Sommaire

Introduction

Dans le cadre de la formation de première année d'IUP Génie Informatique, nous allons réaliser une étude comparative de deux versions d'un même logiciel : Microsoft Access.

Ce projet est sous la direction de Sylvie Damy, maître de conférence à l'Université de Franche-Comté, responsable de l'enseignement de base de données en IUP1. La finalité de ce projet est de changer de version de logiciel dans le cadre des travaux pratiques de l'enseignement de base de données.

Cette étude aura pour support les travaux pratiques réalisés par Mme Damy, qui sont soumis aux étudiants de première année d'IUP GMI. Nous réaliserons une études de ces travaux pratiques sous Access 97 en faisant ressortir les points caractéristiques et éléments importants de ces travaux pratiques. Puis nous effectuerons une migration des ces travaux pratiques vers Access XP pour observer les différences que cette version apporte.

Dans la seconde partie de cette étude, nous aborderons la partie programmation qui se révèle fortement différente à celle des travaux pratiques d'Access 97. Cette partie sera traitée en détail car elle d'une grande importance dans l'enseignement d'un IUP informatique.

Tout au long de cette étude vous trouverez des copies d'écrans ou des exemples de code qui illustreront les propos traités dans un souci de clarté. De plus des annexes sur la partie programmation disponibles en fin de rapport sont une source supplémentaire d'informations. Il sera donc parfois intéressant de se reporter à ces annexes pour celui qui désire de plus amples précisions.

Pour finir, cette étude n'est pas complètement exhaustive et ne prétend pas aborder toutes les différences entre les deux versions du logiciel. Mais nous y avons traité les changements et apports qui nous ont paru fondamentaux.

Chapitre 1 – Avant propos

Nous allons tout d'abord dans ce premier chapitre mettre en place le cadre de notre étude, ce qui nous permettra de préciser à la fois le contexte, les moyens et les buts de ce projet.

Présentation du sujet : Analyse comparative Access 97 / XP

Ce projet de Base de Données a pour but de dresser une étude de l'évolution du logiciel Access© de Microsoft. Ce logiciel est un SGBD (Système de Gestion des Base de Données) au même titre que MySQL, Oracle et de nombreux autres....
Les SGBD sont une famille de logiciels permettant de mettre en place un système d'information qui permet de regrouper, d'organiser et de manipuler les informations nécessaires à toute entreprise ou organisation.

En effet grâce à cet outil, il est possible pour l'entreprise d'organiser :

- ✓ La gestion des salaires des employés, de l'automatisation des calculs de feuille de paye à leur impression automatique tous les 1er vendredis du mois.
- ✓ La gestion des factures des fournisseurs ou des commandes de clients, et pourquoi pas relier cela au site Internet de la société afin de créer un service de vente par correspondance.

Ou, à titre personnel, un particulier peut être amené à utiliser Access pour

- ✓ Gérer une discothèque, une vidéothèque ou tout type de collections.

A travers la présentation d'Access faite dans la quatrième partie de ce chapitre, nous présenterons plus en détail les concepts de base d'un SGBD. Ces logiciels sont très répandus et il est important de savoir s'en servir car s'ils sont très utiles, ils sont parfois aussi peu accessibles voire austères.

Mais les éditeurs ont compris qu'aujourd'hui les entreprises privilégient des logiciels ergonomiques (i.e. facile d'utilisation) permettant une meilleure productivité.
Ils accordent maintenant une importance à la disposition des menus, des icônes sur l'écran selon la fréquence d'utilisation des fonctions associées, de manière à obtenir un sentiment de commodité immédiate dès la première approche du logiciel.

Au cours de ce rapport, nous découvrirons les nouveautés et améliorations qu'apporte la version 2002 d'Access (ou Access XP) par rapport à la version 97.

Il s'agit de mettre en évidence les différences qui existent tant au niveau de l'interface qu'à l'intérieur même du logiciel, car s'il est vrai que ces cinq ans de développement n'ont pas créés de changements révolutionnaires au sein d'Access, il existe de nombreuses différences qui peuvent provoquer bon nombre de problèmes difficiles à identifier lors de la migration d'Access 97 vers Access XP.

Description du travail

Le but de ce projet est donc de faire une analyse comparée des deux versions d'Access, au travers d'un cas pratique afin d'être en mesure de porter le plus facilement possible une base de données Access 97 sous Access XP.

En outre, nous tenterons d'avoir un regard critique vis à vis du support de cette comparaison, à savoir les travaux pratiques du module de Base de Données d'IUP1.

Afin de souligner les problèmes posés par une migration de Access 97 vers Access XP, nous allons refaire les travaux pratiques du module de base de données d'IUP1 adaptés à la version 97 sous Access XP. Ainsi nous serons en mesure d'identifier les problèmes posés par une telle migration.

Nous chercherons ainsi à identifier les différences qui peuvent poser des problèmes, tant au niveau de l'interface que des options proposées par Access tout au long de la création et de l'administration d'une base de données.

Cependant, la majeure partie des différences que nous relèverons se situera au niveau du langage de programmation de la version 97 et XP, étant donné qu'il a été profondément revu entre ces deux versions et qu'il est à l'origine des problèmes les plus complexes en terme d'adaptation.

Contexte technique

Les travaux pratiques de base de données ayant été réalisés sous Access 97, l'étude utilisera les logiciels Microsoft Access 97 et Microsoft Access XP avec leurs VBA associés c'est-à-dire respectivement VBA 3.0 et VBA 6.0. Ces technologies ainsi que les méthodes d'accès aux données seront détaillées dans la partie programmation.

Présentation d'Access

Comme nous l'avons dit précédemment Access est le SGBD relationnel fourni par Microsoft. Un SGBD permet de constituer des collections de données que l'on pourra organiser, sélectionner, trier, modifier, mettre à jour …

Ces collections de données sont contenues dans une base de données. Access propose différents objets pour constituer la base tels que des tables pour stocker les données, des requêtes pour extraire et organiser des données, des formulaires pour réaliser des interfaces utilisateurs, des états permettant des impressions, des pages offrant la possibilité de créer des pages Web actives... Grâce à Access, trier, visualiser, imprimer ou transmettre sur Internet des données sera donc possible.

La première version de Microsoft Access date de 1995. Depuis de nouvelles versions sont apparues : Access 97, Access 2000 et Access 2002. Une des mieux accueillie fût Access 97 pour ses performances remarquables à l'époque. Cependant les versions 2000 et 2002 ont complètement changé d'un point de vu programmation mais cela ne concerne que les utilisateurs confirmés. En effet le mode d'accès aux objets s'est profondément modifier passant de la technologie DAO (Data Access Objects) à la technologie ADO (ActiveX Data Objects). De plus avec l'expansion du Web, Microsoft a adapté son logiciel à l'Internet en instaurant depuis la version 2000 la possibilité de créer des pages Web appelées « Pages d'accès aux données ».

Access est un SGBD qui a su évoluer et multiplier ses possibilités, du fait qu'il soit développé par l'une des plus grande entreprise de développement de logiciel. Mais la grande notoriété de Microsoft et leur certitude concernant leurs ventes les amène parfois à ce négliger concernant certaines parties des logiciels telle que l'aide qui n'est mise à jour que part l'ajout d'informations ce qui rend l'ensemble complexe et difficile d'utilisation.

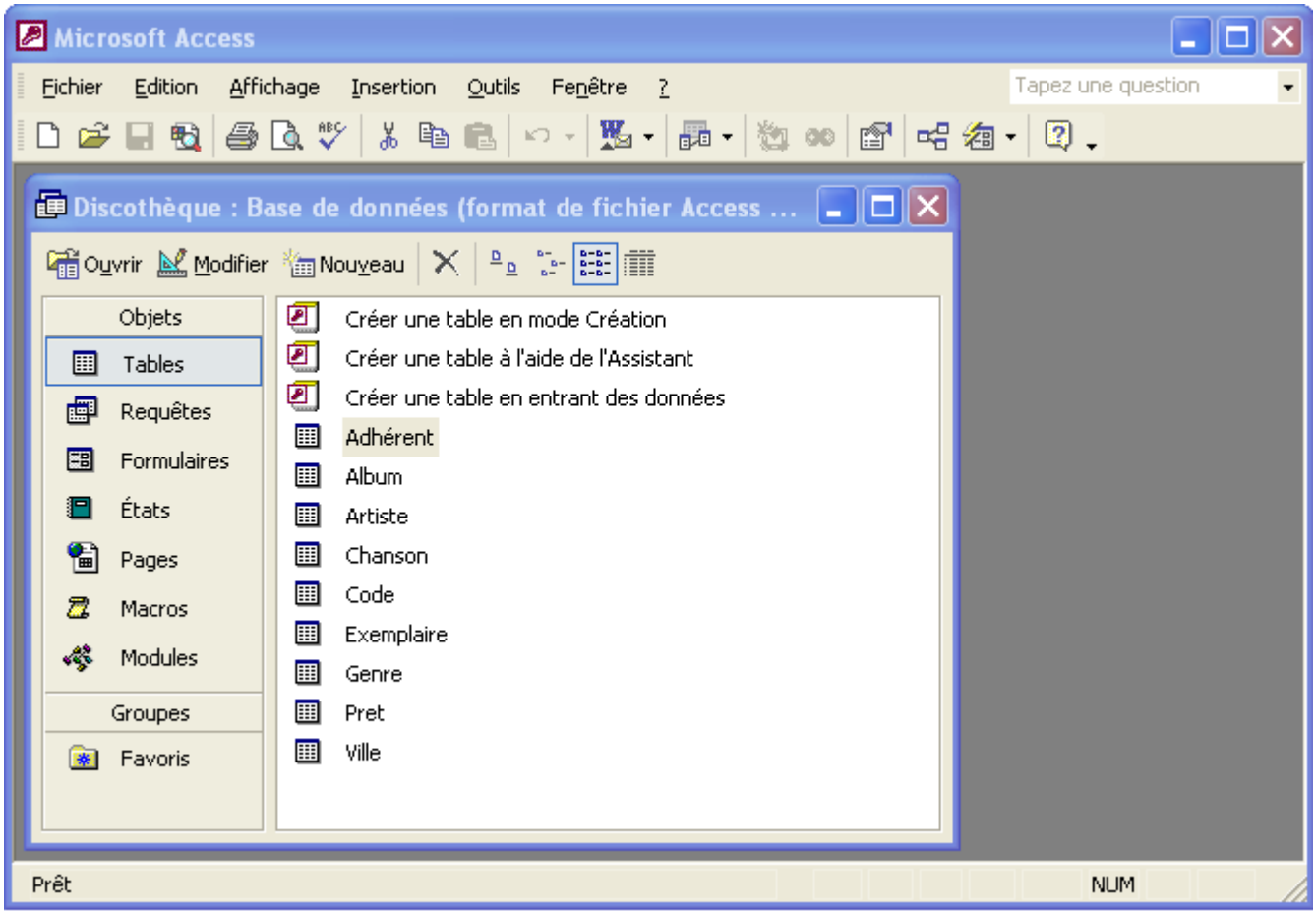


Figure 1 : Access XP et la base de données Discothèque ouverte

Chapitre 2 - Manipulation des objets de base avec Access

Après avoir abordé ces nécessaires préliminaires à notre étude, commençons l'analyse du sujet de ce dossier, c'est-à-dire la comparaison entre Access 97 et Access XP.

Pour cela nous étudierons tout d'abord la création de la base de données Discothèque en Access 97. Nous soulignerons ensuite les différences dans la création de cette même base de données sous la version XP pour mettre en évidence les particularités de Access XP.

I - Etude des travaux pratiques en Access 97

Afin de permettre une comparaison efficace, il est important de pouvoir mettre en évidence des éléments clés qui pourront servir de points de comparaison entre les 2 versions.
Nous utiliserons pour cela des exemples tirés des travaux pratiques du module de Base de Données d'IUP1.

Nous mettrons tout d'abord en avant les points caractéristiques de ces travaux pratiques puis nous nous intéresserons aux difficultés que nous avons rencontrées lors de la réalisation de ces travaux pratiques, car les éléments ainsi identifiés sont souvent intéressants.

I.1 - Eléments caractéristiques des travaux pratiques actuels

Nous allons étudier les travaux pratiques de base de données au travers de six points caractéristiques.

I.1.1 - La fenêtre de gestion de la base de données

Il s'agit de l'interface principale qui permet à l'utilisateur d'accéder à chacun des outils mis à disposition de l'utilisateur afin d'administrer sa base de données.

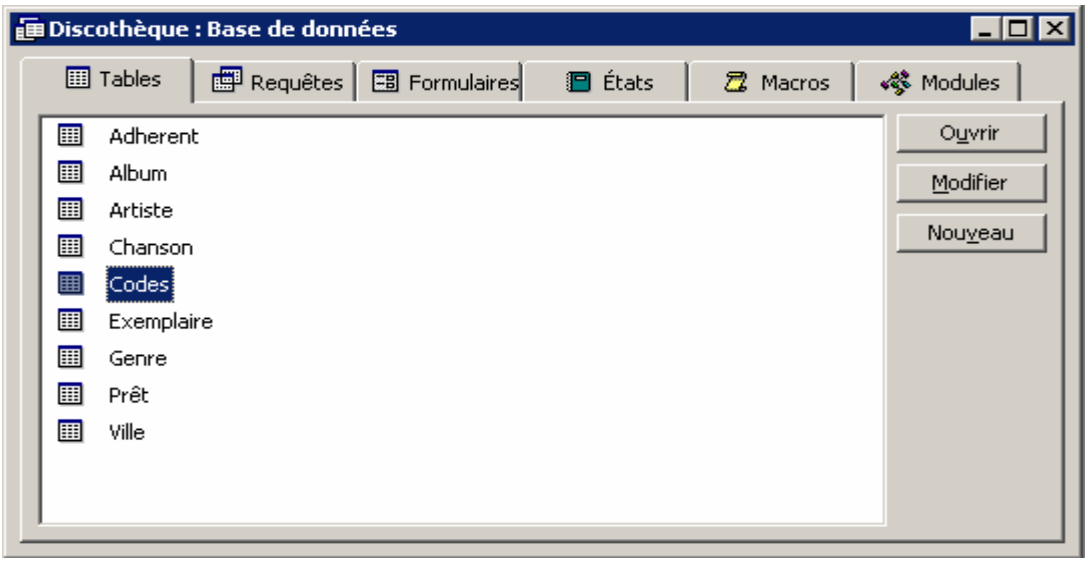


Figure 2 : La fenêtre de gestion de Discothèque sous Access 97

Nous voyons sur cette fenêtre de nombreux menus :

- Une barre horizontale contenant six onglets détaillés ci-après.
- Trois boutons sur le côté droit : Ouvrir, Modifier, Nouveau.

L'onglet horizontal contient six rubriques différentes :

- **Tables :** Cette rubrique permet d'accéder aux tables (ou relations). Ces tables sont les structures qui renferment les données selon un formalisme choisi.

  Par exemple, une table contiendra les noms, prénoms et date d'inscriptions des membres du club X.

- **Requêtes :** Cette rubrique permet de créer des requêtes (ou vues dans d'autres SGBD). Ces requêtes sont basées sur les tables. Elles consistent en une expression de critères et d'instructions qui lorsqu'elles sont interprétées par le moteur de base de données renvoient des informations pouvant provenir de plusieurs tables.

  Par exemple dans Discothèque, avec une table contenant des noms d'albums et des titres de chansons, et une autre contenant les mêmes noms d'albums et leurs interprètes, on peut connaître les titres des chansons interprétées par un interprète donné.

- **Formulaires :** Cette rubrique permet d'accéder aux formulaires. Ces formulaires sont la partie principale de l'interface utilisateur pour l'accès aux données. Ils permettent à l'administrateur de la base de données de créer des fenêtres permettant d'organiser un système d'information convivial et facile d'utilisation.

  On pense par exemple à un enchaînement de menus permettant d'accéder à des traitements de données spécifiques.

- **Etats :** Les états permettent d'imprimer ou de visualiser des données selon un cadre prédéfini.

  Par exemple la création automatique d'une facture, en fonction des informations qui la composent.

- **Macros et Modules :** Ces rubriques permettent le développement d'outils en Visual Basic. Il est parfois nécessaire de développer des modules à l'aide de ce langage lorsque certains traitements ne peuvent être obtenus par d'autres moyens.

Cette fenêtre est donc le point de départ de la base de données.

I.1.2 - La création d'une table

Les tables sont les structures de base de toute base de données. Une table est constituée de deux parties :

- Un en-tête précisant un ensemble fixé d'attributs
- Un corps formé par les enregistrements contenus dans la table, un enregistrement de la table étant une information qui respecte le formalisme dicté par l'en-tête.

Ceci peut sembler compliqué mais un exemple éclaircira tout ceci : examinons l'organisation de la table Album :

Figure 3 : La table Album en mode Feuille de données

On voit sur cette figure l'en-tête formé des attributs CodeAlbum, TitreAlbum, CodeArtiste et AnneeSortie, puis les enregistrements contenus dans la table : par exemple l'album qui a le CodeAlbum "10000" a pour titre "EarthLing" et pour CodeArtiste "Dabow" et est sorti en "1996".

Pour mettre en place une table, il est nécessaire de définir les attributs de son en-tête. Le mode création de table permet de réaliser cela. Ce mode permet de choisir pour chaque champ (chaque attribut de l'en-tête) quel type de données il peut contenir (entier, date, texte…), ce qui permettra à l'ordinateur d'effectuer des traitements sur les enregistrements contenus.

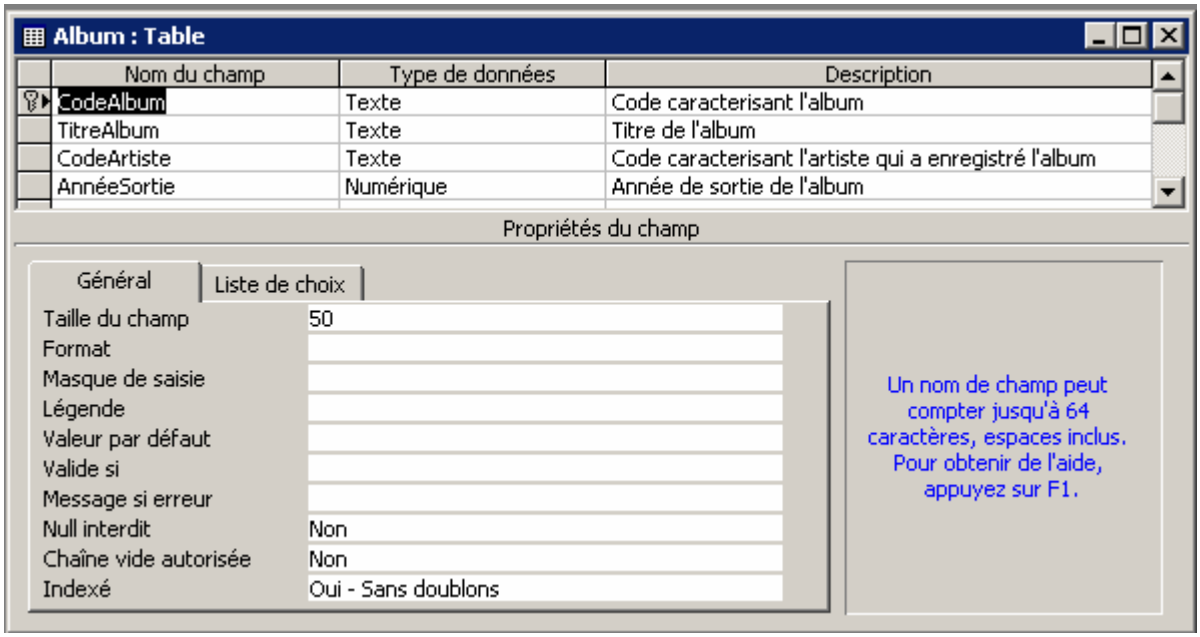


Figure 4 : La table Album en mode Création de table

Access 97 permet le choix parmi 8 types de données différents : *Texte, Mémo, Numérique, Date/Heure, Monétaire, Numérotation automatique, Oui/Non, Objet OLE.*

Il est aussi possible de créer des listes de choix qui permettent d'entrer les valeurs que pourra prendre le champ. *Par exemple un champ Sexe aura la liste de choix Masculin ou Féminin.*

La colonne description permet d'ajouter un descriptif bref et pertinent permettant la bonne compréhension de l'information.

