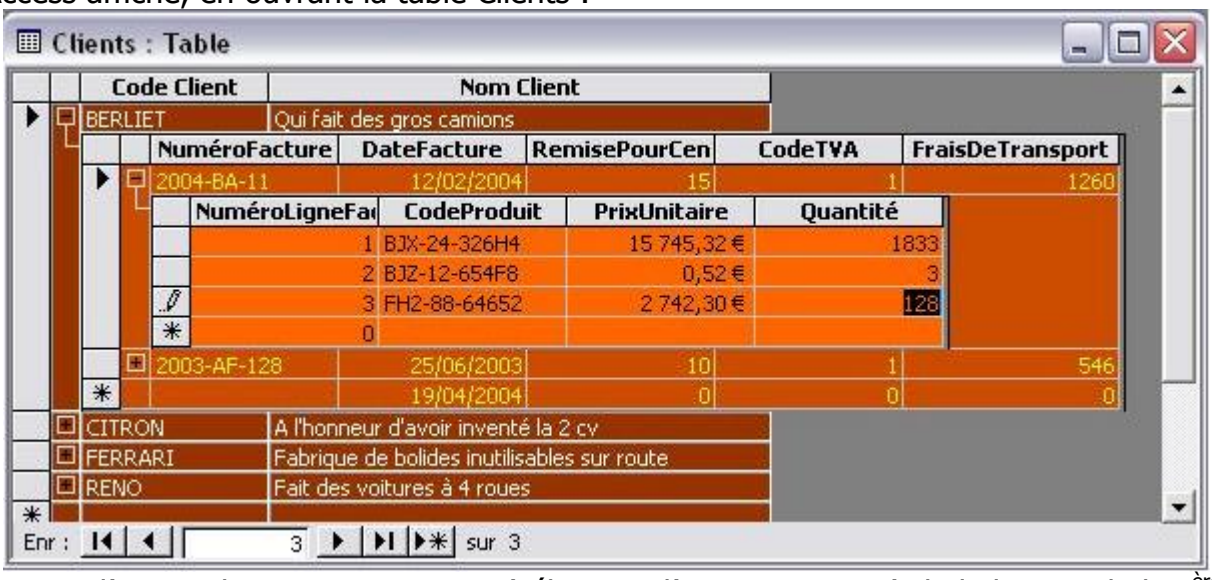
Access, depuis la version 2000, est capable de créer automatiquement une hiérarchie entre les tables, en se basant, par défaut, sur les relations d'intégrité référentielle.

À partir d'un exemple simple, contenant 3 tables liées :



Juste un petit plus, à noter au passage, qui gêne sévèrement la lisibilité des diagrammes SQL Server : **la ligne** symbolisant une relation d'intégrité va non seulement d'une table à l'autre, mais elle **pointe sur les champs** qu'elle relie.

Access affiche, en ouvrant la table Clients :



Comme l'atteste le crayon en marge (sélecteur d'enregistrement) de la ligne 3 de la 1ère facture, **toutes les tables sont modifiables**.
Enfin, pour une **requête** comme pour une table, les propriétés permettent de choisir manuellement le sous-ensemble (table ou requête) à afficher, les champs père et fils, l'affichage ouvert ou fermé par défaut...

La réplication

Access permet la réplication en utilisant les méthodes de DAO ou celles de JRO (Jet Replication Objects, distribué avec ADO). Il est recommandé d'acquérir le complément ODE (Office Developer Edition, appelé « Access 2003 Developer Extensions » depuis la version 2003) qui apporte des outils redistribuables et puissants pour la création et la gestion des réplicats (Replication Manager), ainsi que pour leur synchronisation (Synchronizer). Enfin, d'autres outils, parfois gratuits, sont disponibles sur Internet, en particulier le synchronisateur de Trigeminal, gratuit. Description en français et téléchargement sur : http://www.trigeminal.com/replication/synch.asp?1036.

Cette réplication peut se faire en mode
- **direct,** entre 2 bases sur un réseau local,
- **indirect**, entre 2 bases distantes, réseau local ou WAN (Wide Area Network),
- **Internet**.

Il est possible, avec pas mal d'efforts ou un outil spécialisé, d'y appliquer des **filtres de réplication** partielle : chaque branche, ou chaque commercial disposera juste des données qui le concernent.

La réplication n'est pas des plus simples à mettre en place, mais sans comparaison avec l'équivalent en SQL Server ou similaire, même avec MSDE.

Les applications Internet souffrent de la lenteur de connexion aux données. Même en haut débit, les performances en ligne sont hors du commun avec celles d'une application utilisant une base locale : au mieux, présentation de quelques

enregistrements (une page) à la fois, bon nombre de règles de validation sont contrôlées sur le serveur, etc.

Enfin, disons brièvement que, si les difficultés techniques concernant la réplication, propres à chaque type de base de données, sont relativement bien documentées, il manque, pour résoudre les conflits au-delà d'une grossière attribution de priorités par Réplicats, une étude approfondie sur :

- les **rôles** attribués aux utilisateurs, par application,
- les **responsabilités** sur les données, attribuées à chaque rôle, concernant
  - les champs, en **colonne** (Qui modifie quel champ ?),
  - les groupes d'enregistrements, en **ligne** (Qui modifie tel Client + ses devis… ?).

Une telle étude doit, si elle est bien menée, empêcher tout conflit de données, quel que soit le type de base, en limitant l'accès en écriture aux seuls responsables. Ou tout au moins, sachant ce que vaut la théorie par rapport à la pratique, l'alléger considérablement.

Ni plus, ni moins

On n'hésitera jamais, avec une base Jet, à faire une **copie d'archive** d'une base et à **supprimer** les champs, tables et leur contenu qui ne sont plus utiles. Sachant que l'on pourra en quelques minutes les réimporter, si nécessaire.

On constate, hélas trop souvent, au sein d'équipes plus lourdes, que prévaut le principe du très dangereux « Qui peut le plus peut le moins ». Je comprends que l'on hésite à supprimer quoi que ce soit, si on soupçonne que le besoin peut revenir, ce qui arrive souvent en phase préliminaire. Mais le résultat vire trop souvent à « l'usine à gaz » !

C'est une raison de plus, sachant qu'aucun client ne peut définir exactement ses besoins en données tant qu'il ne peut pas les utiliser, qui me font conseiller Access et une base Jet pour l'analyse de la structure d'une nouvelle base de données. Ce n'est pas le temps de création d'une table et de ses relations qui est pénalisant. En phase d'analyse, ce sont les **multiples modifications**. Dans ce domaine, MSDE est déjà un avantage certain, mais, (point de vue personnel,) Jet est imbattable.