Dans le bas de la fenêtre se trouve un cadran permettant de prédéfinir les propriétés de chaque champ :

- Taille du champs : permet de préciser la taille du champs.
- Format : cette propriété peut permettre au créateur de la base de données de mettre en forme l'affichage d'un champ.
- Masque de saisie : permet de formater les saisies c'est-à-dire d'obliger l'utilisateur à saisir un champ dans un format choisi. Par exemple il suffit de remplir ce champ par LL000 pour que la saisie ce fasse si le champ est de type 2 lettres suivies de 3 chiffres.
- Légende : titre de colonne affichée sur la table.
- Valeur par défaut : précise la valeur par défaut.
- Valide si : permet d'ajouter un paramètre à la validité du champ saisi. Par exemple si le champ est de type Date, on peut empêcher une saisi antérieure à la date courante.
- Message erreur : permet l'affichage d'une boite de dialogue lorsque la saisie est incorrecte, ce qui s'avère très utile pour renseigner l'utilisateur quant aux propriété du champ en question.
- Null interdit : autorise ou non l'absence d'un champ.
- Chaîne vide autorisée : autorise des chaînes nulles dans le champ.
- Indexé : permet de créer des champs indexés qui accélèrerons la recherche lors d'une requête.

I.1.3 - La création des requêtes avec l'assistant

Les requêtes (ou vues dans d'autres SGBD) sont des expressions spécifiant des critères de sélection. Leur exécution par le moteur de base de données permet de visualiser les enregistrements correspondants aux critères fournis.

Prenons l'exemple de la requête RTitres73 dans Discothèque ; son exécution permet d'afficher pour chaque chanson enregistrée en 1973 son titre et le nom de son auteur.

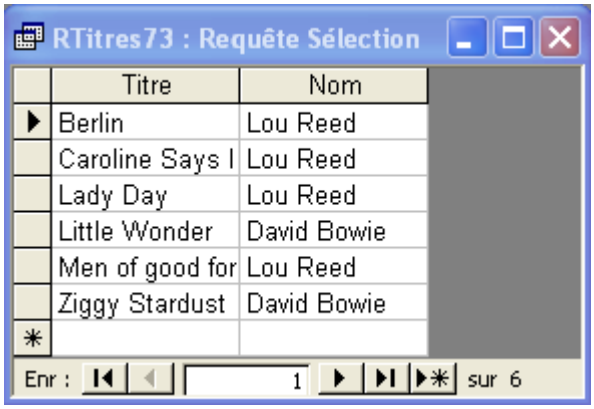


Figure 5 : Résultat de la requête RTitres73

Afin de créer ces requêtes, c'est-à-dire de spécifier les critères de sélection, Access 97 nous propose 2 modes :

- Le mode **QBE** (Query by example) qui est un assistant graphique de création. Cet assistant permet de définir quelles tables entrent en jeu dans la requête, de préciser quels champs de ces tables sont utilisés et avec quels paramètres.

  Par exemple, sur la figure 6 on voit dans la colonne du milieu qu'on extraira de la table Album les albums sortis en 1973.

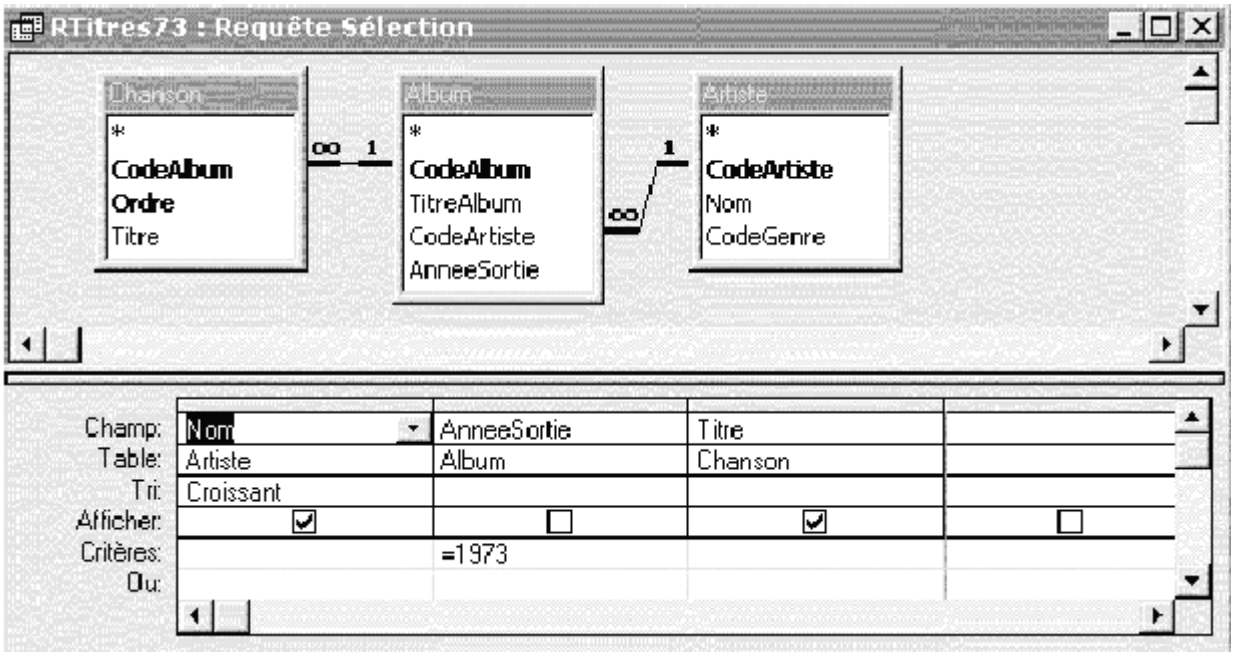


Figure 6 : La requête RTitres73 créée avec l'assistant QBE

- Le mode **SQL** (Structured Query Language) qui est une fenêtre permettant de saisir directement sa requête au clavier en suivant les spécifications du langage SQL[1].

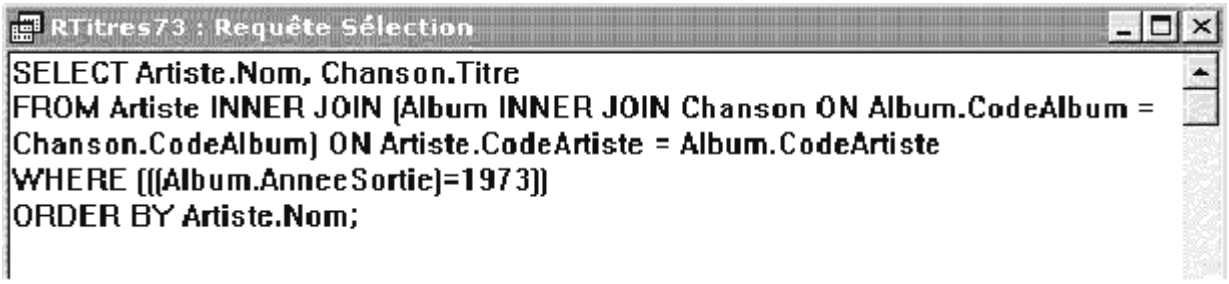


Figure 7 : La requête RTitres73 en mode SQL

Lors de la création d'une requête à l'aide de l'assistant QBE, Access génère lui-même le code SQL correspondant. Cependant, on remarque que sous Access 97 le code ainsi généré est encombré de nombreuses parenthèses inutiles (Figure 7) gênantes pour la compréhension du code.

I.1.4 - La création d'un formulaire ayant pour source une requête

Un formulaire est une interface que l'administrateur de la base de données crée afin de permettre à l'utilisateur de naviguer dans la base de données. Ces formulaires peuvent servir à visualiser les résultats d'une requête : ce sont alors des formulaires ayant pour source une requête.

Pour réaliser ce type de formulaire, il faut importer la requête renvoyant les résultats désirés lors de la création du formulaire.

[1] SQL est le langage standard des SGBD relationnels. C'est IBM qui a commencé à le développer au milieu des années 70. Il répond au besoin d'établir une norme pour la création des requêtes quelque soit le SGBD utilisé.

Pour cela il faut d'abord enregistrer dans la base de données la requête que l'on souhaite. Nous allons ensuite créer notre formulaire en mode création ou grâce à l'assistant de création. Une fois le formulaire créé, il faut cliquer sur les propriétés du formulaire et sélectionner notre requête dans le champ « Source ».

Dans notre cas on sélectionnera la requête RInfoAdhérents de la base de données Discothèque afin de créer le formulaire Adhérents qui renseigne l'utilisateur de la base de données sur les adhérents à la discothèque.

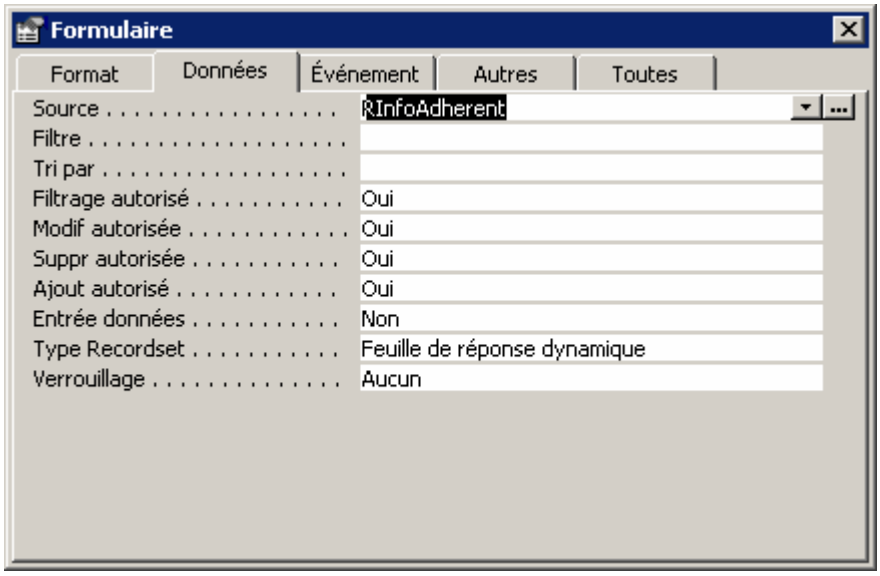


Figure 8 : Création d'un formulaire ayant pour source une requête : liaison avec la requête

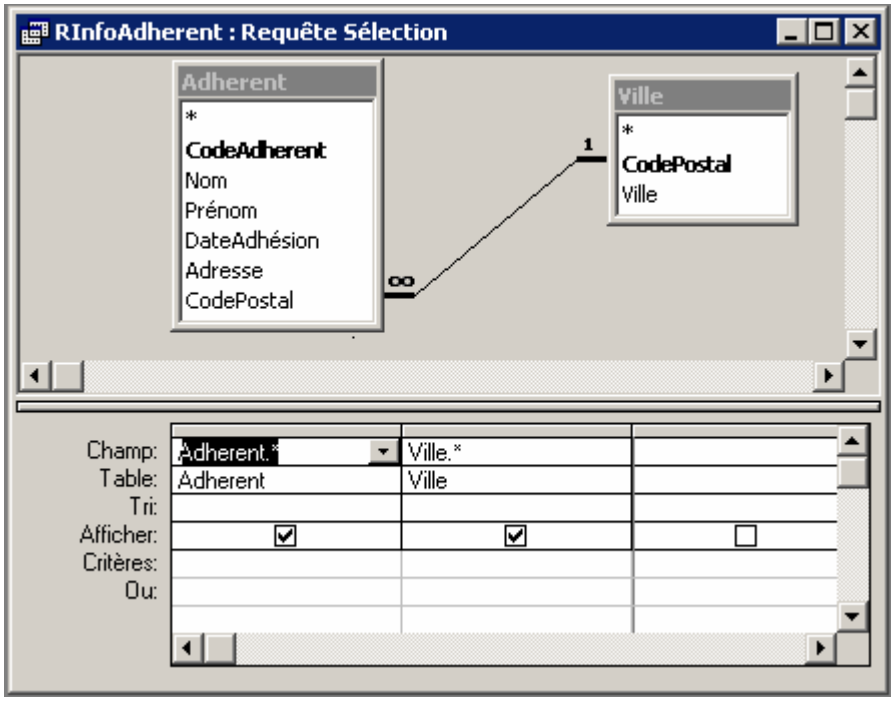


Figure 9 : La requête RInfoAdhérents

Désormais le formulaire dépend de la requête RInfoAdhérents. Maintenant si l'on désire par exemple afficher sur le formulaire un champ dépendant de la requête, il faut cliquer sur les propriétés du champ et sélectionner la source de contrôle dans « l'onglet Données ».

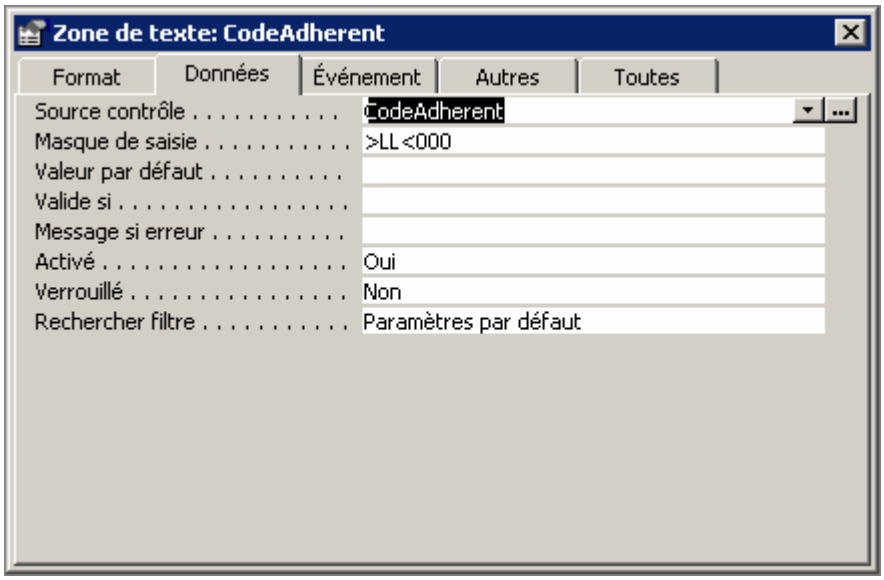


Figure 10 : Zone de texte liée à la requête

I.1.5 - La création d'un sous-formulaire

Un sous-formulaire est un formulaire dans un formulaire. Le formulaire primaire est appelé formulaire principal et le formulaire qu'il contient sous-formulaire. Les sous-formulaires sont particulièrement utiles lorsque l'on veut afficher des données de tables ou de requêtes qui ont une relation un-à-plusieurs ; c'est le cas des chansons d'un album.

Pour créer un sous formulaire, il faut insérer dans le formulaire père une zone de sous-formulaire en utilisant la boite à outils :

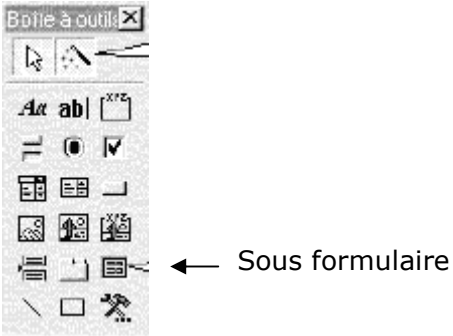


Figure 11 : Boite à outils d'Access 97

Un assistant de création de sous-formulaire s'ouvre alors, demandant d'indiquer tout d'abord la provenance des données du sous-formulaire puis quels champs doivent participer au lien entre les deux formulaires.

Il est possible en Access 97 de créer jusqu'à deux niveaux de sous-formulaires, c'est-à-dire qu'un sous-formulaire peut être formulaire principal d'un second sous-formulaire.

I.1.6 - La création des états et des sous-états

Les sous-états se font de manière similaire à la création de sous-formulaires en utilisant un champ sélectionné à partir de la boite à outils et en le mettant sous la dépendance de l'état désiré.

I.2 - Points difficiles des travaux pratiques actuels

Lors de la réalisation de ces travaux pratiques nous avons relevé plusieurs points difficiles que nous avons choisi de détailler afin de voir si le changement de version contribuait à une réalisation plus aisée de ces points.

Nous avons relevé des difficultés importantes lors de :

- La définition des structures de table, en particulier pour établir les masques de saisie sur les champs.
- La création d'un formulaire lié à un sous formulaire qui reste un point difficile même avec l'utilisation de l'assistant.
- La mise en place des états qui est délicate même avec l'utilisation de l'assistant.

II - Migration des travaux pratiques vers Access XP

Nous allons dans cette partie nous intéresser tout d'abord à l'adaptation des travaux pratiques sous Access XP afin de mettre en valeur les différences entre les deux versions, puis nous aborderons les nouvelles fonctionnalités disponibles dans Access XP.

II.1 – Différences relevées au cours de l'adaptation

Nous reprendrons dans cette partie le même plan que dans la partie I.1, un plan en six points qui nous permettra une comparaison efficace des deux versions d'Access.

II.1.1 – La fenêtre de gestion de la base de données

Tout comme sous Access 97, il s'agit du point de départ de la base de données.

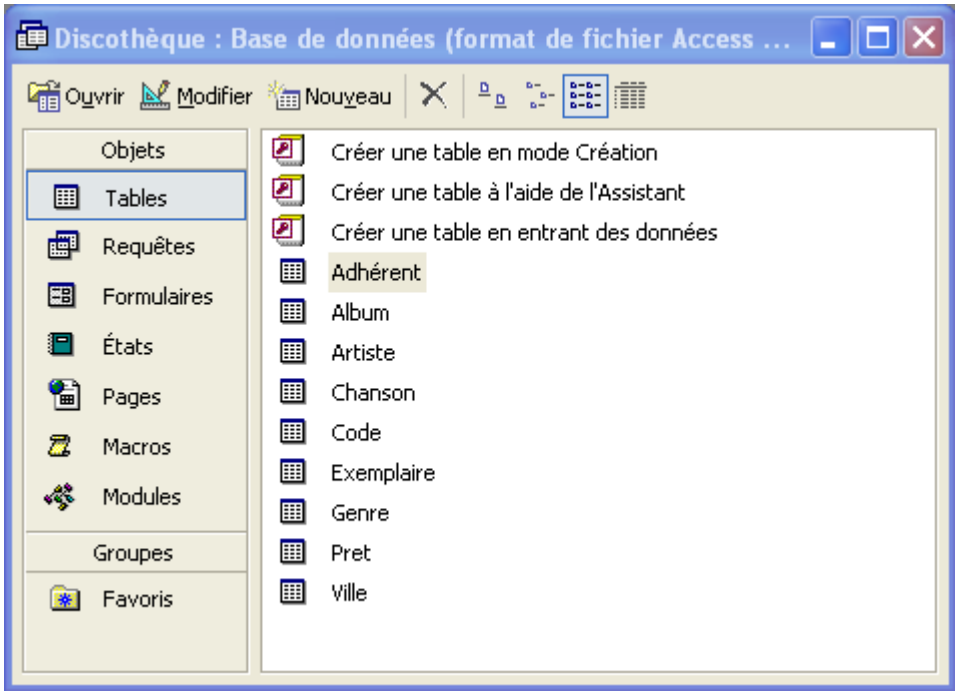


Figure 12 : La fenêtre de gestion de Discothèque sous Access XP

Nous voyons sur cette fenêtre de nombreux menus :

- Une barre verticale contenant deux sous menus : Objets et Groupes.
- Une barre horizontale dont le contenu change en fonction de l'objet choisi dans la barre verticale.

Le menu Objets contient sept rubriques différentes :

- **Tables :** Cette rubrique permet d'accéder aux tables (ou relations). Ces tables sont les structures qui renferment les données selon un formalisme choisi.

  Par exemple, une table contiendra les noms, prénoms et date d'inscriptions des membres du club X.

- **Requêtes :** Cette rubrique permet de créer des requêtes (ou vues dans d'autres SGBD). Ces requêtes sont basées sur les tables. Elles consistent en une expression de critères et d'instructions qui lorsqu'elles sont interprétées par le moteur de base de données renvoient des informations pouvant provenir de plusieurs tables.

  Par exemple dans Discothèque, avec une table contenant des noms d'albums et des titres de chansons, et une autre contenant les mêmes noms d'albums et leurs interprètes, on peut connaître les titres des chansons interprétées par un interprète donné.

- **Formulaires :** Cette rubrique permet d'accéder aux formulaires. Ces formulaires sont la partie principale de l'interface utilisateur pour l'accès aux données. Ils permettent à l'administrateur de la base de données de créer des fenêtres permettant d'organiser un système d'information convivial et facile d'utilisation.

  On pense par exemple à un enchaînement de menus permettant d'accéder à des traitements de données spécifiques.

- **Etats :** Cette rubrique permet d'imprimer ou de visualiser des données selon un cadre prédéfini.

  Par exemple la création automatique d'une facture, en fonction des informations qui la composent.

- Pages : Cette rubrique permet de créer depages interactives d'accès aux données, afin de pouvoir visualiser ces données sur un site Internet ou intranet.

  Par exemple, les pages permettront la consultation en ligne d'une facture.

- **Macros et Modules :** Ces rubriques permettent le développement d'outils en Visual Basic. Il est parfois nécessaire de développer des modules à l'aide de ce langage lorsque certains traitements ne peuvent être obtenus par d'autres moyens.

La barre verticale des menus permet des actions sur les objets des rubriques précédentes : Ouvrir, Modifier, Nouveau…

Cette fenêtre est donc relativement similaire à celle d'Access 97, hormis un changement d'aspect et l'ajout de la rubrique Pages.

II.1.2 - La création d'une table

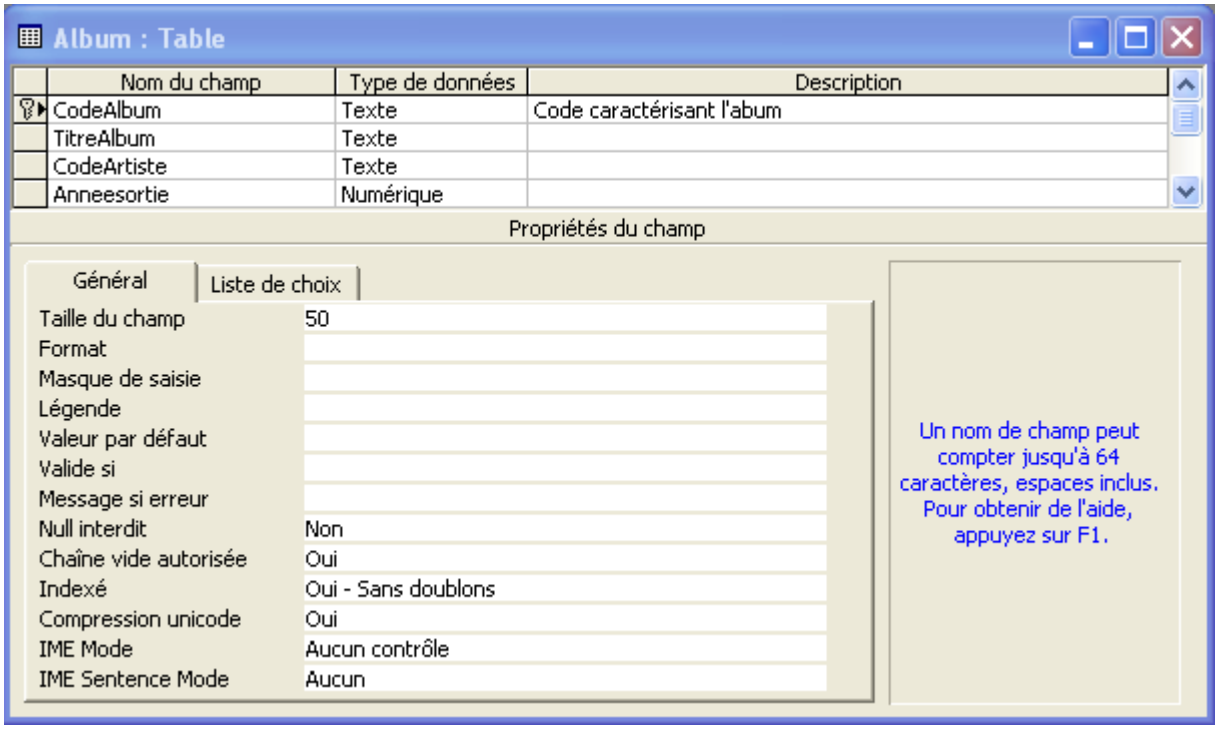


Figure 13 : La table Album en mode Création de table

Access XP permet le choix parmi 9 types de données différents : *Texte, Mémo, Numérique, Date/Heure, Monétaire, Numérotation automatique, Oui/Non, Objet OLE, Lien hypertexte.*

Les propriétés des champs proposés sont similaires à ceux trouvés dans Access 97, à l'exception de trois nouvelles propriétés :

- Compression Unicode : Dans Access 97 chaque caractère est codé sur un octet alors que dans Access 2000 et ultérieur un caractère est codé sur deux octets. Pour compensé cette augmentation de taille et rétablir des performances optimales Access XP possède cette propriété qui compressera les caractères.
- IME mode : permet de rentrer des données dans une langue asiatique.
- IME Sentence Mode : permet de rentrer des données dans une langue asiatique.

On remarquera que l'établissement des masques de saisie qui était un point difficile sous Access 97 est facilité sous Access XP grâce à la présence d'exemples aidant à leur mise en place.

Comme on peut le constater, les différences entre Access 97 et Access XP sont minimes au niveau de la création des tables.

II.1.3 - La création des requêtes avec l'assistant

Le mode QBE pour la création des requêtes est similaire à celui d'Access 97.

Nous allons utiliser le même exemple que précédemment pour illustrer notre propos: la requête RTitres73.

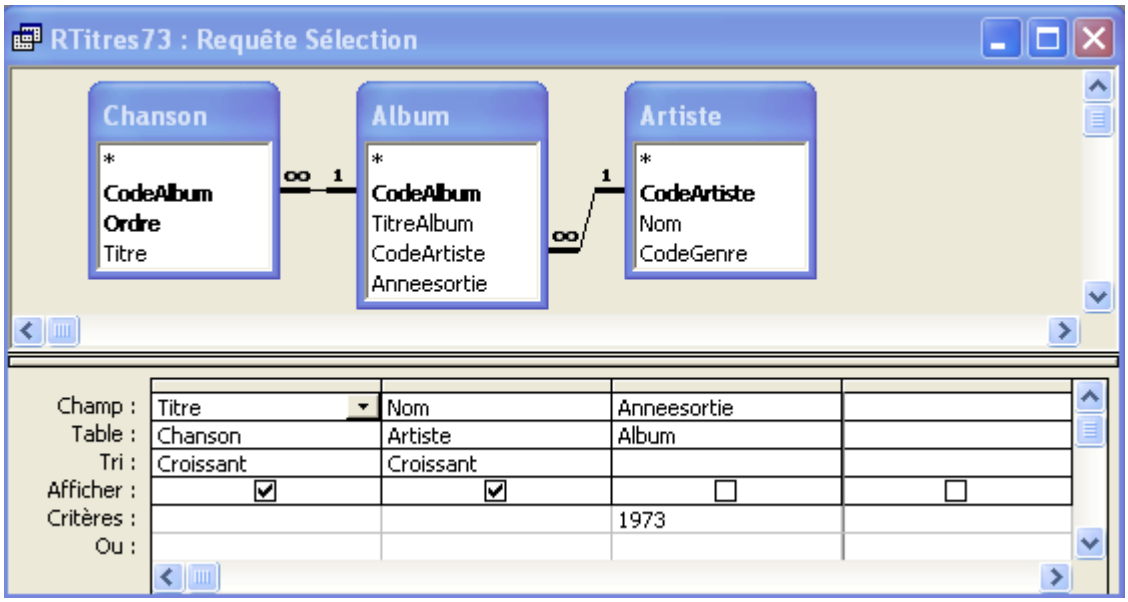


Figure 14 : La requête RTitres73 en mode QBE sous Access XP

Une fois la requête créée grâce au mode QBE, basculons en mode SQL afin d'examiner la syntaxe du code généré.

Figure 15 : La requête RTitres73 en mode SQL sous Access XP

On remarquera que le code SQL généré est moins encombré de parenthèses inutiles et ainsi plus compréhensible.

C'est le seul changement notable dans la création des requêtes entre Access 97 et Access XP.

II.1.4 - La création d'un formulaire ayant pour source une requête

Nous allons reprendre l'exemple utilisé au paragraphe I.1.4 pour illustrer la création d'un formulaire ayant pour source une requête.

Après avoir créé le formulaire en question, il faut le lier avec la requête en utilisant la propriété source des propriétés du formulaire.

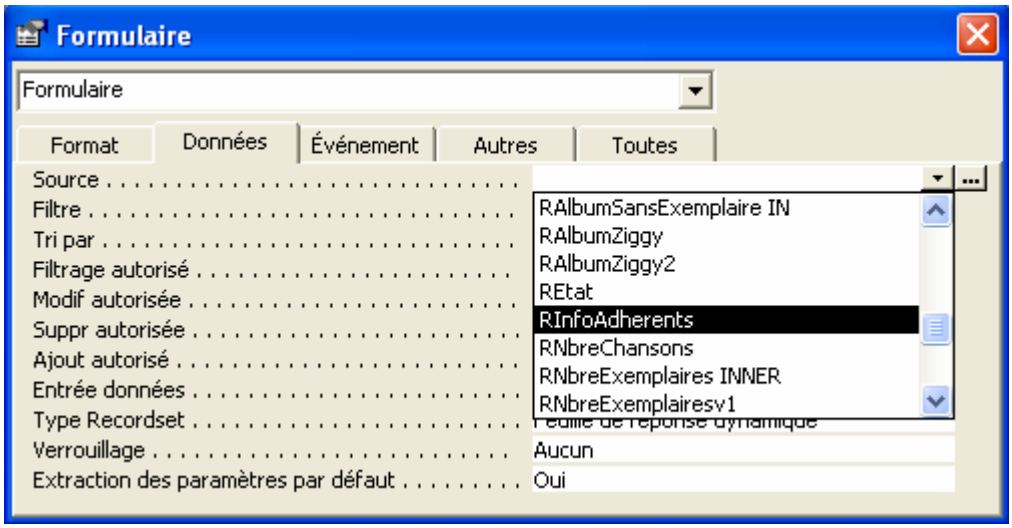


Figure 16 : Création d'un formulaire ayant pour source une requête : liaison avec la requête

Désormais le formulaire dépend de la requête RInfoAdhérents. Maintenant si l'on désire par exemple afficher sur le formulaire un champ dépendant de la requête, il faut cliquer sur les propriétés du champ et sélectionner la source de contrôle dans « l'onglet Données ».

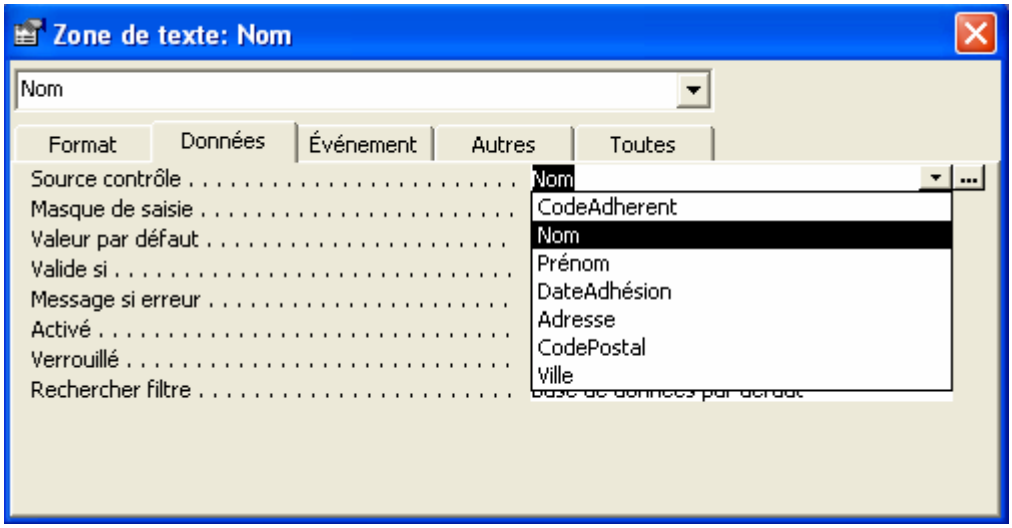


Figure 17 : Zone de texte liée à la requête

Le processus de création d'un formulaire ayant pour source une requête est donc identique sous Access 97 et XP. Ce point reste donc un point difficile sous Access XP.

II.1.5 - La création d'un sous-formulaire

La création d'un sous-formulaire lié à un formulaire en mode création avec Access XP se fait de manière aisée. Il faut premièrement insérer un champ de sous-formulaire en le sélectionnant dans la boîte à outil comme le montre la figure ci dessous :

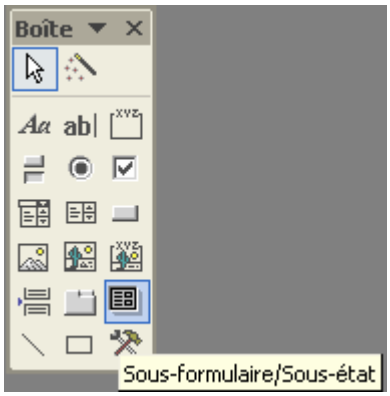


Figure 18 - Sélection d'un champ de sous-formulaire

Après insertion du champ du sous-formulaire il faut sélectionner les propriétés de ce champ et sélectionner le formulaire que l'on souhaite insérer.

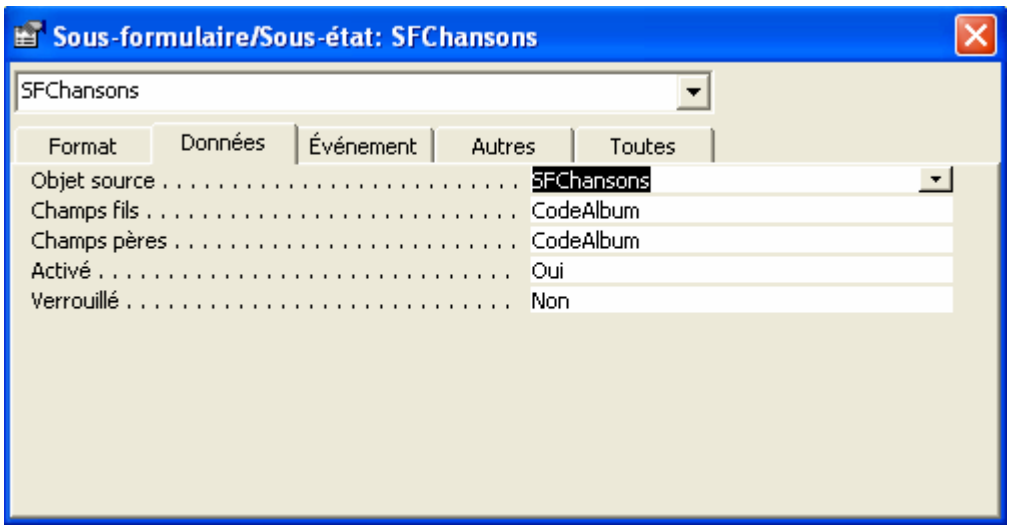


Figure 19 - Propriétés d'un sous-formulaire

Puis pour modifier les champs père fils, il faut sélectionner les propriétés du sous-formulaire puis choisir soit le champs fils soit le champ père pour que la boîte de dialogue qui édite les liens des champs des sous-formulaires apparaisse comme le montre la figure ci dessous. Puis il ne reste qu'à choisir les champs que l'on désire mettre en relation.

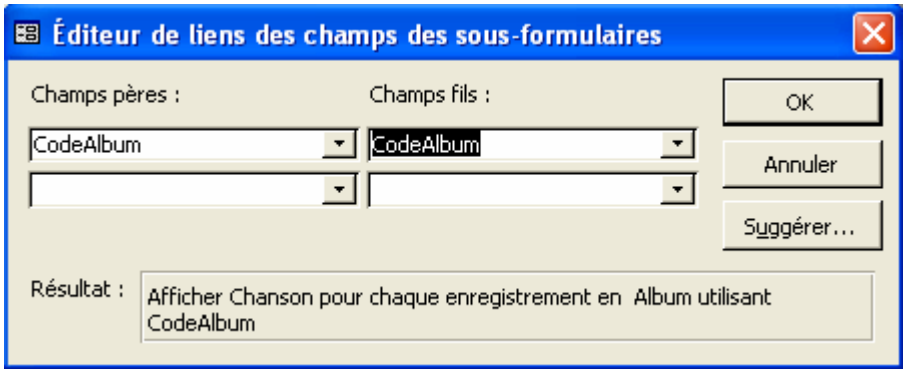


Figure 20 - Editeur de liens des champs d'un sous-formulaire

Remarque : un formulaire principal peut contenir autant de sous-formulaires que l'on souhaite à condition qu'ils soient placés dans le formulaire principal. Par contre il est

possible d'imbriquer sept sous-formulaires les uns dans les autres, contre deux en Access 97.

II.1.6 - La création des états et des sous-états

Tout comme sous Access 97, la création des états et sous-états est similaire à celle des formulaires et sous-formulaires. L'assistant de création des états a été amélioré sous Access XP rendant leur mise en place plus aisée.

II.2 - Nouveautés de Access XP

Cette partie va nous permettre de présenter les éléments introduits par Microsoft dans le logiciel Access XP qui n'existaient pas dans Access 97. Nous avons relevé trois innovations majeures dans Access XP.

II.2.1 - Liens hypertextes

Les liens hypertextes sont une source importante d'interactivité pour l'utilisateur. Sous Access les liens hypertextes ont été largement étudiés pour répondre aux attentes du créateur de la base. Ainsi il est possible de faire des liens hypertextes vers tous éléments de la base courante (tables, requêtes, formulaires, états, modules …), d'une base extérieure mais aussi vers des documents offices et aussi vers des sites Internet et des adresses Email. De plus leurs mises en place sont aisées : il suffit de cliquer sur l'onglet « Insertion » et sélectionner « Lien Hypertexte ». Une boite de dialogue s'affiche et nous guide dans le choix de la direction.