Jet ou MSDE ?

Je constate l'enthousiasme soulevé par MSDE auprès des pros des bases client/serveur. Je suppose que pour eux, MSDE est un outil léger, idéal pour des tests au sein de l'équipe de développement. Ou pour des applications légères permettant une montée vers SQL Server quasi transparente, mais conçues dés le départ pour fonctionner avec SQL Server.

Par contre, en tant que développeur Access, orienté d'abord vers des applications qui n'auront jamais aucun besoin de « monter » vers une base client/serveur, je suis extrêmement déçu que MSDE soit bridé. LE point faible d'Access est la limite en nombre d'utilisateurs simultanés, et MSDE ne le comble pas ! C'est un comble, tout de même ! *Ah, Baronne, je ne vous le fais pas dire…*

Les quelques avantages réels : triggers, procédures stockées… existent bien. Mais, pour une application autonome, ils pèsent beaucoup moins lourd, dans mon jugement qui se base sur les réactions des utilisateurs, que la convivialité et la souplesse d'une vraie base Jet.

Réseau	Application	Type de base	
	Mono poste	Jet (.mdb)	Convivialité maximale. Performances maximales.
Réseau local peu encombré	Base partagée, 5 à 10 utilisateurs maxi.	Jet	Si pas trop de conflits sur les blocages : Convivialité maximale. Performances acceptables.
		MSDE	Perte de souplesse. Bonnes performances.
Réseau local avec serveurs (de domaine, de fichiers, de mail…)	Distribuée, de un à plusieurs dizaines d'utilisateurs, mise à jour périodique	Jet en tampon : une copie (ou un extrait) de la base sur chaque poste, synchronisée avec la base client/serveur ou Jet, répliquée	Convivialité maximale. Performances maximales. Flexibilité de la base tampon.
	Plusieurs centaines, voire milliers d'utilisateurs, mise à jour immédiate de la base centrale	Client/serveur	Convivialité minimale. Bonnes performances.
Internet	Base en lecture seule	Jet zippée, en téléchargement	Convivialité maximale. Performances maximales.
	Base moyenne en lecture et écriture	Jet répliquée	Convivialité maximale. Performances maximales. Réplication à mettre en place.
	Bases lourdes, plusieurs milliers d'utilisateurs	Client/serveur, avec ou sans réplication	Convivialité Internet. Performances Internet.

L'interface de l'application

Le mécanisme des tables attachées

Nous avons évoqué plus haut les diverses bases attachées à une application Access, ainsi que les copies multiples d'une même base. C'est le mécanisme des **tables attachées** qui permet à **la même application**, sans aucune modification, d'utiliser : une base locale, hors connexion au serveur, ou la base en réseau, une base de tests, une base exemple ou « la vraie »…

Il sera fort conseillé d'écrire et d'enregistrer dans une bibliothèque quelques routines de type `AttacheTables()` et `DétacheTables()`, afin de rendre ces opérations transparentes pour l'utilisateur, bien sûr.

Les requêtes

Access dispose de l'un des QBE (Query By Example, requêtes par l'exemple) les plus souples et les plus puissants du marché. Il est aussi facilement abordable par un débutant que capable de créer toutes requêtes actions (`UPDATE`, `INSERT`, `DELETE`), analyses croisées ou union…

Il est très simple de créer des **requêtes imbriquées** en enregistrant la première puis en l'appelant par son nom, dans la requête parent. Non seulement, cette fonctionnalité permet d'**optimiser** des requêtes complexes comme on peut le faire en SQL, mais elle clarifie la vue d'ensemble et permet une **mise au point par étapes** : tester d'abord le niveau le plus bas, puis remonter…

Sans parler des **assistants** qui font gagner un temps précieux : recherche de doublons…

Ce QBE est tellement rapide et simple que, si j'ai l'obligation, dans un programme Visual Basic par exemple, de manipuler du code SQL, je n'hésite pas à :

- ouvrir une base Access,
- attacher les tables nécessaires,
- créer une requête afin d'obtenir le résultat désiré,
- afficher les données pour **vérifier** que les résultats correspondent aux attentes,
- copier le code SQL dans l'application VB,
- remplacer les valeurs paramétrées par des noms de variable...

Sans oublier qu'il est toujours possible de saisir directement du code SQL. D'accord, ce n'est ni le standard international, ni le Transact SQL de SQL Server. Et alors ??? Est-ce que le Transact est au standard ? Non plus.

Est-ce que les applications Access ont besoin de « monter en charge » au point de devoir être réécrites pour SQL Server, ou Oracle, ou Sybase, ou ??? En dehors de l'utilisation d'Access pour prototyper une application client/serveur, non. Et, dans ce cas là, on utilisera MSDE. C'est très beau sur le papier de parler d'universalité, de

normes, mais la réalité, c'est qu'aucun langage ne respecte exactement les standards, et surtout, que la plupart des applications Access n'ont aucun besoin d'être compatibles.

Ce qui ne les empêche nullement d'être synchronisées avec toute base client/serveur.

Et j'insisterai, ayant cité ici et là Access comme prototype d'analyse pour des applications lourdes : du moment que **les règles métier sont** parfaitement analysées et **approuvées**, du moment que **la structure des données correspond aux besoins** du client, maître de l'ouvrage, réécrire entièrement l'application avec un autre langage, ou recréer la base, même sans utiliser les assistants de transfert vers SQL Server, ne représente qu'une très petite partie du travail que font, défont et refont constamment les équipes de développement en phase d'analyse.