Mais il est également possible de faire des liens sur des éléments d'un formulaire, d'un état, d'une table… (zone de texte, champ). Pour cela il ne faut pas omettre d'activer la propriété « Is Hyperlink » comme le montre la figure ci-dessous :

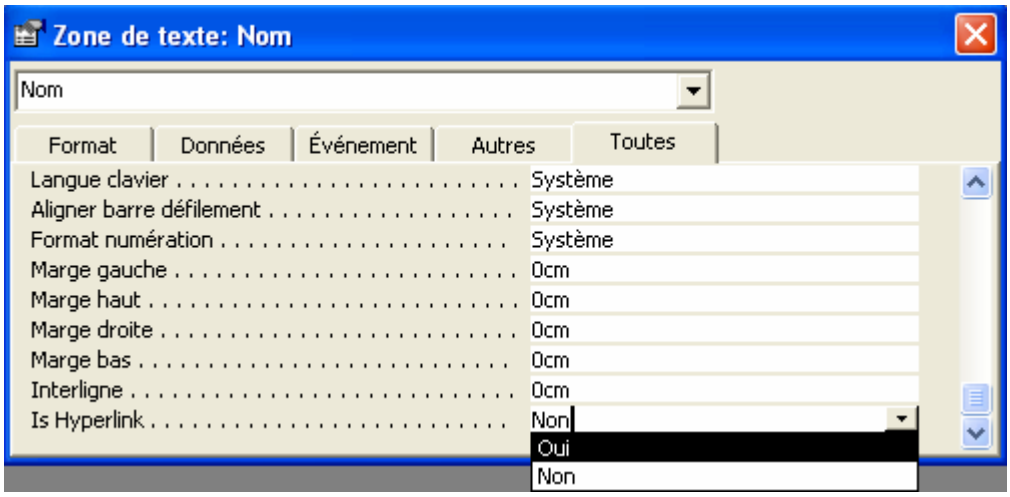


Figure 21 : Liens hypertextes

Il est à noter que les liens créés sur un état ne sont pas visibles sous Access mais le sont sous d'autres logiciels Office (Word, Excel) après exportation.

II.2.2 - Publication de page WEB

Une des grandes nouveautés de Access XP par rapport à Access 97 est la publication de pages Web dynamiques ou plutôt devrait on dire « pages d'accès aux données ». Ces pages sont des pages HTML liées aux données, visualisables à l'aide de Microsoft Internet Explorer version 5 ou ultérieure. On peut également distribuer par messagerie électronique des pages de ce type contenant des données statiques ou dynamiques. Les pages d'accès aux données sont en général destinées à une utilisation sur intranet, mais il est également possible de les déployer sur Internet, en tenant compte toutefois de certaines considérations. Ainsi les pages d'accès aux données offrent les mêmes possibilités qu'un formulaire ou un état : on peut consulter, modifier, insérer des données dans une base de données installée sur le serveur. Néanmoins, on peut également utiliser une page en dehors d'une base de données Microsoft Access, de manière à ce que les utilisateurs puissent mettre à jour ou afficher des données sur Internet ou un réseau intranet.

Attention : les pages d'accès aux données ne sont pas enregistrées dans la base mais dans un dossier annexe dont on spécifie l'emplacement. Mais les pages restent accessible à partir de la base car Access crée automatiquement un lien dans la fenêtre de gestion. Il ne faut surtout pas supprimer ce lien si l'on désire garder l'accès à cette page. Malheureusement le chemin d'accès à ces pages est absolu et non relatif.

II.2.2.1 – Les différentes manières de créer une page d'accès aux données

Création en mode création :

Pour créer une page d'accès aux données en mode création, il faut cliquer sur pages dans la fenêtre de gestion de la base de données et sélectionner « Créer une page d'accès aux données en mode création » comme le montre la figure n° 12 :

Figure 22 - Création d'une page d'accès aux données

Apparaît alors une page d'accès aux données vide, une boîte à outils et une fenêtre contenant tous les champs de la base courante :

Figure 23 - Interface pour la création d'une page d'accès aux données

Création à partir d'un objet existant :

Mais avec la version d'Access 2002 il est désormais possible de créer une page d'accès aux données à partir d'un objet existant. Il suffit d'enregistrer par exemple un formulaire au format « page d'accès aux données » en cliquant sur l'onglet « Fichier » puis « Enregistrer sous » :

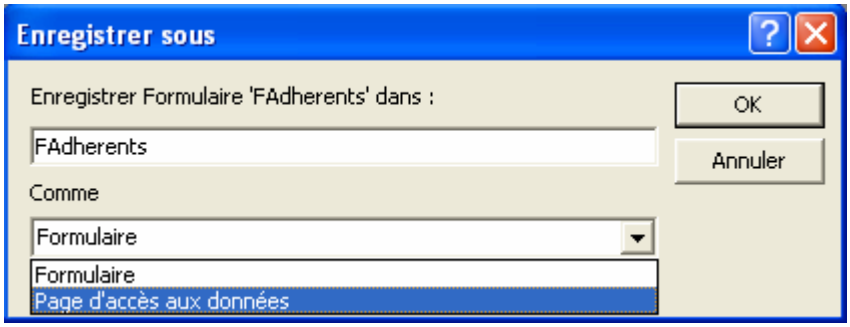


Figure 24 - Fenêtre d'enregistrement

On obtient une page HTML de ce type :

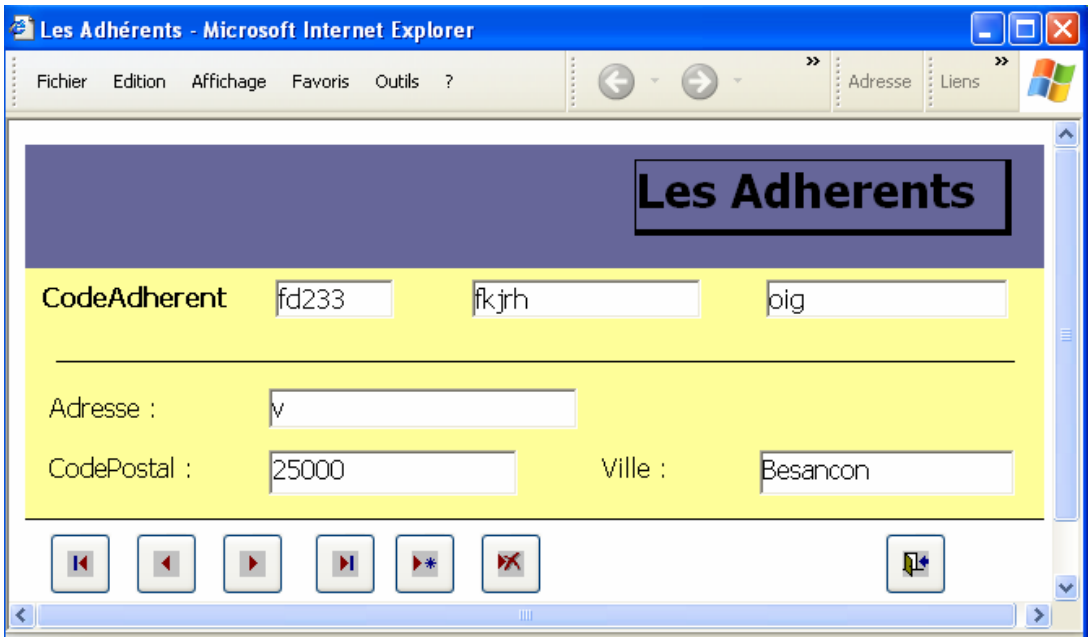


Figure 25 - Aperçu d'une page d'accès au données avec Internet Explorer

Pour la suite de l'étude des pages d'accès aux données, nous traiterons le cas en mode création.

II.2.2.2 – Etude des différentes fenêtres disponibles lors de la création d'une page d'accès aux données en mode création :

La boîte à outil :

On remarque que la boîte à outil est plus fournie que celle d'un formulaire. Voici les options supplémentaires :

- Etendue : permet un autre mode d'affichage d'un texte.
- Texte défilant : permet d'insérer un texte défilant sur la page.
- Développer : permet l'insertion d'une image.
- Déplacement entre les enregistrements : permet d'insérer une barre d'outils qui manipule les enregistrements.
- Tableau croisé dynamique office : permet l'insertion d'un tableau croisé dynamique.
- Graphique office : permet l'insertion d'un graphique Office.
- Feuille de calcul Office : permet l'insertion d'une feuille de calcul office.
- Image - Lien hypertexte : permet d'insérer une image qui sera réactive et fonctionnera comme un lien hypertexte.
- Film : permet d'insérer une vidéo sur la page.

La plupart de ces options ont pour but d'agrémenter la page en décors mais également en utilisation. Ces fonctions facilitent la transformation d'une page d'accès aux données en une véritable page HTML à la fois pratique et interactive.

La liste des champs :

La liste des champs est simplement un explorateur de tous les champs disponibles dans la base courante. Il suffit de choisir le champ désiré et de le glisser sur la page d'accès aux données en construction pour l'y insérer.

La page d'accès aux données :

Le page d'accès aux données est composé de différentes parties :

- Le corps : le corps constitue la surface de création de base d'une page d'accès aux données sur laquelle on peut insérer une infinité de section ainsi que d'agréments tel qu'un texte défilant, une image, une vidéo…
- Les sections : elles sont utilisées pour afficher du texte, des données d'une base de données et des barres d'outils.
    - **En-tête et pied de groupe** : utilisés pour afficher des données et calculer des valeurs.
    - **Déplacement entre les enregistrements** Utilisée pour afficher le contrôle déplacement entre les enregistrements pour le niveau de regroupement. Une section de déplacement entre les enregistrements d'un groupe apparaît après la section d'en-tête de groupe. Il est impossible de placer des contrôles dépendants dans une section de déplacement entre les enregistrements.
    - **Légende** : elle permet l'affichage des légendes de zones de texte et autres contrôles. Elle apparaît juste devant l'en-tête de groupe.

Chaque niveau de regroupement d'une page d'accès aux données possède une source d'enregistrements. Le nom de cette dernière est affiché dans la barre de section pour chaque section utilisée pour un niveau de regroupement.

II.2.2.3 - Différents types de Page d'accès aux données

États interactifs

Ce type de page d'accès aux données est souvent utilisé pour consolider et grouper des informations stockées dans la base de données, puis pour publier des synthèses des données. Bien que la page d'accès aux données puisse également contenir des boutons de barre d'outils pour le tri et le filtrage des données, il n'est pas possible de modifier des données sur ce type de page.

Saisie de données

Ce type de page d'accès aux données est utilisé pour afficher, ajouter et modifier des états.

Analyse des données

Ce type de page d'accès aux données peut comprendre une liste de tableau croisé dynamique, semblable à un formulaire de tableau croisé dynamique ACCESS© ou à un rapport de tableau croisé dynamique EXCEL©, qui permet de réorganiser les données afin de les analyser de diverses manières. Cette page pourrait contenir un graphique qui peut être utiliser pour analyser des tendances, détecter des modèles et comparer des données dans la base de données. Elle pourrait également contenir une feuille de calcul

dans laquelle des données peuvent être entrer et modifier, et utiliser des formules de calcul comme dans EXCEL©.

II.2.2.4 - Utilisation des Pages d'accès aux données

Une page d'accès aux données est directement connectée à une base de données. Lorsque les utilisateurs affichent la page d'accès aux données dans Internet Explorer, ils affichent leur propre copie de la page.

Cela signifie que tout filtrage, tri et autre modification qu'il apportent à la manière dont les données s'affichent (*notamment les modifications qu'ils apportent dans un liste de tableau croisé dynamique ou une feuille de calcul*) n'affectent que leur copie de la page d'accès aux données.

Toutefois, les modifications qu'ils apportent aux données elles-mêmes (*telles que la modification de certaines valeurs et l'ajout ou la suppression de données*) sont stockées dans la base de données sous-jacente et sont donc accessibles à toute personne consultant la base de données.

Mais les pages d'accès aux données peuvent également être utilisées à l'intérieur d'une base de données en tant que formulaire ou état. Le choix réside dans les tâches que le concepteur veut réaliser. Voici les avantages des pages d'accès aux données par rapport aux états :

- Les pages dépendantes de données affichent les données en cours car elles sont connectées à une base de données.
- Les pages sont interactives. Les utilisateurs peuvent filtrer, trier et afficher uniquement les enregistrements qui les intéressent.
- Les pages peuvent être diffusées électroniquement par courrier électronique. Les destinataires voient des données actualisées chaque fois qu'ils ouvrent le message.

II.2.3 - Prise en charge XML

Access suit la vague d'Internet. Pour la première fois, Access XP prend en charge le format de fichier XML (eXtensible Markup Language).

XML est un langage de description qui permet de définir d'autres langages de description de page. Un des formats, s'appuyant sur XML, est SVG, un format vectoriel qui joue un rôle prépondérant sur Internet.

XML peut être utilisé comme format de fichier universel. Il suffit d'enregistrer les données, correctement structurées, et XML s'occupe de leur description.

Les données XML offrent un énorme avantage : elles sont indépendantes de la plate-forme, ce qui permet de les lire sous Windows mais aussi sous d'autres systêmes tels que Linux, MacOs ou BeOs.

Nous allons étudier l'exportation au format XML de la table Album de Discothèque :

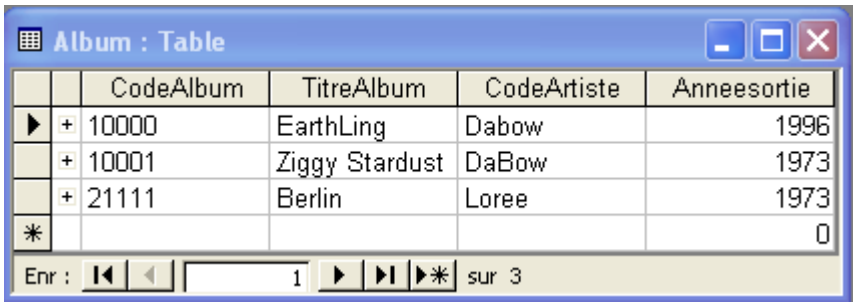


Figure 26 : La table Album en mode feuille de données

Une fois la table Album ouverte, il faut choisir "Exporter" dans le menu Fichier. La fenêtre suivante s'ouvre alors :

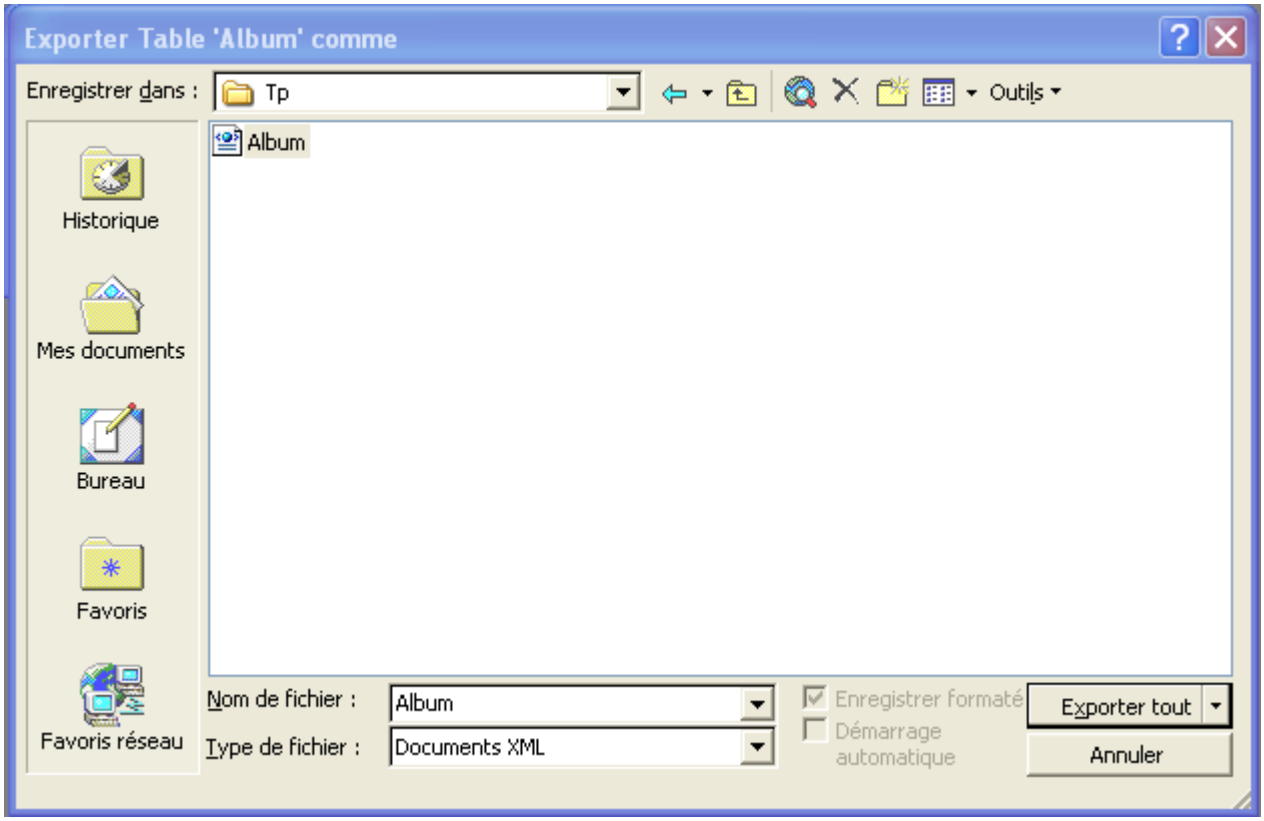


Figure 27 : Fenêtre d'exportation d'Access XP

Il faut ensuite choisir "Documents XML" dans la zone Type de fichier, puis "Exporter tout"

Figure 28 : Fenêtre d'exportation XML

Par défaut l'exportation crée deux fichiers :

- Un fichier XML contenant les données de la table à l'intérieur des balises XML
- Un fichier XSD contenant les définitions des balises XML

Il est néanmoins possible d'inclure les définitions des balises dans le fichier XML grâce au bouton "Avancé" qui ouvre la fenêtre suivante :

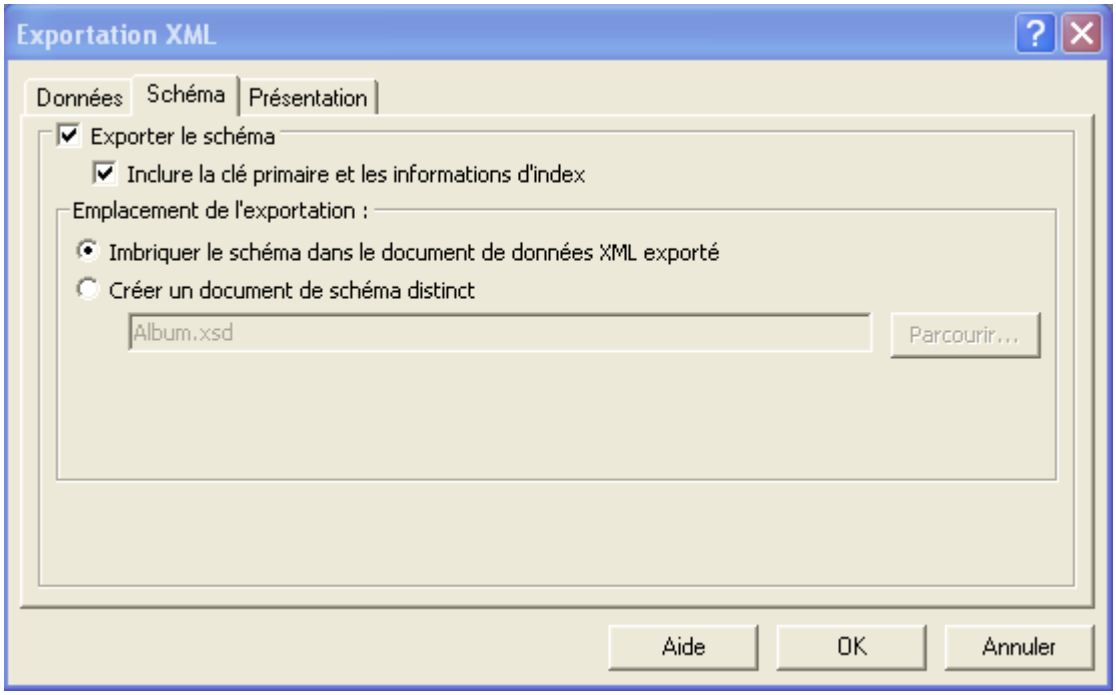


Figure 29 : Propriétés de l'exportation XML

On choisira alors dans l'onglet "Schéma" le bouton "Imbriquer le schéma dans le document XML exporté".

On obtient alors le fichier XML suivant, découpé ici pour des raisons de lisibilité en une première partie de définition des balises (document de schéma) et une seconde partie renfermant les données au sein des balises prédéfinies (document de données).

```
<?xml version="1.0" encoding="UTF-8"?>
<root xmlns:xsd="http://www.w3.org/2000/10/XMLSchema" xmlns:od="urn:schemas-microsoft-com:officedata">
<xsd:schema>
<xsd:element name="dataroot">
<xsd:complexType>
<xsd:choice maxOccurs="unbounded">
<xsd:element ref="Album"/>
</xsd:choice>
</xsd:complexType>
</xsd:element>
<xsd:element name="Album">
<xsd:annotation>
<xsd:appinfo>
<od:index index-name="PrimaryKey" index-key="CodeAlbum " primary="yes" unique="yes" clustered="no"/>
<od:index index-name="ArtisteAlbum" index-key="CodeArtiste " primary="no" unique="no" clustered="no"/>
<od:index index-name="CodeAlbum" index-key="CodeAlbum " primary="no" unique="no" clustered="no"/>
<od:index index-name="CodeArtiste" index-key="CodeArtiste " primary="no" unique="no" clustered="no"/>
</xsd:appinfo>
</xsd:annotation>
<xsd:complexType>
<xsd:sequence>
<xsd:element name="CodeAlbum" minOccurs="0" od:jetType="text" od:sqlSType="nvarchar">
<xsd:simpleType>
<xsd:restriction base="xsd:string">
<xsd:maxLength value="50"/>
</xsd:restriction>
</xsd:simpleType>
</xsd:element>
<xsd:element name="TitreAlbum" minOccurs="0" od:jetType="text" od:sqlSType="nvarchar">
<xsd:simpleType>
<xsd:restriction base="xsd:string">
<xsd:maxLength value="50"/>
</xsd:restriction>
</xsd:simpleType>
</xsd:element>
<xsd:element name="CodeArtiste" minOccurs="0" od:jetType="text" od:sqlSType="nvarchar">
<xsd:simpleType>
<xsd:restriction base="xsd:string">
<xsd:maxLength value="50"/>
</xsd:restriction>
</xsd:simpleType>
</xsd:element>
<xsd:element name="Anneesortie" minOccurs="0" od:jetType="longinteger" od:sqlSType="int">
<xsd:simpleType>
<xsd:restriction base="xsd:integer"/>
</xsd:simpleType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:schema>
```

Figure 30 : Première partie du fichier XML généré : Définition des balises

```
<dataroot xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance">
<Album>
<CodeAlbum>10000</CodeAlbum>
<TitreAlbum>EarthLing</TitreAlbum>
<CodeArtiste>Dabow</CodeArtiste>
<Anneesortie>1996</Anneesortie>
</Album>
<Album>
<CodeAlbum>10001</CodeAlbum>
<TitreAlbum>Ziggy Stardust</TitreAlbum>
<CodeArtiste>DaBow</CodeArtiste>
<Anneesortie>1973</Anneesortie>
</Album>
<Album>
<CodeAlbum>21111</CodeAlbum>
<TitreAlbum>Berlin</TitreAlbum>
<CodeArtiste>Loree</CodeArtiste>
<Anneesortie>1973</Anneesortie>
</Album>
</dataroot>
</root>
```

Figure 31 : Seconde partie du fichier XML généré : Données de la table Album

Chapitre 3 - Le langage de programmation de Access : VBA

Après avoir abordé les différences et nouveautés apportées par un passage à Access XP dans le cadre de l'interface normale de travail d'Access, nous allons maintenant étudier les changements dans l'interface de programmation d'Access : VBA.

I - Introduction

Le VBA (Visual Basic for Applications) est la réponse de Microsoft aux besoins des développeurs de base de données en terme de programmation.

Si les assistants présentés précédemment permettent au plus grand nombre de créer des applications simples de manière intuitive, ils ne permettent pas de réaliser certaines opérations plus complexes. C'est pourquoi les développeurs d'applications de base de données professionnelles ont besoin d'une autre interface, une interface de programmation brute leur permettant d'effectuer n'importe quel traitement : c'est le VBA d'Access.

Grâce au VBA dans Access, on peut totalement contrôler sa base de données et effectuer des opérations complexes, comme par exemple recopier le contenu d'une table dans une autre base afin d'en archiver les données. Mais c'est surtout dans la création de l'interface entre l'utilisateur et la base de données, c'est-à-dire principalement au travers des formulaires, que le VBA est le plus utilisé. Il permet par exemple de créer des formulaires permettant de manipuler les données contenues dans le système d'information.

Afin de pouvoir créer de tels formulaires, le développeur a besoin d'accéder aux enregistrements contenus dans le système au travers du VBA. Pour ce faire le langage s'appuie sur des bibliothèques qui sont un ensemble de méthodes d'accès aux données, de définition de structure et de gestion de sécurité.

Avant Access 2000, Microsoft avait développé un modèle objet d'accès aux données baptisé **DAO** (Data Access Objects). Ce modèle était basé sur les pilotes ODBC[2] afin d'accéder aux sources de données.

Depuis Access 2000, Microsoft développe un nouveau modèle objet standard d'accès aux données : **ADO** (ActiveX Data Objects). Ce modèle est basé sur OLE DB (Object Linking and Embedding DataBase).

Chaque application manipule des documents (au sens large : texte, images, sons, etc.) qui sont des collections d'objets. Une application doit pouvoir charger des objets, les afficher, en permettre la modification... Si l'objet n'a pas une structure que l'application peut traiter en natif, le système d'exploitation charge l'application associée à cet objet, à l'intérieur de la première application. C'est la technologie **OLE** (Object Linking and Embedding).
OLE DB est une extension de cette technologie aux bases de données. Quand Access rencontre une base de données incompatible, il utilise l'application associée à la base de données afin de pouvoir utiliser ce qu'elle contient. Dans un certain sens, un fournisseur OLE DB est similaire à un pilote ODBC qui fournit un mécanisme uniforme d'accès à des données relationnelles. Les fournisseurs OLE DB fournissent non seulement un

[2] **ODBC** signifie Open Database Connectivity. Il s'agit d'un ensemble de fonctions de haut niveau qui permettent de travailler avec une base de données relationnelle au travers d'une interface identique quelque soit le type de cette base.

mécanisme uniforme d'accès aux données relationnelles, mais également aux données non relationnelles.

ADO est une couche supplémentaire au dessus de OLE DB, celui-ci étant jugé encore trop complexe pour être utilisé par les développeurs. ADO est l'interface de programmation d'OLE DB.

Cette migration vers ADO a été décidée par Microsoft car DAO et ODBC ne permettent d'accéder qu'à des données contenues dans des bases de données Microsoft Access et dans certains autres types spécifiques (ISAM par exemple). De plus, le modèle objet ADO est composé de moins d'objets et plus facile à utiliser.

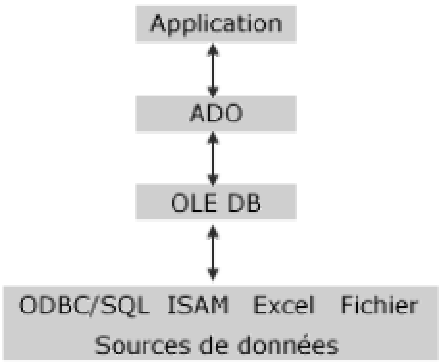


Figure 32 : Mécanisme de fonctionnement d'OLE DB

Le but de ADO (et de OLE DB, sa couche sous-jacente) est de permettre à terme un accès uniforme aux données contenues dans des fichiers créés avec n'importe quel outil de traitement de données.

I.1 - Conséquences dans l'utilisation de la fenêtre VBA d'Access :

Comme nous l'avons vu précédemment, Access utilise des bibliothèques pour interpréter les instructions contenues dans le code VBA. Il est donc nécessaire d'indiquer à Access quelle bibliothèque il doit utiliser.

C'est le but de la fenêtre Références :

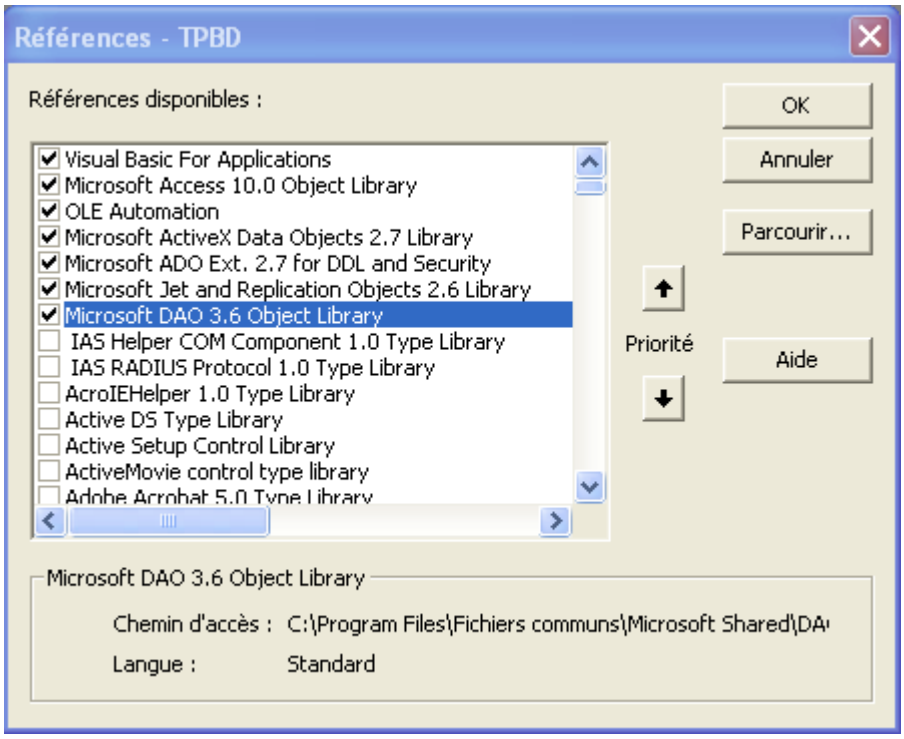


Figure 33 : La fenêtre Références de Visual Basic

Cette fenêtre est située dans le menu Outils→Références de la fenêtre Visual Basic de Access XP. Pour accéder à cette fenêtre Visual Basic, il faut ouvrir depuis la fenêtre principale d'Access un module de code ou la partie de code d'un formulaire. La fenêtre qui s'ouvre alors est la fenêtre Microsoft Visual Basic.

Elle permet d'indiquer à Access XP quelle bibliothèque utiliser lorsqu'il doit interpréter du code VBA. En effet Access, au travers du VBA s'appuie sur ces bibliothèques pour accéder aux données et les traiter.

Afin de permettre aux bases de données contenant des modules écrits en DAO de fonctionner, Microsoft a gardé dans la version XP de Access la bibliothèque DAO. Dans ces circonstances, il est nécessaire de choisir la référence :

- [Microsoft DAO 3.6 Object Library]

Si l'on souhaite développer des modules en ADO, il est nécessaire de choisir les références :

- [Microsoft ActiveX Data Objects 2.7 Library][3]
- [Microsoft ADO Ext. 2.7 for DDL And Security]
- [Microsoft Jet and Replication Objects 2.6 Library] (Accessoire)

Nous verrons bientôt quels sont les rôles de ces différents composants.

Attention à l'ordre des bibliothèques dans cette fenêtre références
En effet, lorsque Access aura besoin d'une bibliothèque pour interpréter une méthode d'un objet par exemple, il utilisera de préférence les bibliothèques situées en haut de la liste.
Par exemple, l'objet Recordset est défini en DAO et aussi en ADO. Ainsi, en fonction de la position dans la bibliothèque correspondante, le recordset créé par le code suivant : `Dim record As Recordset` sera soit un recordset ADO (ce sera le

[3] Access XP contient les références ADO 2.0 à 2.7. Afin de pouvoir utiliser tous les exemples de cet exposé nous vous recommandons de choisir la référence ADO 2.7.

cas avec la liste de références ci-dessus) ou un recordset DAO, deux objets différents n'ayant pas les mêmes méthodes ou fonctions.

Dans un souci de sécurité et de stabilité du code développé, nous recommandons de spécifier le type de modèle utilisé afin d'éviter des anomalies de fonctionnement provenant de l'ordre de priorité des références. Nous écrirons ainsi l'exemple précédent `Dim record As ADODB.Recordset` ou `Dim record As DAO.Recordset` en fonction du cas choisi.

Pour la même raison, il est aussi déconseillé de cocher à la fois des références DAO et ADO dans la fenêtre Références.

I.2 - Utilisation de l'aide de VBA

Afin de nous aider au développement d'applications en VBA, Microsoft met à notre disposition une aide très fournie. Cependant, cette aide est constituée de mises à jour successives depuis Access 97, et on y trouve parfois des redondances d'informations voire des informations obsolètes.

Pour accéder à cette aide, il faut ouvrir depuis la fenêtre principale d'Access un module ou la partie de code d'un formulaire. Une seconde fenêtre s'ouvre alors intitulée "Microsoft Visual Basic" dans lequel se trouve le code de notre application.

Une fois dans cette fenêtre, une pression sur la touche F1 permet d'ouvrir l'aide de Visual Basic.

Afin de permettre une navigation aisée, nous allons donner ci-dessous quelques points d'entrée dans cette aide :

- Pour accéder au modèle objet ADO, qui permet par la suite de retrouver chaque méthode et propriété des objets ADO, on tapera dans la zone "aide intuitive" les mots clés : "modèle objet ADO" (on fera de même pour le modèle objet ADOX).
- D'une manière générale, utiliser le mot-clé Nouveautés suivi du modèle objet souhaité (ADO, ADOX, ADO MD) permet d'accéder à une page sommaire qui contient des liens vers des exemples de code en VBA.

Pour rechercher un objet, une méthode ou une propriété spécifique, on spécifiera tout d'abord ADO puis la méthode ou propriété à rechercher, par exemple "ADO find". Le fait de spécifier tout d'abord ADO permet de n'avoir des résultats ne contenant que les éléments ADO et non un mélange ADO/DAO qui sème vite la confusion. Par exemple si on tape seulement "find", les résultats renvoyés contiendront FindFirst, FindNext… qui sont des méthodes DAO obsolètes pour le développeur ADO.

Cette coexistence des références ADO et DAO est à l'origine de nombreuses confusions. Nous vous conseillons donc de toujours bien vérifier que les résultats rendus correspondent bien à la bibliothèque que vous utilisez pour votre développement.

II - Présentation de ADO

Nous allons détailler dans cette partie le modèle d'accès aux données ADO.

Pour cela, nous ne nous appuierons pas sur une comparaison stricte entre les deux versions comme dans le chapitre II car cela n'apporterait pas d'éléments significatifs.

Nous établirons cependant des correspondances[4] lorsqu'elles seront nécessaires afin que le développeur habitué à l'environnement de programmation d'Access 97 puisse facilement créer ses applications en ADO.

Afin de proposer les mêmes fonctionnalités de manipulation des données que DAO, ADO contient plusieurs sous-modèles objets :

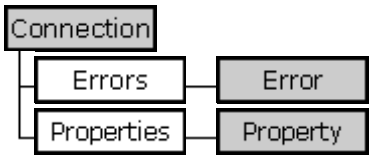
- **ADO DB :** Le modèle ADO DB (ou ADO par abus de langage) permet à l'application client d'accéder et de manipuler les données au travers d'une interface OLE. ADO DB regroupe les objets permettant de se connecter à une source de données, de lire, d'ajouter ou de mettre à jour les données.
- **ADOX (Microsoft ADO Extensions for DDL and Security) :** Le modèle ADOX regroupe des objets permettant la définition des structures de données (tel tables, vues ou index) et l'administration de la sécurité et des permissions d'accès aux différents utilisateurs et groupes d'utilisateurs.
- **ADO MD (MultiDimensionnal) :** Le modèle ADO MD permet la manipulation de source de données multidimensionnelles. Nous n'aborderons pas cette partie très technique dans cet exposé.
- **RDS (Remote Data Service) :** RDS est un dispositif d'ADO qui permet de transférer des données d'un serveur vers une application client, de manipuler les données depuis le client puis de retourner les modifications vers le serveur. Nous ne détaillerons pas ce modèle qui est spécifique à l'environnement client serveur dans cet exposé.
- **JRO (Microsoft Jet and Replication Objects) :** Le modèle JRO regroupe les objets, propriétés et méthodes pour créer, modifier et synchroniser des répliques de base de données. A la différence de ADO et ADOX, le modèle JRO ne peut être utilisé qu'avec des sources de données Microsoft Jet Database. Nous ne nous intéresserons pas à ce modèle dans cet exposé, car il n'a pas d'utilité pour les traitements que nous effectuerons.

Le moteur de base de données Microsoft Jet n'étant plus développé aujourd'hui par Microsoft, il en est de même du modèle JRO.

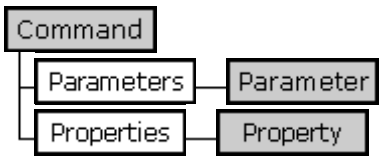
1 - Modèle objet ADO :

Abordons désormais l'étude des objets d'accès aux données composant le modèle ADO[5].

Il existe cinq objets différents dans le modèle objet ADO de la référence ADO 2.7.



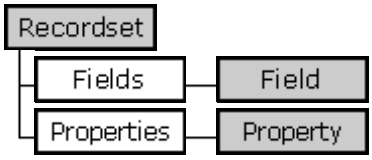
L'objet Connection permet d'établir les connexions entre le client et la source de données.



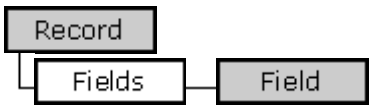
[4] Ces correspondances entre les méthodes DAO et ADO sont détaillées sous forme d'un tableau dans l'annexe A : "DAO vers ADO – Aide mémoire".

[5] Ce modèle objet ADO est détaillé dans l'annexe B :"Modèle objet ADO détaillé" à la fin de ce rapport.