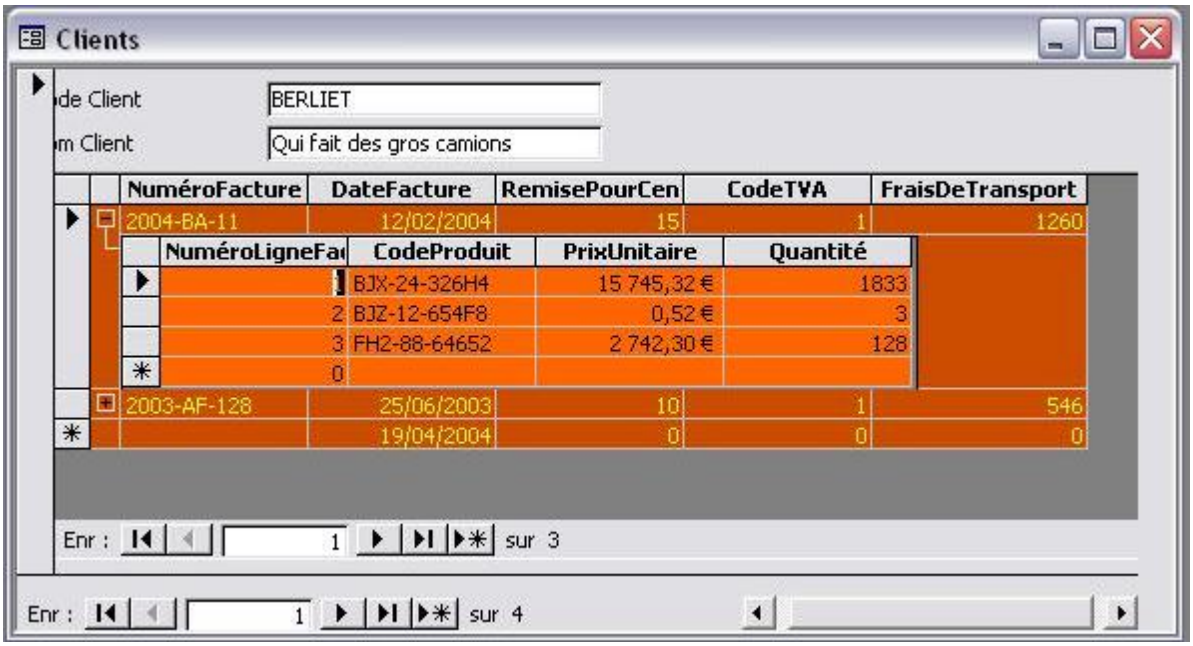
Ergonomie exceptionnelle : les (sous-)formulaires

Les formulaires Access offrent, depuis plus de 10 ans, une vue
- soit en mode « feuilles de données », en tableau,
- soit en mode « formulaire » ou fiche. Le mode formulaire permet, selon le volume de données et au choix pour le développeur :
  - soit 1 fiche unique, équivalent d'un formulaire Visual Basic,
  - soit plusieurs enregistrements, équivalent du **très récent** Data Repeater de VB.

Les sous-formulaires peuvent être imbriqués les uns dans les autres jusqu'à 3 niveaux, et même 7 depuis Access 2003. Ce qui n'empêche pas de mettre autant de sous-formulaires que l'on veut, dans des limites raisonnables, tout de même, **à côté** les uns des autres.

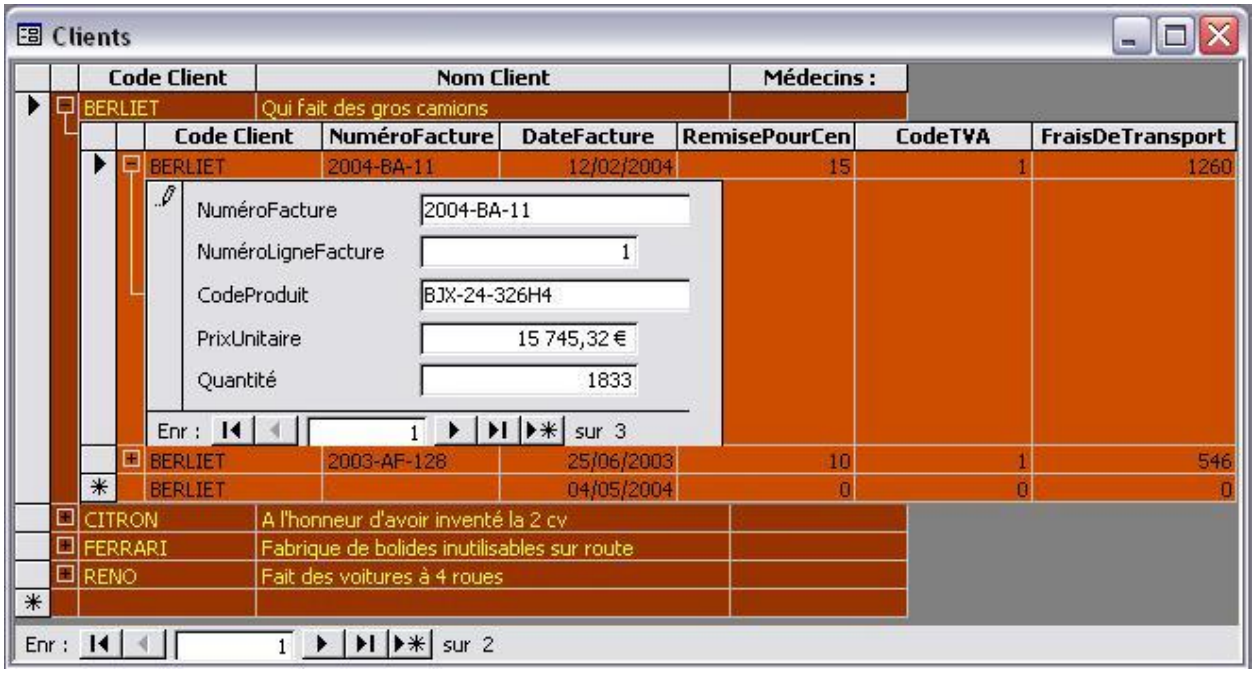
Les hiérarchies

Allez, un clic sur la table Clients de tout à l'heure, un clic sur l'assistant Formulaire instantané et hop !



1 formulaire, avec 2 tables imbriquées. Quelques clics supplémentaires remplaceront les tables par des sous-formulaires.

Et chacun de basculer, en mode formulaire, ou en mode tableau (feuille de données) :



Les possibilités des formulaires, jointes aux circonstances dans lesquelles on utilise une base Access (mono poste indépendant, réseau peu encombré ou base tampon synchronisée) font que **l'ergonomie** de l'application peut être **radicalement différente** de ce que l'on voit généralement dans les applications client/serveur :

- finies les applications à fiche unique, où on ne voit qu'un produit à la fois. Avec Access, **on n'hésite pas à ouvrir et présenter dès l'ouverture de l'application, l'ensemble de la table** voulue. De toute façon, elle est déjà en mémoire !
- pour travailler sur les détails d'un enregistrement, un clic pour basculer en mode fiche. A tout moment, retour à la « vue d'ensemble », en tableau, pour effectuer recherches, sélections, etc.
- finis les boutons « Modifier », qui ouvre un *autre* formulaire, « Ajouter » qui en ouvre *encore un autre !*, « Supprimer »…
- il est même possible, depuis Access 2000, de sélectionner un **bloc de données rectangulaire**, pour un copier-coller vers Excel, Word … ou Access.
- les menus et boutons de la barre standard, en mode formulaire ou tableau, permettent, sans une ligne de code,
  - de **chercher** une valeur dans un champ spécifique, ou dans l'ensemble des champs,
  - de **trier** la vue sur une ou plusieurs colonnes,
  - de sélectionner un texte quelconque et, **par un seul clic, filtrer les données**,
  - de créer des **requêtes plus complexes** soit par les champs du formulaire, soit par l'éditeur de requête, ou en SQL…
- en mode tableau, dit « feuille de données », il est possible

- de déplacer les colonnes dans n'importe quel ordre (très pratique pour les tris multi-colonnes),
- de les redimensionner,
- d'en figer quelques-unes à gauche du tableau, etc.

De plus, cette disposition est maintenant (Access 2000) enregistrée à la fermeture du formulaire.

Je ne peux pas refaire la liste exhaustive de tout ce que l'on peut faire avec un formulaire Access, mais je rappelle qu'aucun langage ne donne d'outils comparables. Toutes ces fonctions sont présentes pendant l'exécution, sans aucune ligne de code... Et, en dehors d'Excel et autres tableurs, je n'ai jamais vu une application écrite dans un autre langage qui donne une interface aussi riche à l'utilisateur final (notre client).

Les (sous-)états

Combien d'applications Visual Basic (ou autre) n'ont aucun état, pas le moindre petit listing à se mettre sous la dent… ? Et depuis combien de temps, VB 6 dispose t'il des Data Reports ? Access, depuis la version 1.

Les états sont tellement simples à créer et modifier qu'un des atouts majeurs d'une application Access consiste à laisser les utilisateurs, non informaticiens, créer, copier et modifier eux-mêmes toutes sortes d'états : étiquettes aux formats non standards, analyses statistiques, etc. N'oublions jamais qu'avec une version complète d'Access, l'environnement de développement est disponible partout.

Les sous-états ont le même comportement que les sous-formulaires : 3 niveaux d'imbrication, (pratiquement) pas de limites au nombre de sous-états les uns à côté, ou en dessous des autres.

Les **regroupements** sont particulièrement riches, avec options sur les dates (par an, mois, jour…, options de cumul…).

Rappelons que l'on n'est nullement obligé de se lancer dans l'automation OLE (Object Linking and Embedding, liaisons et insertions d'objets) pour créer un **mailing** : un simple état contenant le texte de base et les champs imbriqués suffit. Et peut être créé ou modifié en quelques heures, voire minutes.

On n'hésitera pas, vu la facilité, à proposer un état sous la forme : **imprimée**, **envoyée par mail** au format « aperçu » (snapshot viewer), enregistré dans un **tableau Excel**, tableau Excel envoyé par mail, en format RTF (Rich Text Format) dans Word, etc. etc. etc.

Les pages d'accès aux données

Depuis Access 2000 existent des pages d'accès aux données. J'avoue ne pas les avoir utilisées encore, probablement parce qu'il ne vient à l'idée de personne, dans un milieu de développeurs, de demander à un programmeur Access une interface Web.

Mais j'y vois un intérêt certain pour, soit des sites dynamiques personnels ou de petite entreprise, soit surtout, pour un **Intranet** de PME/PMI… et entreprises moins petites.

Les contrôles ActiveX

On peut utiliser des contrôles ActiveX dans Access. Du moins, c'est ce que dit le livre. La pratique est moins enthousiasmante.

S'il n'y a généralement que peu de problèmes à utiliser des contrôles conçus par Microsoft : Rich Text Box, Calendrier, Tree View et autres contrôles présents dans Visual Basic, il n'en va pas du tout de même pour des contrôles fabriqués avec Visual Basic, par exemple, pour être utilisés dans Access. Aucune documentation, à ma connaissance (et j'ai cherché ! et questionné !) sur Access et ses objets (Form…) en tant que conteneurs.

Le résultat, très honnêtement, est que j'ai perdu beaucoup de temps avant de m'orienter vers la très simple technique proche du « surclassement ». Plutôt que de créer un nouvel objet tel qu'un super-formulaire ou un contrôle spécialisé, attacher au contrôle standard une classe objet écrite en VBA et contenant toutes les nouvelles méthodes et propriétés, fonctionne parfaitement. C'est très rapide, **débogable sur place** et peut être enregistré dans une bibliothèque pour **réutilisation ultérieure**.

Le danger de l'ergonomie Access

Le principal danger consisterait, dans le cadre d'un développement complexe, à préparer la maquette d'un projet sous Access, d'y inclure toutes les facilités offertes à la fois par une base Jet légère et locale, mais aussi par les formulaires, les états, le langage...

Puis de s'apercevoir qu'avec un langage plus rigide, les performances et les fonctionnalités sont quasi-impossibles à reproduire, ou très lourdes : le client ayant testé la maquette Access sera déçu du résultat final. J'en vois qui sourient ! Mais combien de fois avons-nous été confrontés à un utilisateur « ravi de son tableau Excel », qui grince des dents devant un formulaire trop rigide…

À titre d'exemple bref, je vous propose d'essayer de reproduire, sous Visual Basic ou .Net, le fonctionnement « modes de formulaire : feuille de données + fiche, unique ou multiple ». Sans oublier que, même en plaçant un DataGrid sur une page d'onglets dite « feuille de données », les contrôles correspondants sur une autre page, dite « formulaire », il faut encore synchroniser les deux à chaque déplacement de curseur ; partager le code des évènements de la page DataGrid avec ceux de la page

« formulaire » ; en mode création, veiller à ce que toute modification sur une page se retrouve sur l'autre…

Il sera donc fortement conseillé, dans le cadre d'un prototype, d'utiliser une base distante, sur un serveur, et de **ne pas profiter** de toutes les possibilités ergonomiques d'Access… (« *Pas assez cher, mon fils, mais beaucoup trop riche !* »)

Programmation de l'application

Quel langage offre à la fois une programmation simplifiée par macros et un langage complet, en l'occurrence celui là même qui est au cœur de Visual Basic ?