L'objet Command permet de réaliser des commandes, telles que des requêtes SQL ou des mises à jour d'une base.



L'objet Recordset permet de voir et de manipuler les résultats d'une requête



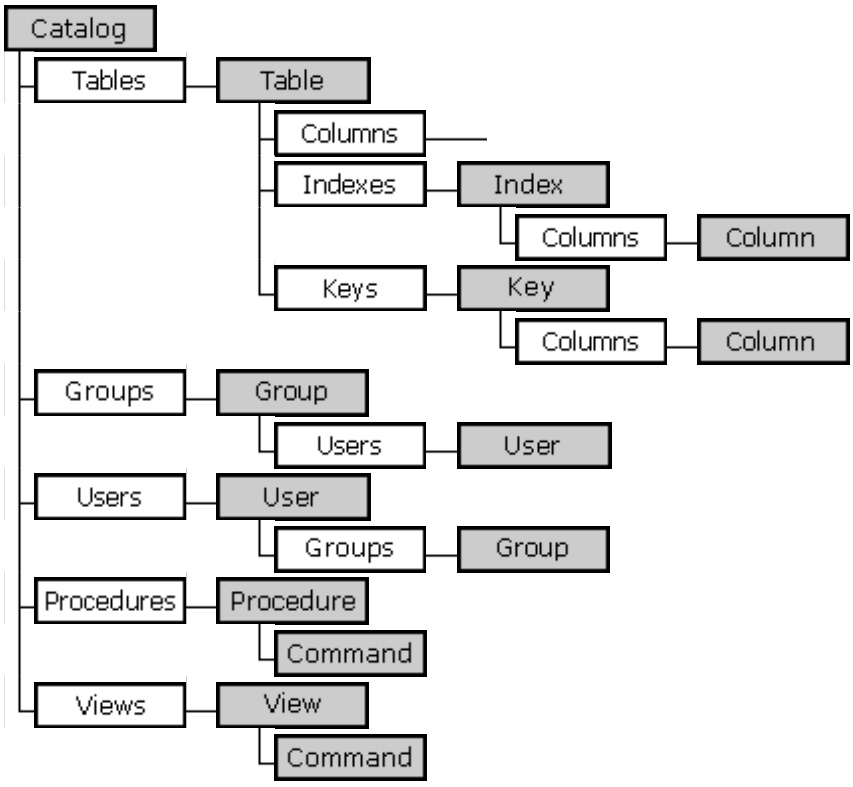
L'objet Record représente une colonne de données. Il est conceptuellement similaire à un recordset à une seule colonne.

L'objet Stream représente un flux de données binaires ou de texte.

2 – Modèle objet ADOX

Ce modèle objet ADOX permet de compléter le précédent (ADO).

Il permet entre autre de définir les schémas de structure de la base de données et de fixer des autorisations d'accès à la base de données en fonction d'un utilisateur particulier ou d'un groupe d'utilisateurs.



L'objet Catalog contient des collections qui décrivent le schéma d'une base de données.

L'objet Table représente une table, incluant les colonnes, les index et les clés qu'elle contient.

L'objet Index représente un index d'une table de la base de données.

L'objet Key représente un champ clé primaire, étrangère ou unique d'une table de la base de données.

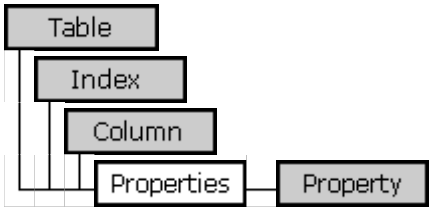
L'objet Column représente une colonne d'une table, d'un index ou d'une clé.

L'objet Group représente un groupe d'utilisateurs avec ses permissions d'accès à l'intérieur d'une base de données sécurisée.

L'objet User représente un compte d'utilisateur avec ses permissions d'accès à l'intérieur d'une base de données sécurisée.

L'objet Procedure représente une procédure stockée.

L'objet View représente un ensemble d'enregistrements filtrés ou une table virtuelle.



Chacun des objets Table, Index et Colonnes a une collection standard de Propriétés.

III - Utiliser ADO avec VBA

Une fois ces précisions sur les modèles d'accès aux données mis en place par Access XP effectuées, nous allons dans cette partie nous attacher à l'utilisation à l'intérieur de nos modules VBA de ces objets.

Afin d'illustrer l'emploi de ces objets, et de leurs méthodes et propriétés, abordés dans cette partie, nous essaierons d'utiliser deux sortes d'exemples :

- un exemple propre à chaque méthode expliquée car si elle apparaît dans cette partie c'est qu'elle nous semblait importante et qu'elle a donc besoin d'être illustrée
- un exemple tiré de la création de la base de données Discothèque, support des travaux pratiques, lorsque l'utilisation de cette méthode est requise.

III.1 – Connection (ADO) et Catalog (ADOX)

L'objet connection est le premier objet manipulé par le développeur puisqu'il permet de se connecter à une source de données.

III.1.1 - Ouvrir une connexion à une base de données existante

En DAO pour ouvrir une connexion à une base de données sous Access 97 on utilisait `DBEngine.OpenDatabase ()`.

En ADO on doit créer une connexion à cette base de données ou à toute autre source de données. C'est pourquoi il existe un objet **connection**, qui permet de définir une session pour un utilisateur sur une source de données spécifique.
Cet objet connection doit être défini puis créé :

```
Dim cnn As ADODB.Connection
Set cnn = New ADODB.Connection
```

Il doit ensuite être affecté. On utilise la méthode **Open** de **Connection** :

```
cnn.Open "Provider=Microsoft.Jet.OLEDB.4.0;" & _
      "Data Source=.\NorthWind.mdb;"
```

Cet exemple passe deux paramètres à la méthode **Open** :

- Le premier est une chaîne contenant le driver OLE DB à utiliser pour l'ouverture, soit : "Provider=Microsoft.Jet.OLEDB.4.0;" avec des bases de données Access. La configuration de cette chaîne peut aussi être effectuée par la propriété `Connection String`.
- Le second est aussi une chaîne contenant le chemin vers la base à ouvrir, dans cet exemple Microsoft : "Data Source=.\NorthWind.mdb;"

III.1.2 - Accéder à la base de données courante

Lorsque du développement de modules en VB, on a souvent besoin d'accéder à la base de données ouverte sous Access, support de notre code.
Afin de simplifier cette ouverture fréquente, on utilise la propriété **AccessConnection** de l'objet **CurrentProject** pour obtenir une connexion à la base de données courante.

Exemple : *Création d'un accès à la base de données courante :*

```
Sub ADOAcceder_a_la_BD_Courante ()
   Dim cnn As New ADODB.Connection
   Set cnn = CurrentProject.AccessConnection
End Sub
```

Au sein de n'importe quelle fonction dans Discothèque lorsque nous voudrons créer une connexion à Discothèque, nous emploierons :

```
Dim connexion As New ADODB.Connection
Set connexion = CurrentProject.AccessConnection
```

III.1.3 - Définition du schéma de la base de données

Pour créer à proprement dit une nouvelle base de données on utilise la méthode **Create** de l'objet **Catalog** d'ADOX.

```
Sub ADOCreationDeBD()
   Dim cat As New ADOX.Catalog
   cat.Create "Provider=Microsoft.Jet.OLEDB.4.0;" "Data Source=.\New.mdb;"
End Sub
```

Pour créer une nouvelle table dans la base de données couramment ouverte, on utilise l'objet **Catalog** de ADOX. L'objet **Catalog** correspond à l'objet Database en DAO.

```
Sub ADOCreationDeTable()
   Dim cat As New ADOX.Catalog
   Dim table As New ADOX.Table

   ' Ouvrir le catalogue
   cat.ActiveConnection = "Provider=Microsoft.Jet.OLEDB.4.0;" & _
      "Data Source=.\New.mdb;"
```

```
    ' Créer un nouvel objet Table
    With table
        .Name = "Contacts"
        .Columns.Append "ContactName", adVarWChar
        .Columns.Append "Phone", adVarWChar
        .Columns("Notes").Attributes = adColNullable
    End With

    ' Ajouter la nouvelle table à la base de données
    cat.Tables.Append table

    ' Fermeture des connexions
    Set cat = Nothing
End Sub
```

Il est nécessaire de créer les champs et de les ajouter à la nouvelle table avant d'ajouter la nouvelle table à la collection de tables de l'objet Catalog.

Les noms des types de données sont différents en ADO. Voici les correspondances avec les types de données DAO:

Types de données DAO	Types de données ADO correspondant
DbBinary	adBinary
DbBoolean	adBoolean
DbByte	adUnsignedTinyInt
DbCurrency	adCurrency
DbDate	adDate
DbDecimal	adNumeric
DbDouble	adDouble
dbGUID	adGUID
dbInteger	adSmallInt
dbLong	adInteger
dbLongBinary	adLongVarBinary
dbMemo	adLongVarWChar
dbSingle	adSingle
dbText	adVarWChar

III.2 – Recordset

Les recordsets permettent d'accéder et de manipuler les enregistrements.

III.2.1 - Ouvrir un recordset avec ADO

Comme en DAO, il existe de nombreux moyens d'ouvrir un recordset :

- Avec la méthode **Execute** de l'objet **Connection**
- Avec la méthode **Execute** de l'objet **Command**
- Avec la méthode **Open** de l'objet **Recordset**

Il existe de nombreux paramètres permettant de spécifier le type de recordset que l'on souhaite ainsi créer.

Les deux tableaux suivants font la correspondance entre les anciennes méthodes de spécification en DAO et celles utilisées en ADO, le premier s'attachant aux types de recordset à créer et le second aux propriétés de verrouillage du recordset :

Types de recordset DAO	**Propriété ou paramètres du recordset ADO correspondant**
DbOpenDynaset	CursorType=adOpenDynamic
DbOpenSnapshot	CursorType=adOpenStatic
DbOpenForwardOnly	CursorType=adOpenForwardOnly
DbOpenTable	CursorType=adOpenDynamic, Options=adCmdTableDirect

Valeurs LockType du recordset DAO	**Valeur LockType du Recordset ADO correspondant**
DbReadOnly	adLockReadOnly
DbPessimistic	adLockPessimistic
DbOptimistic	adLockOptimistic

Exemple : *Ouverture d'un recordset seulement vers l'avant, en lecture seule:*

```
Sub ADOOuvrirRecordset()

   Dim cnn As New ADODB.Connection
   Dim rst As New ADODB.Recordset
   Dim fld As ADODB.Field

   ' Ouvrir la connexion
   cnn.Open "Provider=Microsoft.Jet.OLEDB.4.0;" & _
      "Data Source=.\NorthWind.mdb;"

   ' Ouverture du recordset avec défilement en avant et en lecture seule
   rst.Open _
      "SELECT * FROM Customers WHERE Region = 'WA'", _
      cnn, adOpenForwardOnly, adLockReadOnly

   ' Fermer le recordset
   rst.Close
End Sub
```

Dans Discothèque :

```
record.Open "SELECT * FROM Code", connexion, adOpenKeyset, adLockOptimistic
```

III.2.2 - Déterminer la position courante d'un recordset

Tout comme en DAO, il existe en ADO la propriété **AbsolutePosition** qui permet de connaître le numéro du record sur lequel pointe le curseur en cours. Cependant, il y a deux points importants à retenir :

- Tout d'abord, le comptage grâce à la propriété **`AbsolutePosition`** en ADO commence à un ; c'est à dire que le premier enregistrement porte le numéro un, tandis qu'en DAO le premier enregistrement portait le numéro zéro.
- Ensuite, il est nécessaire de configurer la propriété **`CursorLocation`** à **`adUseClient`**. En effet, si la propriété n'est pas spécifiée ou est fixée à **`adUseServer`**, la propriété **`AbsolutePosition`** renverra **`adUnknown (-1)`**, car Jet ne peut retrouver l'information avec cette propriété.

ADO propose aussi une propriété **PercentPosition** qui retourne un pourcentage représentant la position approximative du record à l'intérieur du Recordset.

III.2.3 - Trouver des enregistrements dans un recordset

ADO possède deux mécanismes comme DAO pour rechercher un Recordset : **Find** et **Seek**.

Si **Seek** est en général plus performant, il ne peut être utilisé qu'avec des objets Recordset qui ont des **Index** associés. De plus, pour les bases de données Microsoft Jet, seuls les objets Recordsets basés sur une table avec un index supportent **Seek**.

III.2.3.1 - La méthode Find

DAO inclut quatre méthodes : **FindFirst, FindLast, FindNext, FindPrevious**. En ADO il existe une seule méhode : **Find**. La recherche débute toujours à partir de la position courante dans le recordset. La méthode Find accepte des paramètres qui permettent de spécifier le sens de la recherche (en avant, en arrière) ainsi qu'une adresse depuis le record courant pour spécifier où commencer la recherche.

Méthode DAO	**Find ADO avec SkipRows**	**Direction de la recherche ADO**
FindFirst	0	adSearchForward (Si la position courante est différente du premier record, utiliser **MoveFirst** avant **Find**)
FindLast	0	adSearchBackward (Si la position courante est différente du premier record, utiliser **MoveLast** avant **Find**)
FindNext	1	adSearchForward
FindPrevious	1	adSearchBackward

***Exemple** : Trouver un record en ADO en spécifiant un critère de recherche:*

```
Sub ADOFindRecord()


   Dim cnn As New ADODB.Connection
   Dim rst As New ADODB.Recordset
```

```
   ' Ouvrir la connexion
   cnn.Open "Provider=Microsoft.Jet.OLEDB.4.0;" & _
      "Data Source=.\NorthWind.mdb;"

   ' Ouvrir le recordset
   rst.Open "Clients", cnn, adOpenKeyset, adLockOptimistic

   ' Trouver le premier client dont le pays est USA
   rst.Find "Pays='USA'"

      ' Fermer le recordset
   rst.Close
End Sub
```

Remarque : En ADO, le critère de recherche spécifié dans la commande **Find** ne peut **pas porter sur plus d'une colonne.**

Tandis qu'en DAO on pouvait effectuer des recherches sur de multiples colonnes en les spécifiant dans le critère (le paramètre critère était considéré come une clause WHERE en SQL), il faut en ADO utiliser la propriété **Filter** afin de créer une vue sur le Recordset ne contenant que les records correspondants au critère.

L'exemple de Discothèque ou l'on utilise la propriété Filter (détaillée plus loin) pour créer un recordset utilisable avec Find :

```
record.Filter = "[Code] = '" & UserType & "'"and "[codeAdherent] = '" & passwd & "'
record.Find "CodeAdherent= '" & passwd & "'"
```

Par ailleurs, DAO et ADO ont un comportement différent dans le cas où **Find** ne trouve pas de correspondance.
En DAO, la propriété **NoMatch** est mise à **True** et le record courant est non-défini.
En ADO, si aucune correspondance n'est trouvée, le record courant est positionné soit :

- **Avant le début du Recordset** si on cherche en avant (**adSearchForward**)
- **Après la fin du Recordset** si on cherche en arrière (**adSearchBackward**).

Il conviendra donc d'utiliser les propriétés

- **EOF** avec **adSearchForward**
- **BOF** avec **adSearchBackward**

afin de déterminer si un record correspondant a été trouvé.

Remarque : ADO ne reconnaît pas l'opérateur Is, ni Is Not.
ADO utilise les opérateurs = et <> pour remplacer les opérateurs précédents.

Exemple :

DAO:

```
"ColumnName Is Null"
"ColumnName Is Not Null"
```

ADO:

```
"ColumnName = Null"
```

```
"ColumnName <> Null"
```

III.2.3.2 - La méthode Seek

Exemple : *Le code suivant présente une recheche avec* ***Seek*** *en ADO*

```
Sub ADOSeekRecord()
   Dim cnn As New ADODB.Connection
   Dim rst As New ADODB.Recordset

   ' Ouvrir la connexion
   cnn.Open "Provider=Microsoft.Jet.OLEDB.4.0;" & _
      "Data Source=.\NorthWind.mdb;"

   ' Ouvrir le recordset
   rst.Open "Order Details", cnn, adOpenKeyset, adLockReadOnly, _
      adCmdTableDirect

   ' Selectionner l'index utilisé pour trier les données dans le recordset
   rst.Index = "PrimaryKey"

   ' Recherche de la commande correspondante aux critères
   rst.Seek Array(10255, 16), adSeekFirstEQ

   ' Fermer le recordset
   rst.Close
End Sub
```

La méthode **Seek** base sa recherche sur un **index**.
Il est donc important de spécifier quel index le moteur de base de données doit utiliser pour la recherche. Par défaut, Microsoft Jet utilise la clé primaire de la relation considérée.

Dans l'exemple, la fonction **Array** (appartenant à la bibliothèque Visual Basic For Applications) est utilisée pour spécifier un critère de recherche sur plus d'une colonne (recherche multicritères). Si la recherche ne porte que sur une seule colonne, il n'est pas nécessaire d'utiliser **Array**.

Pour déterminer si un record correspondant au critère a été trouvé, il faut utiliser les propriétés de position du record courant **EOF** et **BOF** comme avec la méthode **Find**.

Remarque : La méthode **Seek** ne fonctionne correctement qu'avec des bases de données Microsoft Jet 4.0. Tous les autres formats entraîneront une erreur : « run-time error ».
Il est possible de tester le recordset ouvert grâce à la méthode **Supports** de l'objet **Recordset**, afin de déterminer si la méthode **Seek** est disponible ou non pour le recordset courant.

III.2.4 - Filtrer et Trier les données dans un recordset

ADO aborde ces fonctions sur les Recordset d'une manière différente que DAO. Alors qu'en DAO ces propriétés **Filter** et **Sort** s'appliquaient aux **recordsets** ouverts après leur définition, ADO applique ces propriétés au **recordset** duquel on appelle ces propriétés.

III.2.4.1 - Utiliser la propriété Filter (Filtre):

Exemple : *Filtrer un recordset en fonction de critères définis dans une clause SQL*

```
Sub ADOFilterRecordset()

   Dim cnn As New ADODB.Connection
   Dim rst As New ADODB.Recordset

   ' Ouvrir la connexion
   cnn.Open "Provider=Microsoft.Jet.OLEDB.4.0;" & _
      "Data Source=.\NorthWind.mdb;"

   ' Ouvrir le recordset
   rst.Open "Customers", cnn, adOpenKeyset, adLockOptimistic

   ' Filtrer le recordset pour avoir uniquement les clients américains ayant un fax
   rst.Filter = "Country='USA' And Fax <> Null"
   Debug.Print rst.Fields("CustomerId").Value

   ' Fermer le recordset
   rst.Close
End Sub
```

La propriété **Filter** d'ADO permet de créer une vue temporaire qui peut être utilisée pour localiser un record, ou un ensemble de records, correspondants au(x) critère(s) de recherche au sein du Recordset.

Quand un filtre est appliqué à un Recordset grâce à **Filter**, la propriété **RecordCount** renvoie le nombre de records à l'intérieur de cette vue filtrée du Recordset.

Le filtre peut être enlevé en configurant la propriété **Filter** sur **adFilterNone**.