J'aimerais faire remarquer aux détracteurs d'Access qui lui reprochent la **possibilité** de faire des applications non structurées, de bidouiller : on peut, sous n'importe quel langage et avec n'importe quelle base de données, faire des usines à gaz, des développements qui s'éternisent pendant des mois, voire des années, avant de voir le jour, ou dont l'ergonomie est frustrante pour les utilisateurs… Est-ce que, pour autant, cela donnerait le droit de décréter : « tel langage, ou les bases Client/Serveur, **ne permettent pas** de réaliser des applications simples, efficaces, réactives » ?

Les macros

Je ne m'étendrai pas beaucoup sur les macros dans la mesure où il n'est pas recommandé de les utiliser, à part 2 exceptions :

- une macro **Autoexec** lance le code d'initialisation de l'application (fonction `InitApplic()` par exemple), bien qu'on puisse également démarrer en ouvrant un formulaire,
- une macro **AutoKeys** peut contenir des raccourcis clavier, bien qu'on puisse également placer les raccourcis directement dans les contrôles, ou dans les commandes des menus.

Disons tout de même que les macros ont l'énorme mérite d'ouvrir la porte aux non programmeurs, en leur permettant de commencer sans trop de difficultés. Et, pour le plaisir d'être désagréable, je regrette amèrement qu'Access ne bénéficie pas d'un **enregistreur** qui, comme ceux de Word et Excel, produisent directement du code VBA.

Les limites des macros

Les macros sont réservées à l'usage des nouveaux venus. Profitez en !

Elles sont formellement interdites à tout développeur professionnel confirmé.

Je ne pourrai pas énumérer toutes les limites. Disons brièvement :

- pas de débogueur : test des variables, arrêt sur changement de valeur…
- pas de contrôle d'erreur : s'il y a un problème, la macro plante sans espoir,
- pas d'instruction de boucles, (`Do… While`, `For… Next`, etc.),
- expression conditionnelles permettant tout juste de sauter un groupe d'instructions, sans rapport avec un `If… Then… Else…` ou mieux encore, un `Select Case…`
- pas de programmation structurée : procédures `Sub()`, `Fonctions()`,
- pas de paramètres dans l'appel d'une macro,
- pas de variables globales non plus, pour passer une valeur. Même si je ne les aime pas (un objet vaut mieux que 1000 variables globales…),

- encore moins de programmation objet,
- aucun accès aux références extérieures : pas d'automation OLE pour piloter Word, Excel, Outlook et les autres…,
- de nombreuses limitations : string SQL de 256 caractères (au lieu de 32000 en VBA), …
- etc. etc.

Il faut savoir enfin qu'Access comporte une commande **Outils – Macro – Convertir les macros en Visual Basic**. À consommer sans modération. C'est un des meilleurs moyens d'apprendre le VBA !

Le grand intérêt des macros

Enfin, je profiterai de l'occasion pour répéter : **J'adore reprendre une application « bidouillée »**, même avec des macros, par quelqu'un qui connaît son métier.

Pour une simple raison : même si souvent, il faut tout réécrire pour concevoir une architecture solide, **les règles métier sont parfaitement définies**. Et ça, c'est un bonus non négligeable. De plus, je suis assuré d'avoir un interlocuteur qui a envie de comprendre, qui assurera sans aucun problème la validation des données et tout ce qui concerne le travail du client/utilisateur/maître de l'ouvrage en général. Un vrai bonheur.

Et que personne ne s'en plaigne : parmi tous ceux qui « bidouillent » en découvrant les bases de données grâce à Access, il y a obligatoirement quelques futurs clients pour du client/serveur…

VBA : les limites

Certaines applications, notamment scientifiques, industrielles (pas toutes !) et surtout graphiques, ne seraient pas réalisables en Access, ou souffriraient d'une exécution trop lente du code VBA semi-compilé. Ceci dit, je ne conseillerais pas Visual Basic non plus.

Le sujet des performances de laboratoire (benchmarks) par rapport aux performances perçues par l'utilisateur nous entraînerait trop loin. Mais j'insisterai lourdement sur le fait que les performances d'un moteur ou d'un langage améliorent les résultats bien sûr, mais sont beaucoup moins importantes qu'une bonne optimisation du code, des requêtes et autres éléments critiques.

Sans parler de l'adéquation aux **vrais besoins** des utilisateurs. Je ne considère pas comme une performance fascinante le fait qu'une application s'ouvre en 2 secondes pour montrer un formulaire vide. Alors qu'il faut 20 secondes ou plus à la même application, sous Access, pour afficher une table complète.

Encore moins, si la même application effectue une requête en 2,45 millièmes de secondes de moins qu'Access, mais que, au quotidien, on s'aperçoit qu'une séance de travail qui prend une demi-heure, peut être terminée en moins de 10 minutes sous Access. *Aïïïïïe, caramba ! Speedy Gonzales n'est pas toujours celui qu'on pense*.

Il n'y a aucun moyen simple d'exécuter des appels **OutOfProcess**, pour applications multi-couches. VB pas beaucoup mieux, il faut dire. .Net, ou autre, est fortement conseillé dans ce cas.

On ne crée pas de contrôles ActiveX, comme avec VB.

Accès aux données par VBA

Nous avons depuis quelque temps le choix entre DAO (Data Access Objects) et ADO (ActiveX Data Objects). Microsoft a décidé d'abandonner DAO, et **je le regrette amèrement**. **Très amèrement**.

Je préfère DAO d'abord parce que je le connais et l'utilise depuis plus longtemps, bien sûr. Mais surtout, il est **conçu pour accéder aux bases Jet** et il est beaucoup **plus complet** qu'ADO. Même ADOX, complément d'ADO qui permet d'accéder à la structure d'une base, ne permet pas de la modifier !!! Ou alors, j'ai raté quelque chose ?

Enfin, JRO (Jet Replication Objects, distribué avec ADO) est utilisé pour les opérations de réplication.

La puissance de DAO est telle qu'elle permet la création des outils d'analyse les plus complexes, tels que

- un synchronisateur pour échanger des données entre deux bases, Jet ou autres,
- l'analyse d'une base pour déterminer l'ordre des tables en fonction de l'intégrité, à utiliser pendant une synchronisation,
- la détection des boucles d'intégrité, qui interdisent toute réplication,
- la mise au point de filtres de réplication complexes,
- un moteur de mise à jour des bases de données, pour permettre à l'application d'installer la nouvelle version de la base automatiquement, en quelques minutes, en récupérant les données du client,
- tout ce que vous voudrez… (concernant les bases de données !)