III.2.4.2 - Utiliser la proriété Sort (Tri)

Exemple : *Utiliser **Sort** pour ordonner les records dans un Recordset*

```
Sub ADOSortRecordset()
   Dim cnn As New ADODB.Connection
   Dim rst As New ADODB.Recordset

   ' Ouvrir la connexion
   cnn.Open "Provider=Microsoft.Jet.OLEDB.4.0;" & _
      "Data Source=.\NorthWind.mdb;"
```

```
    ' Ouvrir le recordset
    rst.CursorLocation = adUseClient
    rst.Open "Customers", cnn, adOpenKeyset, adLockOptimistic

    ' Tri du recordset selon le pays et la région, tous deux en ordre croissant
    rst.Sort = "Country, Region"
    Debug.Print rst.Fields("CustomerId").Value

    ' Fermer le recordset
    rst.Close
End Sub
```

Remarque : Pour utiliser la propriété **Sort**, il faut spécifier la propriété **CursorLocation** à **adUseClient** avant d'ouvrir le Recordset.

III.2.5 - Ajouter ou modifier des enregistrements

Après l'ouverture d'un recordset modifiable (Options ***adOpenKeyset, adLockOptimistic*** de **Open**), on utilisera la méthode **AddNew** afin d'insérer un nouvel enregistrement.

Exemple :

```
Sub ADOAjouterRecord()
  Dim cnn As New ADODB.Connection
  Dim rst As New ADODB.Recordset

  cnn.Open "Provider=Microsoft.Jet.OLEDB.4.0;" & "Data Source=.\NorthWind.mdb;"
  rst.Open "SELECT * FROM Customers", cnn, adOpenKeyset, adLockOptimistic

  ' Ajouter un nouveau record
  rst.AddNew

  ' Spécifier les valeurs des champs
  rst!CustomerId = "HENRY"
  rst!PostalCode = "98107"

  ' Enregistrer les changements faits dans l'enregistrement en cours du recordset
  rst.Update

  ' Fermer le recordset
  rst.Close
End Sub
```

Le curseur du recordset est automatiquement positionné sur l'enregistrement nouvellement inséré avec **AddNew**.

Il est possible de passer les valeurs de l'enregistrement en paramètres de **AddNew** ; il faudra alors décrire un tableau (Array) de champ, puis un tableau de valeurs :

Dans l'exemple précédent :

```
   rst.AddNew Array("CustomerId", "PostalCode"), Array("HENRY", "98107")
   rst.Update
```

Pour modifier les données d'un recordset ouvert, on accède au champ à changer par la propriété **Value** de **Fields** ou directement par la syntaxe à ! :

Dans l'exemple précédent, à la place de AddNew :

```
' Update the CustomerId of the first record
  rst.Fields("CustomerId").Value = "New Name"
  rst!PostalCode = "New PostalCode"
```

Il n'est pas nécessaire en ADO d'appeler la méthode Update pour confirmer les changements. Ils sont automatiquement pris en compte lorsqu'on se déplace dans le recordset vers un autre enregistrement.

Grâce à ces méthodes détaillées, il est possible d'effectuer la plupart des traitements courants pour lesquels l'utilisation de VBA est nécessaire.

Cependant, des annexes sont disponibles à la fin de ce rapport pour celui qui souhaite de plus amples précisions. A ce titre, la version informatique de ce rapport comporte de nombreux liens hypertextes vers le site Internet de référence qui nous a permis d'écrire ce chapitre : Microsoft developper network, où il est possible de trouver de nombreuses explications et exemples de code.

Conclusion

Nous avons vu lors de l'adaptation des travaux pratiques que les différences entres Access XP et Access 97 sont peu significatives. Peu de nouveautés ont été apportés à la création des tables, requêtes, formulaires, états...

Toutefois Microsoft Access a su suivre l'évolution concernant le Web et mettre à disposition un nouvel objet dans la création d'une base de données en l'occurrence les pages d'accès aux données. Ils ont également instauré la prise en charge du XML, standard universel de présentation des données. Et nous avons vu que les pages d'accès aux données sont non seulement une source pour Internet mais elles peuvent aussi être utilisées au sain même d'une base en remplacement d'un formulaire ou d'un état.

Microsoft Access a également révolutionné le mode d'accès aux données en changeant radicalement de technologie : passant de DAO (Data Access Objects) à ADO (ActiveX Data Objects). Comme nous l'avons vu la technologie ADO facilite l'accès aux données en créant une méthode unique contrairement à la technologie DAO qui nécessitait un accès particulier pour chaque type d'objet.

D'autre part, il nous paraît important de signaler que l'accès à des sources d'informations concernant cette étude est très difficile. En effet peu voire pas d'ouvrages traitent des changements et évolutions d'Access. Ceci fut très contraignant pour la réalisation de ce projet. Bien que le thème de ce projet est intéressant et instructif (car la technologie ADO est également utilisée par la technologie ASP (Active Server Pages) développée par Microsoft pour la création de site Internet en lien avec des bases de données) la difficultés d'accès aux sources et l'aide obsolète d'Access ont largement contribué à la difficulté de réalisation de ce projet.

De plus la plupart de nos sources d'informations sont en anglais car elles proviennent essentiellement du site Microsoft Developper Network entièrement en anglais.

En résumé l'utilisation d'Access XP pour l'utilisateur non initié qui connaît le mode de fonctionnement d'Access 97 ne sera pas trop bouleversé. Alors que l'utilisateur confirmé qui désire réaliser de la programmation en VBA peut se retrouver confronté à une incapacité de création tant les technologies sont différentes.

Access XP est loin d'être un logiciel parfait mais ses évolutions sont remarquables. Il est tout de même regrettable que Microsoft Access ne fasse évoluer son aide que par l'ajout répété d'information sans réelles mises à jour.

Bibliographie

L'aide de Access XP nous a aussi été très utile, même si la navigation à l'intérieur de cette aide n'est pas facile car elle a été complétée sans cesse depuis le début d'Access ce qui conduit parfois à des redondances d'informations voire à des informations obsolètes.

Afin d'écrire la partie programmation VBA de ce rapport nous avons essentiellement utilisé le centre de documentation en ligne MSDN (Microsoft Developper Network) à l'adresse www.msdn.com. Ce site est très fourni en informations sur la technologie ADO ainsi qu'en exemples de codes facilitant la mise en pratique des informations recueillies. Cependant, il est entièrement en anglais, et si une version française existe (www.microsoft.com/france/msdn/), elle est beaucoup moins développée et mise à jour. Nous recommandons donc de s'appuyer sur le site américain beaucoup plus fourni.

Nous avons par ailleurs utilisé aussi les forums de discussions regroupant des développeurs.

Enfin nous nous sommes également appuyé sur deux livres :

- HASENFRATZ JM

ACCESS 2002 Gérer ses bases de données

Grenoble, PUG, 2002

Cet ouvrage n'aborde peu (voire pas) les nouveautés d'Access 2002 et encore moins la programmation VBA.

- JONES E

ACCESS 2002 et VBA Le guide du développeur

Paris, OEM, 2002

Cet ouvrage, bien que très fourni, survole complètement la réelle nouveauté de programmation VBA en l'occurrence le modèle ADO.

Il est à préciser qu'aucun ouvrage distribué n'aborde la grande nouveauté apportée par ADO ce qui est très regrettable. Mais ce phénomène n'est pas exceptionnel car les ouvrages édités par Microsoft ou autres sont rarement mis à jour.

Remerciements

Nous tenons à remercier Mme Sylvie Damy, tutrice de ce projet pour ses conseils, ses orientations et ses relectures tout au long de l'élaboration de ce projet.

Nous tenons aussi à remercier John John, ingénieur-support chez Microsoft qui nous a ouvert les portes de la documentation ADO sur le site MSDN.

Annexes

Annexe A : DAO vers ADO - Aide-mémoire

Le tableau ci-dessous est un aide-mémoire destiné à établir des correspondances entre les méthodes et propriétés de DAO et celles de ADO, ADOX et JRO. Toutefois, les correspondances entre les méthodes et propriétés énumérées ci-dessous ne sont pas forcément directes ni symétriques. Il peut exister des différences, légères ou même nettes, entre les méthodes et propriétés mises en parallèle. Pour plus d'informations sur les propriétés et méthodes de ADO, ADOX et JRO, veuillez consulter la documentation relative au modèle objet

Objet DAO	Propriété/Méthode	Modèle : ADO/ADOX/ JRO	Objet	Propriété/Méthode
DBEngine	DefaultType1	N/A	N/A	N/A
DBEngine	DefaultPassword1	N/A	N/A	N/A
DBEngine	DefaultUser1	N/A	N/A	N/A
DBEngine	IniPath	ADO	Connection	Jet OLEDB:Registry Path2
DBEngine	LoginTimeout	ADO	Connection	ConnectionTimeout
DBEngine	SystemDB	ADO	Connection	Jet OLEDB:System Database2
DBEngine	Version	ADO	Connection	Version
DBEngine	BeginTrans	ADO	Connection	BeginTrans
DBEngine	CommitTrans	ADO	Connection	CommitTrans
DBEngine	Rollback	ADO	Connection	RollbackTrans
DBEngine	CompactDatabase	JRO	JetEngine	CompactDatabase
DBEngine	CreateDatabase	ADOX	Catalog	Create
DBEngine	CreateWorkspace	ADO	Connection	Open
DBEngine	Idle	JRO	JetEngine	RefreshCache
DBEngine	OpenDatabase	ADO	Connection	Open
DBEngine	RegisterDatabase1	N/A	N/A	N/A
DBEngine	RepairDatabase1	N/A	N/A	N/A
DBEngine	SetOption	ADO	Connection	Properties3
Workspace	IsolateODBCTrans	ADO	Connection	Isolation Levels2
Workspace	LoginTimeout	ADO	Connection	ConnectionTimeout
Workspace	Name1	N/A	N/A	N/A
Workspace	Type1	N/A	N/A	N/A
Workspace	UserName	ADO	Connection	User Id2
Workspace	BeginTrans	ADO	Connection	BeginTrans
Workspace	CommitTrans	ADO	Connection	CommitTrans
Workspace	Rollback	ADO	Connection	RollbackTrans
Workspace	Close	ADO	Connection	Close
Workspace	CreateDatabase	ADOX	Catalog	Create
Workspace	CreateGroup	ADOX	Groups	Append
Workspace	CreateUser	ADOX	Users	Append
Workspace	OpenDatabase	ADO	Connection	Open
Database	CollatingOrder	ADO	Connection	Locale Identifier2
Database	Connect	ADO	Connection	ConnectionString

Database	Name	ADO	Connection	Data Source2
Database	QueryTimeout	ADO	Connection	CommandTimeout
Database	Replicable	JRO	Replica	MakeReplicable
Database	ReplicaId	JRO	Replica	ReplicaId
Database	ReplicationConflictFunction	JRO	Replica	ConflictFunction
Database	RecordsAffected	ADO	Connection	Execute(RecordsAffected)
Database	Transactions	ADO	Connection	Transaction DDL2
Database	Updatable	ADO	Connection	Mode
Database	V1xNullBehavior	N/A	N/A	N/A
Database	Version	ADO	Connection	DBMS Version2
Database	Close	ADO	Connection	Close
Database	CreateProperty	N/A	N/A	Not supported in this release
Database	CreateQueryDef	ADOX	Command	Dim New4
Database	CreateRelation	ADOX	Key	Dim New4
Database	CreateTableDef	ADOX	Table	Dim New4
Database	Execute	ADO	Connection	Execute
Database	MakeReplica	JRO	Replica	CreateReplica
Database	NewPassword	ADOX	Catalog	Modify
Database	OpenRecordset	ADO	Recordset	Open
Database	PopulatePartial	JRO	Replica	PopulatePartial
Database	Synchronize	JRO	Replica	Synchronize
Recordset	AbsolutePosition	ADO	Recordset	AbsolutePosition
Recordset	BOF	ADO	Recordset	BOF
Recordset	EOF	ADO	Recordset	EOF
Recordset	Bookmark	ADO	Recordset	Bookmark
Recordset	Bookmarkable	ADO	Recordset	Supports
Recordset	CacheSize	ADO	Recordset	Jet OLEDB:Fat Cursor Cache Size2
Recordset	CacheStart1	N/A	N/A	N/A
Recordset	DateCreated	ADOX	Table	DateCreated
Recordset	LastUpdated	ADOX	Table	DateModified
Recordset	EditMode	ADO	Recordset	EditMode
Recordset	Filter	ADO	Recordset	Filter
Recordset	Index	ADO	Recordset	Index
Recordset	LastModified1	N/A	N/A	N/A
Recordset	LockEdits	ADO	Recordset	LockType
Recordset	Name1	N/A	N/A	N/A
Recordset	NoMatch	ADO	Recordset	Find
Recordset	PercentPosition	N/A	N/A	Not supported in this release.
Recordset	RecordCount	ADO	Recordset	RecordCount
Recordset	RecordStatus	ADO	Recordset	EditMode
Recordset	Restartable1	N/A	N/A	N/A
Recordset	Sort	ADO	Recordset	Sort
Recordset	Transactions1	N/A	N/A	N/A
Recordset	Type	ADO	Recordset	CursorType
Recordset	Updatable	ADO	Recordset	Recordset.Supports(adUpdate)
Recordset	ValidationRule	ADOX	Table	ValidationRule
Recordset	ValidationText	ADOX	Table	ValidationText

Recordset	AddNew	ADO	Recordset	AddNew
Recordset	CancelUpdate	ADO	Recordset	CancelUpdate
Recordset	Clone	ADO	Recordset	Clone
Recordset	Close	ADO	Recordset	Close
Recordset	CopyQueryDef	ADO	Recordset	Source
Recordset	Delete	ADO	Recordset	Delete
Recordset	Edit1	N/A	N/A	N/A
Recordset	FillCache1	N/A	N/A	N/A
Recordset	FindFirst	ADO	Recordset	Find
Recordset	FindLast	ADO	Recordset	Find
Recordset	FindNext	ADO	Recordset	Find
Recordset	FindPrevious	ADO	Recordset	Find
Recordset	GetRows	ADO	Recordset	GetRows
Recordset	Move	ADO	Recordset	Move
Recordset	MoveFirst	ADO	Recordset	MoveFirst
Recordset	MoveLast	ADO	Recordset	MoveLast
Recordset	MoveNext	ADO	Recordset	MoveNext
Recordset	MovePrevious	ADO	Recordset	MovePrevious
Recordset	OpenRecordset	ADO	Recordset	Open
Recordset	Requery	ADO	Recordset	Requery
Recordset	Seek	ADO	Recordset	Seek
Recordset	Update	ADO	Recordset	Update
QueryDef	CacheSize	ADO	Command	Jet OLEDB:Fat Cursor Cache Size2
QueryDef	Connect	ADO	Command	Jet OLEDB:Link datasource2
QueryDef	DateCreated	ADOX	Procedure	DateCreated
QueryDef	LastUpdated	ADOX	Procedure	DateModified
QueryDef	KeepLocal	JRO	Replica	Get/SetObjectReplicability
QueryDef	LogMessages	N/A	N/A	Not supported in this release.
QueryDef	MaxRecords	ADO	Command	MaxRecords
QueryDef	Name	ADOX	Procedure	Name
QueryDef	ODBCTimeout	ADO	Command	Jet OLEDB:ODBC Command Timeout2
QueryDef	RecordsAffected	ADO	Command	Execute(RecordsAffected)
QueryDef	Replicable	JRO	Replica	Get/SetObjectReplicability
QueryDef	ReturnsRecords1	N/A	N/A	N/A
QueryDef	SQL	ADO	Command	CommandText
QueryDef	Type	N/A	N/A	Not supported in this release.
QueryDef	Updatable	N/A	N/A	N/A
QueryDef	Close	ADO/X	Command / Procedure	Set to Nothing
QueryDef	CreateProperty	N/A	N/A	Not supported in this release
QueryDef	Execute	ADO	Command	Command.Execute
QueryDef	OpenRecordset	ADO	Recordset	Open
TableDef	Attributes	ADOX	Table	Properties5

TableDef	ConflictTable	JRO	Replica	ConflictTables
TableDef	Connect	ADOX	Table	Jet OLEDB:Link Datasource2
TableDef	DateCreated	ADOX	Table	DateCreated
TableDef	LastUpdated	ADOX	Table	DateModified
TableDef	KeepLocal	JRO	Replica	Get/SetObjectReplicability
TableDef	Name	ADOX	Table	Name
TableDef	RecordCount	ADO	Connection	OpenSchema6
TableDef	Replicable	JRO	Replica	Get/SetObjectReplicability
TableDef	ReplicaFilter	JRO	Filter	FilterCriteria
TableDef	SourceTableName	ADOX	Table	Jet OLEDB:Remote Table Name2
TableDef	Updatable	N/A	N/A	N/A
TableDef	ValidationRule	ADOX	Table	Jet OLEDB:Table Validation Rule2
TableDef	ValidationText	ADOX	Table	Jet OLEDB:Table Validation Text2
TableDef	CreateField	ADOX	Columns	Append
TableDef	CreateIndex	ADOX	Indexes	Append
TableDef	CreateProperty	N/A	N/A	Not supported in this release.
TableDef	OpenRecordset	ADO	Recordset	Open
TableDef	RefreshLink	ADOX	Table	Jet OLEDB:Create Link2
Field	AllowZeroLength	ADOX	Column	Jet OLEDB:Allow Zero Length2
Field	Attributes	ADOX	Column	Properties5
Field	CollatingOrder	ADO/X	Field/Column	Collation Name2
Field	DataUpdatable	ADO	Field	Attributes
Field	DefaultValue	ADOX	Column	DefaultValue
Field	FieldSize	ADO	Field	ActualSize
Field	ForeignName	ADO	Column	RelatedColumn
Field	Name	ADO/X	Field/Column	Name
Field	OrdinalPosition	N/A	N/A	Not supported in this release.
Field	Required	ADO/X	Field/Column	Attributes
Field	Size	ADO/X	Field/Column	DefinedSize
Field	SourceField	N/A	N/A	Not supported in this release.
Field	SourceTable	N/A	N/A	Not supported in this release.
Field	Type	ADO/X	Field/Column	Type
Field	ValidateOnSet	ADOX	Column	Jet OLEDB:Validate On Set2
Field	ValidationRule	ADOX	Column	Jet OLEDB:Column Validation Rule2
Field	ValidationText	ADOX	Column	Jet OLEDB:Column Validation Text2

Field	Value	ADO	Field	Value
Index	Clustered	ADOX	Index	Clustered
Index	DistinctCount	ADO	Connection	OpenSchema6
Index	Foreign	ADOX	Key	Type
Index	IgnoreNulls	ADOX	Index	IndexNulls
Index	Name	ADOX	Index	Name
Index	Primary	ADOX	Index	PrimaryKey
Index	Required	ADOX	Index	Index.IndexNulls
Index	Unique	ADOX	Index	Unique
Index	CreateField	ADOX	Column	Dim New3
Index	CreateProperty	N/A	N/A	Not supported in this release
Relation	Attributes	ADOX	Key	Properties5
Relation	ForeignTable	ADOX	Key	RelatedTable
Relation	Name	ADOX	Key	Name
Relation	PartialReplica	JRO	Filter	FilterCriteria
Relation	Table	ADOX	Key	Parent Table Object7
Relation	CreateField	ADOX	Column	Dim New3
User	Name	ADOX	User	Name
User	Password	N/A	N/A	N/A
User	PID	N/A	N/A	N/A
User	CreateGroup	ADOX	Groups	Append
User	NewPassword	ADOX	User	ChangePassword
Group	Name	ADOX	Group	Name
Group	PID	N/A	N/A	N/A
Group	CreateUser	ADOX	Users	Append
Container	AllPermissions	ADOX	User/Group	GetPermissions8
Container	Inherit	ADOX	User/Group	Get/SetPermissions
Container	Name1	N/A	N/A	N/A
Container	Owner	ADOX	Catalog	Get/SetObjectOwner
Container	Permissions	ADOX	User/Group	Get/SetPermissions
Container	UserName	ADOX	User/Group	Get/SetPermissions
Document	AllPermissions	ADOX	User	GetPermissions8
Document	Container1	N/A	N/A	N/A
Document	DateCreated	ADOX	Applicable Object	DateCreated
Document	LastUpdated	ADOX	Applicable Object	DateModified
Document	KeepLocal	JRO	Replica	Get/SetObjectReplicability
Document	Name1	N/A	N/A	N/A
Document	Owner	ADOX	Catalog	Get/SetObjectOwner
Document	Permissions	ADOX	User/Group	Get/SetPermissions
Document	Replicable	JRO	Replica	Get/SetObjectReplicability
Document	UserName	ADOX	User/Group	Get/SetPermissions

1 Cette propriété ou méthode n'a aucun équivalent dans ADO, ADOX ou JRO.

2 Cette propriété fait partie de la collection Properties de l'objet.

3 Pour plus d'informations sur les équivalences entre la méthode SetOption et les propriétés connection, veuillez vous reporter à la section "Paramétrage des options de Microsoft Jet".