Pour les accès courants aux données, nombreux sont les professionnels qui préfèrent utiliser le langage SQL, qu'ils maîtrisent parfaitement. Bien sûr, il y a des cas où le langage SQL s'impose comme la solution la plus simple et la plus efficace. Mais, en ce qui me concerne, et bien entendu, dans les cas où j'en ai le choix, **faire de l'accès aux données par SQL plutôt que par DAO est du même ordre que programmer (et déboguer !) en vbScript plutôt qu'en Visual Basic.** Et je ne parle pas des manipulations de la structure ! Les avantages de l'éditeur et du débogueur de VBA, avec les bibliothèques DAO ou ADO(X), sont sans comparaison avec la saisie et la mise au point du code SQL.

Sans compter que, pour accéder à un seul enregistrement à la fois, des méthodes telles que `Seek()` peuvent être **beaucoup plus rapides**, sur une table indexée, qu'un `SELECT… FROM… WHERE…`

Pire encore. Je vais faire frémir les puristes du « tout SQL » en admettant, que pour construire une clause `WHERE` solide, sans se préoccuper de simples ou doubles

guillemets, ni de dièses pour les dates, rien ne vaut la fonction **BuildCriteria()** d'Access.

Contrôle d'erreurs

L'avenir, nous le savons, est à la gestion d'exceptions, comme dans .Net, et, Microsoft dixit, le contrôle d'erreur est mort. Je soulèverai une objection : j'ai déjà posé cette question à plusieurs experts .Net, et n'ai pour l'instant aucune réponse.

Comment faire, avec .Net et la gestion d'exception, l'équivalent d'un **Resume**, **Resume Next**, **Resume *Label*** ?

Sans cet équivalent, il serait impossible, ou extrêmement lourd, de proposer à l'utilisateur un message d'erreur doté de 3 boutons : Réessayer (**Resume**), Ignorer (**Resume Next**) et Abandonner (**Resume *Finally***).

Tant que je n'aurai pas de réponse, **je continuerai à utiliser le contrôle d'erreur classique** de VBA. Même en .Net. La méthode, intitulée « Objectif zéro bug », consiste à **détecter toutes les erreurs non prévues** et les acheminer par mail au programmeur, avec une **trace complète** incluant les appels de procédures, et les paramètres (valeur des **données**). Ceci afin de résoudre les pires cas de débogage en quelques heures, sinon minutes. Le tout en VBA, bien entendu.

Réutilisation de code et d'objets

Certainement, ne serait ce qu'en termes d'héritage, les véritables langages objet font mieux... Peut être, un jour, .Net déteindra-t-il sur VBA ?

Mais, à condition d'utiliser des techniques simples, il est très facile d'accumuler tout type de code répétitif :

Les **bibliothèques** (aussi appelées librairies, libraries en anglais) **de code structuré et/ou objet** constituent la méthode la plus efficace pour, à la fois, réutiliser procédures et objets d'une application à l'autre, mais aussi pour leur assurer une évolution rapide et souple. Ces bibliothèques peuvent être protégées par compilation en .mde. Une simple référence vers une bibliothèque permet de la réutiliser dans n'importe quel projet.

Je ne saurais trop conseiller à tous les programmeurs Access d'accumuler ainsi des bibliothèques « utilisateur », à insérer dans toute application : une bibliothèque de base, pour la gestion d'erreur, les journaux (logs)…, une pour les accès à Excel en OLE, une vers Word, une pour les e-mails et autres fonctions Outlook, la même en CDO (Collaboration Data Objects), si Outlook est absent, etc.

Ainsi qu'une ou des bibliothèques « programmeurs », contenant tous les outils que vous créez pour vous-même : remplacement d'un nom de champ par un autre dans toutes les requêtes, insertion de code de débogage dans un module, analyses diverses, etc.

Bien que ces bibliothèques « programmeurs » aient plutôt avantage à être installées en tant que Compléments, via le **Gestionnaire de compléments**.

Le fait d'isoler dans une bibliothèque le code commun à plusieurs applications permet de mettre en valeur le code et les objets **spécifiques à cette application**. Les responsables du développement se concentreront sur les objets métier et l'interface.

L'utilisation de références à un projet Access permet également de **décomposer une application complexe** en projets distincts (plusieurs fichiers .mdb séparés), et de les présenter aux utilisateurs, soit séparément, soit comme un seul et unique projet (**intégration d'applications**).

Enfin on peut, même si ce n'est pas aussi simple que sur le papier, utiliser sous Access des **contrôles ActiveX** créés avec Visual Basic, ou achetés chez des éditeurs spécialisés.

La programmation objet

vaut largement celle de Visual Basic.

On peut programmer aussi clairement et aussi mal qu'on le souhaite, avec n'importe quel langage. Profitons en donc pour conseiller à tous les programmeurs Access et VBA d'utiliser

- un minimum de code dans les formulaires et états (l'**interface utilisateur**). Typiquement, un événement contient une ligne d'appel à une propriété ou méthode d'un objet métier, récupère éventuellement quelques valeurs en retour, et du code de contrôle d'erreur.
- effectuer tous calculs dans des modules de classe indépendants des formulaires, les **objets métier**. Ces objets permettront de se débarrasser des innombrables variables globales, mais surtout donneront une structure très claire et facile à maintenir à l'ensemble de l'application. Et pourront être copiés dans Visual Basic, si besoin est !
- regrouper en une classe ou **objet d'accès aux données** tous les accès à la base. Cette question est simple mais délicate, car il faut éviter tout conflit entre l'accès aux données directement par un formulaire, dans l'interface, et les accès depuis le code VBA.

Ceci, non pas au nom de notions théoriques quelconques, mais pour permettre un contrôle d'erreurs efficace, le partage du code avec d'autres développeurs, une maintenance et des évolutions faciles…

VBA l'universel

Le simple fait de trouver **le même langage** à la base de Visual Basic, des divers produits Office, de Corel (Draw, Photopaint…) et tant d'autres en fait, sous Windows, l'outil universel par excellence.

Dans combien de temps VBA sera-t-il balayé par la vague .Net ? Aucune idée ! Espérons seulement que la prochaine mouture sera aussi flexible et aussi largement répandue que l'actuelle.

Access : système intégré complet

En dehors de la création de graphiques, d'un bon éditeur d'aide en ligne, Access contient pratiquement tous les outils nécessaires à la création d'applications personnelles ou professionnelles.

Mais surtout, cette intégration, jointe à la souplesse des bases fichier Jet, permet une conception différente, pour le plus grand confort des utilisateurs.

Une approche différente de l'ergonomie

Étant donné qu'on ne dispose pas d'un serveur de données, que le fait d'ouvrir une table ou une requête en lit le contenu entier en mémoire, **pourquoi se priver d'afficher une ou plusieurs tables entières ?**

Avec une base Jet, on n'hésitera pas à afficher, dès l'ouverture d'une application, une feuille de données complète, donnant à l'utilisateur une vue d'ensemble sur ses données. On utilisera les sélections pour **filtrer** l'affichage. La suppression de tout filtre, pour revenir à la liste complète, est instantanée.

En client/serveur, les contraintes de réseau imposent l'inverse : partir d'un formulaire (presque) vide. Les recherches permettront d'afficher un seul enregistrement. Les sélections, s'il y en a, afficheront une liste restreinte. La liste ne sera souvent pas modifiable. La suppression de tout filtre est difficilement envisageable.

Je ne connais pas un seul utilisateur qui, ayant testé les deux configurations, préfèrerait la deuxième solution, restreinte ou vide, avec ajouts, à la première : globale, avec filtres. *Pfff ! Si, **en plus**, il faut tenir compte des goûts des utilisateurs…*

De même, plus d'une fois lors de séminaires client/serveur, j'ai eu du mal à m'enfoncer dans mon siège pour essayer de me cacher, suite aux plaisanteries du démonstrateur : je conçois qu'une liste déroulante de plusieurs milliers de lignes le fasse frémir d'horreur. J'ai atrocement honte, mais je fais cela tous les jours avec Access et Jet, sans avoir jamais subi la moindre perte de performances, même sur un « vieux » Pentium à 300 MHz.

Toujours sur les listes déroulantes, le temps de conception d'une liste multi-colonnes est hors de proportion entre Access et Visual Basic, par exemple. L'intégration de la base Jet dans Access lui donne un avantage considérable. Comme d'ailleurs, en tout ce qui concerne l'attache des contrôles aux données (binding).

Ces techniques sont tellement simples qu'on n'hésitera pas à créer des **listes dynamiques**, qui se remplissent automatiquement avec chacune des valeurs saisies dans ce champ.

Rappelons que, dans une base Jet, on peut afficher un champ d'une **table** sous forme de liste déroulante. Ou de case à cocher, ou de… Et que cette propriété se répercutera sur les requêtes, les formulaires, …

Vitesse, coûts, réactivité

L'élément principal du **coût de développement** est bien entendu le nombre d'heures de développement. C'est précisément l'**intégration** de tous les éléments dans un **environnement largement distribué** qui permet une réactivité sans comparaison.

Exemple courant de mise au point en phase de développement :

Lors de la présentation à un client d'une nouvelle version apportant de nouvelles fonctionnalités, le client souhaite l'ajout, ou une modification, d'un ou de plusieurs champs dans sa base.

Sur place, **sur le PC du client**, le programmeur Access n'hésite pas, dans la plupart des cas, à

- ouvrir la base de données, ajouter ou modifier les champs des tables,
- si les données pour remplir ce champ sont disponibles ailleurs, créer une requête pour « nourrir » le nouveau champ,
- fermer la base, retour dans l'application,
- modifier les requêtes utilisant cette table,
- ajout dans les formulaires et les états concernés,
- mise au point, dans l'événement concerné, des calculs et fonctions « métier »,
- affichage, contrôle,
- un ou 2 retours sur la base, les requêtes, les formulaires et les états pour la mise au point, jusqu'à obtenir
- l'approbation du client.

Retour au poste de développement, après avoir sauvegardé et emporté une copie de la base et de l'application modifiées.

Sur le poste du programmeur :

- ajout de code de débogage si nécessaire,
- nettoyage de la structure des objets métier,
- tests,
- distribution de la nouvelle version au Client pour mise en production bêta.

Typiquement, la version bêta est chez le client le lendemain.

Si on compare cela avec les étapes nécessaires pour faire aboutir et évoluer un projet mené par une équipe pluridisciplinaire, on a du mal à comprendre pourquoi aussi bien l'architecture de la base de données, mais surtout les règles métier proprement dites, ne sont pas mises au point sur une maquette réalisée avec Access ou autre système disponible en tout lieu et r-é-a-c-t-i-f.

2ème partie : Programmer pour un client

Même avec Access, si l'on parle de développement professionnel, il faut au minimum : un développeur **et un client**, ou « utilisateur final ».

Le client est celui qui utilisera le logiciel et donc, qui gagnera du temps, de la précision, de l'information ou tout autre bénéfice à tirer d'une application bien conçue. Ou qui en perdra, s'il bute sur un obstacle quelconque.

Avec le développement Access et grâce à la proximité client/développeur, se développe une tendance à **faire passer les préoccupations du client/utilisateur avant toute chose** :

- flexibilité de l'analyse : il est normal que le client change d'avis et précise ses besoins, au fur et a mesure de la découverte du nouvel outil,
- ergonomie plus souple grâce aux formulaires, aux états, à la présentation globale des données et à l'utilisation d'une interface riche,
- souplesse pour les mises au point et évolutions diverses,
- tendance à satisfaire ses demandes et rien de plus,
- possibilité de laisser le client personnaliser certains éléments ou de compléter lui-même l'application,
- intégration avec Office et autres applications familières à tous les usagers…

L'analyse

La méthode favorite, hélas pas toujours possible, est de prendre la place d'un assistant, **de faire le travail « à la main » avec les utilisateurs**, afin de comprendre leur métier : avant, pour l'analyse ; pendant les tests ; après, avec la nouvelle application, pour vérifier que les réalisations correspondent aux besoins **réels** et optimiser ce qui en a besoin.

Un cahier des charges sera toujours indispensable, surtout si l'on procède par devis forfaitaires. Mais il sera généralement très succinct, se contentant de préciser les fonctionnalités à mettre en œuvre.

Même pour un simple devis, phase pendant laquelle on essaye de fournir un minimum d'investissement, on n'hésitera pas à créer une base et divers éléments. C'est encore la manière la plus simple et la plus efficace de noter ce qu'il faudra faire.

Bien sûr, ici comme ailleurs, il importe d'utiliser les méthodes d'analyse appropriées : Merise, UML, etc. Avant, pour clarifier tout cas complexe. Ou après, en rétro-ingénierie, pour transmettre l'information à l'équipe de développement.

Nous savons que la légèreté des fichiers Jet permet de les inclure dans une politique de sauvegarde qui permettra tous les retours en arrière, si nécessaire. Il en va de même pour le fichier applicatif. Ce qui n'empêche pas d'utiliser SourceSafe ou autre utilitaire de gestion du développement.

Le développement

Le développement, avec la souplesse d'Access, fait partie de l'analyse. Il faut toujours s'attendre, quelle que soit la qualité du cahier des charges ou des schémas et diagrammes d'analyse, à ce que le client ne précise, adapte et **finalise ses besoins** exacts que **lorsqu'il aura utilisé**

- **une application opérationnelle,**
- **avec de vraies données** (« Où sont mes produits ? et les produits non standard ? et mes Clients ? et celui qui pose toujours problème… ? »)

La vitesse et la souplesse de développement d'Access permettent de décomposer les besoins par phases ou étapes, les réaliser une par une, en produisant **dès que possible** « quelque chose qui marche », que l'on teste et qu'on met en production immédiatement.

Puis rajouter, étape par étape, dans l'ordre des **priorités définies par le client**, toutes fonctionnalités ultérieures.

La sécurité

La sécurité sous Access est un ensemble très complet, souvent mal utilisé :

- sécurité par mot de passe :
  - gestion des utilisateurs et des groupes, que l'on peut utiliser pour détecter si l'utilisateur est un programmeur ou non,
  - gestion fine des autorisations par objet : tables, attaches de tables, requêtes (autorisations spécifiques ou autorisations « du propriétaire »), formulaires, états, pages d'accès aux données,
  - depuis Access 2000, un mot de passe simplifiant la sécurité du code VBA,
  - possibilité, peu utilisée, d'un mot de passe global pour protéger une base de données,
  - possibilité, dans un Espace de travail (« Workspace ») indépendant, de donner à l'application des autorisations dont l'utilisateur en cours ne dispose pas : le code VBA pourra, par exemple, accéder à une table de paramètres protégée contre toute manipulation par les utilisateurs,
- possibilité de distribuer une application compilée (.mde),
- cryptage de données sensibles,
- sécurité d'une bonne procédure de sauvegarde,
- mais aussi stabilité du code : contrôle d'erreur rigoureux,
- etc.

Le déploiement

Pour les cas particuliers incluant DLLs, contrôles ActiveX et autres références, l'assistant de déploiement (n'a pas toujours été, mais) est devenu simple et efficace, surtout depuis la version 2000. Il est livré avec l'ODE (Office Developers' Edition).

Pour une première installation incluant de tels composants, il est facile de déployer, **dès que possible**, une mini application ne comportant qu'un formulaire et 2 ou 3 boutons

pour vérifier chaque composant externe. **Le plus tôt** seront détectés et résolus de tels problèmes, le plus tôt une solution pourra être trouvée **sans avoir à (re-)modifier le reste de l'application**.

Enfin, rappelons que les programmeurs Access n'ont pas attendu .Net pour réaliser leurs installations par une **simple copie de fichiers**. Même dans le cas ci-dessus, il n'y a besoin de faire fonctionner un véritable programme d'installation qu'**une seule fois par poste**. Les mises à jour ultérieures pourront être exécutées par simple copie des fichiers .mdb ou .mde : bases, applicatifs et bibliothèques.

Au cas où on utilise des bibliothèques, il suffit de les placer dans le même dossier que l'application pour qu'Access rétablisse les références.

La formation

La formation des utilisateurs est simplifiée du fait que l'ergonomie, la réactivité des développeurs, leur proximité avec les utilisateurs rendent le besoin d'une documentation complète très rare. En tout cas pour les applications sur mesure, destinées à un public restreint, ce qui est le cas le plus fréquent.

Même dans ce cas, Access et Jet offrent des possibilités particulières déjà évoquées, telles qu'une base de tests ou de formation, à côté de la base réelle.

Sinon, un simple état ou un formulaire, ouvert par un bouton d'aide, peut suffire pour afficher les instructions et démarches à suivre.

Le débogage, ou phase beta

Outre les multiples possibilités du langage VBA, le fait qu'Access soit présent sur un grand nombre de postes de travail en entreprise permet au développeur **le débogage sur site**.

Nous connaissons tous les bugs impossibles à reproduire sur le poste de développement. Généralement liés à la version de Windows et aux autres logiciels plus ou moins fantaisistes installés sur chaque poste, le fait de s'installer sur le poste de l'utilisateur ou de le prendre en main à distance et de passer en mode de débogage peut représenter un gain de temps considérable. C'est un des points sur lesquels, quels que soient leurs avantages par ailleurs, **aucun langage compilé ne pourra égaler Access**.

Évidemment, cet argument ne tient pas si l'utilisateur tourne « en runtime ». Mais, dans une PME/PMI, un seul poste doté d'un Access complet permet généralement toutes les interventions sur site, pour un coût très bas.

Vous avez dit « méthodes agiles » ?

Mon impression, en comparant, chaque année, les nouveautés d'Access d'une part aux nouveautés de Visual Basic et de SQL Server d'autre part, mon impression, donc, est

qu'Access dispose régulièrement de nouvelles fonctionnalités qui, soit n'apparaîtront jamais dans les produits plus lourds parce que trop complexes à y mettre en oeuvre, soit n'y figureront que plusieurs années plus tard.

J'allais dire : .Net est hors comparaison pour l'instant. Sauf qu'il semblerait bien que Whidbey semble promettre des sous-formulaires .Net pour l'année prochaine ? Avec un peu d'espoir, les programmeurs .Net pourront peut être, d'ici à 2010, utiliser des formulaires avec mode « feuilles de données », comme dans Access ? *Non, je rigole.*

En tout cas, si ce bref aperçu a pu donner envie à quelques pros du client/serveur et autres langages nobles de venir voir de plus près nos modestes réalisations, je suis convaincu qu'ils en ressortiront avec bon nombre d'idées nouvelles, dynamiques, et constructives.

J'irai plus loin, en faisant remarquer que,
comme M. Jourdain faisait de la prose sans le savoir,
l'ergonomie, l'état d'esprit, l'approche globale que permettent Access font que, généralement sans le savoir, les développeurs Access utilisent, depuis plus de 10 ans, ces **méthodes agiles** dont on parle depuis quelques années à propos d'environnements tels que .Net ou J2EE.

Enfin, que dire de plus que :

Utiliser Access à bon escient,
c'est enclencher le turbo
dans vos développements.