4 L'objet peut être créé. Utilisez la syntaxe Dim New de Visual Basic pour Applications pour créer un nouvel objet.

5 La propriété Attributes de DAO est un masque binaire de plusieurs constantes correspondant à plusieurs propriétés dans le modèle ADOX. Pour connaître les informations de correspondance entre les constantes DAO et les propriétés ADOX, veuillez vous reporter aux sections "Création et modifications des tables" et "Garantie de l'intégrité référentielle".

6 Le nombre d'enregistrements dans une table peut être retrouvé via la colonne Cardinality dans les ensembles de lignes du schéma TABLES_INFO. Un comptage distinct est réalisé pour les index ; il peut être retrouvé via la colonne Cardinality dans l'ensemble de lignes du schéma INDEXES.

7 Dans ADOX, la table primaire d'une relation est représentée par l'objet Table qui contient un objet Key primaire dans sa collection Keys. La propriété Type des clés primaires est définie sur la valeur adKeyPrimary.

8 Contrairement à la propriété AllPermissions de DAO, la propriété GetPermissions de ADOX n'ajoute pas les autorisations dont l'utilisateur hérite des groupes auxquels il appartient. Ces informations doivent être extraites des objets Group.

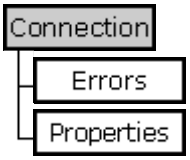
Annexe B : Modèle objet ADO détaillé

MSDN Home > MSDN Library > Data Access > Microsoft ActiveX Data Objects (ADO) > ADO Programmer's Reference > ADO API Reference > ADO Objects and Interfaces

ADO 2.8 API Reference

Connection Object

Represents an open connection to a data source.



Remarks

A **Connection** object represents a unique session with a data source. In the case of a client/server database system, it may be equivalent to an actual network connection to the server. Depending on the functionality supported by the provider, some collections, methods, or properties of a **Connection** object may not be available.

With the collections, methods, and properties of a **Connection** object, you can do the following:

- Configure the connection before opening it with the ConnectionString, ConnectionTimeout, and Mode properties. **ConnectionString** is the default property of the **Connection** object.
- Set the CursorLocation property to client to invoke the Microsoft Cursor Service for OLE DB, which supports batch updates.
- Set the default database for the connection with the DefaultDatabase property.
- Set the level of isolation for the transactions opened on the connection with the IsolationLevel property.
- Specify an OLE DB provider with the Provider property.
- Establish, and later break, the physical connection to the data source with the Open and Close methods.
- Execute a command on the connection with the Execute method and configure the execution with the CommandTimeout property.

  Note To execute a query without using a Command object, pass a query string to the Execute method of a Connection object. However, a Command object is required when you want to persist the command text and re-execute it, or use query parameters.
- Manage transactions on the open connection, including nested transactions if the provider supports them, with the BeginTrans, CommitTrans, and RollbackTrans methods and the Attributes property.
- Examine errors returned from the data source with the Errors collection.
- Read the version from the ADO implementation used with the Version property.
- Obtain schema information about your database with the OpenSchema method.

You can create **Connection** objects independently of any other previously defined object.

You can execute named commands or stored procedures as if they were native methods on a **Connection** object, as shown below. When a named command has the same name as that of a stored procedure, invoke the "native method call" on a **Connection** object always execute the named command instead of the store procedure.

Note Do not use this feature (calling a named command or stored procedure as if it were a native method on the **Connection** object) in a Microsoft® .NET Framework application, because the underlying implementation of the feature conflicts with the way the .NET Framework interoperates with COM.

Execute a command as a native method of a Connection object

To execute a command, give the command a name using the **Command** object Name property. Set the **Command** object's **ActiveConnection** property to the connection. Then issue a statement where the command name is used as if it were a method on the **Connection** object, followed by any parameters, then followed by a **Recordset** object if any rows are returned. Set the **Recordset** properties to customize the resulting **Recordset**. For example:

```
Dim cnn As New ADODB.Connection
Dim cmd As New ADODB.Command
Dim rst As New ADODB.Recordset
...
cnn.Open "..."
cmd.Name = "yourCommandName"
cmd.ActiveConnection = cnn
...
'Your command name, any parameters, and an optional Recordset.
cnn.yourCommandName "parameter", rst
```

Execute a stored procedure as a native method of a Connection object

To execute a stored procedure, issue a statement where the stored procedure name is used as if it were a method on the **Connection** object, followed by any parameters. ADO will make a "best guess" of parameter types. For example:

```
Dim cnn As New ADODB.Connection
...
'Your stored procedure name and any parameters.
cnn.sp_yourStoredProcedureName "parameter"
```

The **Connection** object is safe for scripting.

ADO 2.8 API Reference

Connection Object Properties, Methods, and Events

Properties/Collections

Attributes Property

CommandTimeout Property

ConnectionString Property

ConnectionTimeout Property

CursorLocation Property

DefaultDatabase Property

Errors Collection

IsolationLevel Property

Mode Property

Properties Collection

Provider Property

State Property

Version Property

Methods

BeginTrans, CommitTrans, and RollbackTrans Methods

Cancel Method

Close Method

Execute Method (ADO Connection)

Open Method (ADO Connection)

OpenSchema Method

Events

BeginTransComplete, CommitTransComplete, and RollbackTransComplete Events

ConnectComplete and Disconnect Events

ExecuteComplete Event

InfoMessage Event

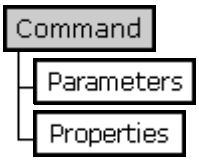
WillConnect Event

WillExecute Event

ADO 2.8 API Reference

Command Object

Defines a specific command that you intend to execute against a data source.



Remarks

Use a **Command** object to query a database and return records in a Recordset object, to execute a bulk operation, or to manipulate the structure of a database. Depending on the functionality of the provider, some **Command** collections, methods, or properties may generate an error when referenced.

With the collections, methods, and properties of a **Command** object, you can do the following:

- Define the executable text of the command (for example, an SQL statement) with the CommandText property. Alternatively, for command or query structures other than simple strings (for example, XML template queries) define the command with the CommandStream property.
- Optionally, indicate the command dialect used in the **CommandText** or **CommandStream** with the Dialect property.
- Define parameterized queries or stored-procedure arguments with Parameter objects and the Parameters collection.
- Indicate whether parameter names should be passed to the provider with the NamedParameters property.
- Execute a command and return a **Recordset** object if appropriate with the Execute method.
- Specify the type of command with the CommandType property prior to execution to optimize performance.
- Control whether the provider saves a prepared (or compiled) version of the command prior to execution with the Prepared property.
- Set the number of seconds that a provider will wait for a command to execute with the CommandTimeout property.
- Associate an open connection with a **Command** object by setting its ActiveConnection property.
- Set the Name property to identify the **Command** object as a method on the associated Connection object.
- Pass a **Command** object to the Source property of a **Recordset** in order to obtain data.
- Access provider-specific attributes with the Properties collection.

Note To execute a query without using a **Command** object, pass a query string to the Execute method of a **Connection** object or to the Open method of a **Recordset** object. However, a **Command** object is required when you want to persist the command text and re-execute it, or use query parameters.

To create a **Command** object independently of a previously defined **Connection** object, set its **ActiveConnection** property to a valid connection string. ADO still creates a **Connection** object, but it doesn't assign that object to an object variable. However, if you are associating multiple **Command** objects with the same connection, you should explicitly create and open a **Connection** object; this assigns the **Connection**

object to an object variable. Make sure the **Connection** object was opened successfully before you assign it to the **Command** object's **ActiveConnection** property, because assigning a closed **Connection** object causes an error. If you do not set the **Command** object's **ActiveConnection** property to this object variable, ADO creates a new **Connection** object for each **Command** object, even if you use the same connection string.

To execute a **Command**, simply call it by its Name property on the associated **Connection** object. The **Command** must have its **ActiveConnection** property set to the **Connection** object. If the **Command** has parameters, pass their values as arguments to the method.

If two or more **Command** objects are executed on the same connection and either **Command** object is a stored procedure with output parameters, an error occurs. To execute each **Command** object, use separate connections or disconnect all other **Command** objects from the connection.

The **Parameters** collection is the default member of the **Command** object. As a result, the following two code statements are equivalent.

```
objCmd.Parameters.Item(0)

objCmd(0)
```

The **Command** object is not safe for scripting.

ADO 2.8 API Reference

Command Object Properties, Methods, and Events

Properties/Collections

ActiveConnection Property

CommandStream Property

CommandText Property

CommandTimeout Property

CommandType Property

Dialect Property

Name Property

NamedParameters Property

Parameters Collection

Prepared Property

Properties Collection

State Property

Methods

Cancel Method

CreateParameter Method

Execute Method (ADO Command)

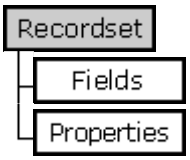
Events

None.

ADO 2.8 API Reference

Recordset Object

Represents the entire set of records from a base table or the results of an executed command. At any time, the **Recordset** object refers to only a single record within the set as the current record.



Remarks

You use **Recordset** objects to manipulate data from a provider. When you use ADO, you manipulate data almost entirely using **Recordset** objects. All **Recordset** objects consist of records (rows) and fields (columns). Depending on the functionality supported by the provider, some **Recordset** methods or properties may not be available.

ADODB.Recordset is the ProgID that should be used to create a **Recordset** object. Existing applications that reference the outdated ADOR.Recordset ProgID will continue to work without recompiling, but new development should reference ADODB.Recordset.

There are four different cursor types defined in ADO:

- **Dynamic cursor** — allows you to view additions, changes, and deletions by other users; allows all types of movement through the **Recordset** that doesn't rely on bookmarks; and allows bookmarks if the provider supports them.

- **Keyset cursor** — behaves like a dynamic cursor, except that it prevents you from seeing records that other users add, and prevents access to records that other users delete. Data changes by other users will still be visible. It always supports bookmarks and therefore allows all types of movement through the **Recordset**.

- **Static cursor** — provides a static copy of a set of records for you to use to find data or generate reports; always allows bookmarks and therefore allows all types of movement through the **Recordset**. Additions, changes, or deletions by other users will not be visible. This is the only type of cursor allowed when you open a client-side **Recordset** object.

- **Forward-only cursor** — allows you to only scroll forward through the **Recordset**. Additions, changes, or deletions by other users will not be visible. This improves performance in situations where you need to make only a single pass through a **Recordset**.

Set the CursorType property prior to opening the **Recordset** to choose the cursor type, or pass a CursorType argument with the Open method. Some providers don't support all cursor types. Check the documentation for the provider. If you don't specify a cursor type, ADO opens a forward-only cursor by default.

If the CursorLocation property is set to **adUseClient** to open a **Recordset**, the **UnderlyingValue** property on Field objects is not available in the returned **Recordset** object. When used with some providers (such as the Microsoft ODBC Provider for OLE DB in conjunction with Microsoft SQL Server), you can create **Recordset** objects independently of a previously defined Connection object by passing a connection string with the **Open** method. ADO still creates a Connection object, but it doesn't assign that object to an object variable. However, if you are opening multiple **Recordset** objects over the same connection, you should explicitly create and open a **Connection** object; this assigns the **Connection** object to an object variable. If you do not use this object variable when opening your **Recordset** objects, ADO creates a new **Connection** object for each new **Recordset**, even if you pass the same connection string.

You can create as many **Recordset** objects as needed.

When you open a **Recordset**, the current record is positioned to the first record (if any) and the BOF and EOF properties are set to **False**. If there are no records, the **BOF** and **EOF** property settings are **True**.

You can use the MoveFirst, **MoveLast**, **MoveNext**, and **MovePrevious** methods; the Move method; and the AbsolutePosition, AbsolutePage, and Filter properties to reposition the current record, assuming the provider

supports the relevant functionality. Forward-only **Recordset** objects support only the MoveNext method. When you use the **Move** methods to visit each record (or enumerate the **Recordset**), you can use the **BOF** and **EOF** properties to determine if you've moved beyond the beginning or end of the **Recordset**.

Before using any functionality of a **Recordset** object, you must call the **Supports** method on the object to verify that the functionality is supported or available. You must not use the functionality when the **Supports** method returns false. For example, you can use the **MovePrevious** method only if Recordset.Supports(adMovePrevious) returns true. Otherwise, you will get an error, because the **Recordset** object might have been closed and the functionality rendered unavailable on the instance. If a feature you are interested in is not supported, **Supports** will return false as well. In this case, you should avoid calling the corresponding property or method on the **Recrodset** object.

Recordset objects can support two types of updating: immediate and batched. In immediate updating, all changes to data are written immediately to the underlying data source once you call the Update method. You can also pass arrays of values as parameters with the AddNew and **Update** methods and simultaneously update several fields in a record.

If a provider supports batch updating, you can have the provider cache changes to more than one record and then transmit them in a single call to the database with the UpdateBatch method. This applies to changes made with the **AddNew**, **Update**, and Delete methods. After you call the **UpdateBatch** method, you can use the Status property to check for any data conflicts in order to resolve them.

Note To execute a query without using a Command object, pass a query string to the **Open** method of a **Recordset** object. However, a **Command** object is required when you want to persist the command text and re-execute it, or use query parameters.

The Mode property governs access permissions.

The **Fields** collection is the default member of the **Recordset** object. As a result, the following two code statements are equivalent.

```
Debug.Print objRs.Fields.Item(0)   ' Both statements print
Debug.Print objRs(0)               '  the Value of Item(0).
```

When a **Recordset** object is passed across processes, only the **rowset** values are marshalled, and the properties of the **Recordset** object are ignored. During unmarshalling, the **rowset** is unpacked into a newly created **Recordset** object, which also sets its properties to the default values.

The **Recordset** object is safe for scripting.

Recordset Object Properties, Methods, and Events

Properties/Collections

AbsolutePage Property

AbsolutePosition Property

ActiveCommand Property

ActiveConnection Property

BOF, EOF Properties

Bookmark Property

CacheSize Property

CursorLocation Property

CursorType Property

DataMember Property

DataSource Property

DeleteCommand Property

EditMode Property

Fields Collection

Filter Property

Index Property

InsertCommand Property

LockType Property

MarshalOptions Property

MaxRecords Property

PageCount Property

PageSize Property

Properties Collection

RecordCount Property

Sort Property

Source Property (ADO Recordset)

State Property

Status Property (ADO Recordset)

StayInSync Property

UpdateCommand Property

Methods

AddNew Method

Cancel Method

CancelBatch Method

CancelUpdate Method
Clone Method
Close Method
CompareBookmarks Method
Delete Method (ADO Recordset)
Find Method
GetRows Method
GetString Method
Move Method
MoveFirst, MoveLast, MoveNext, and MovePrevious Methods
NextRecordset Method
Open Method (ADO Recordset)
Requery Method
Resync Method
Save Method
Seek Method
SetAllRowStatus Method
Supports Method
Update Method
UpdateBatch Method

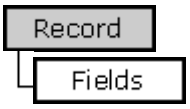
Events

EndOfRecordset Event
FetchComplete Event
FetchProgress Event
WillChangeField and FieldChangeComplete Events
WillChangeRecord and RecordChangeComplete Events
WillChangeRecordset and RecordsetChangeComplete Events
WillMove and MoveComplete Events

ADO 2.8 API Reference

Record Object

Represents a row from a Recordset or the data provider, or an object returned by a semi-structured data provider, such as a file or directory.



Remarks

A **Record** object represents one row of data, and has some conceptual similarities with a one-row **Recordset**. Depending upon the capabilities of your provider, **Record** objects may be returned directly from your provider instead of a one-row **Recordset**, for example when an SQL query that selects only one row is executed. Or, a **Record** object can be obtained directly from a **Recordset** object. Or, a **Record** can be returned directly from a provider to semi-structured data, such as the Microsoft Exchange OLE DB provider.

You can view the fields associated with the **Record** object by way of the Fields collection on the **Record** object. ADO allows object-valued columns including **Recordset**, **SafeArray**, and scalar values in the **Fields** collection of **Record** objects.

If the **Record** object represents a row in a **Recordset**, then it is possible to return to that original **Recordset** with the Source property.

The **Record** object can also be used by semi-structured data providers such as the Microsoft OLE DB Provider for Internet Publishing, to model tree-structured namespaces. Each node in the tree is a **Record** object with associated columns. The columns can represent the attributes of that node and other relevant information. The **Record** object can represent both a leaf node and a non-leaf node in the tree structure. Non-leaf nodes have other nodes as their contents while leaf nodes do not have such contents. Leaf nodes typically contain binary streams of data while non-leaf nodes may also have a default binary stream associated with them. Properties on the **Record** object identify the type of node.

The **Record** object also represents an alternative way for navigating hierarchically organized data. A **Record** object may be created to represent the root of a specific sub-tree in a large tree structure and new **Record** objects may be opened to represent child nodes.

A resource (for example, a file or directory) can be uniquely identified by an absolute URL. A Connection object is implicitly created and set to the **Record** object when the **Record** is opened with an absolute URL. A **Connection** object may explicitly be set to the **Record** object via the ActiveConnection property. The files and directories accessible via the **Connection** object define the context in which **Record** operations may occur.

Data modification and navigation methods on the **Record** object also accept a relative URL, which locates a resource using an absolute URL or the **Connection** object context as a starting point.

Note URLs using the http scheme will automatically invoke the Microsoft OLE DB Provider for Internet Publishing. For more information, see Absolute and Relative URLs.

A **Connection** object is associated with each **Record** object. Therefore, **Record** object operations can be part of a transaction by invoking **Connection** object transaction methods.

The **Record** object does not support ADO events, and therefore will not respond to notifications.

With the methods and properties of a **Record** object, you can do the following:

- Set or return the associated **Connection** object with the ActiveConnection property.
- Indicate access permissions with the Mode property.
- Return the URL of the directory, if any, that contains the resource represented by the **Record** with the ParentURL property.
- Indicate the absolute URL, relative URL, or **Recordset** from which the **Record** is derived with the Source property.

- Indicate the current status of the **Record** with the State property.
- Indicate the type of **Record**—simple, collection, or structured document—with the RecordType property.
- Halt execution of an asynchronous operation with the Cancel method.
- Disassociate the **Record** from a data source with the Close method.
- Copy the file or directory represented by a **Record** to another location with the CopyRecord method.
- Delete the file, or directory and subdirectories, represented by a **Record** with the DeleteRecord method.
- Open a **Recordset** containing rows that represent the subdirectories and files of the entity represented by the **Record** with the GetChildren method.
- Move (rename) the file, or directory and subdirectories, represented by a **Record** to another location with the MoveRecord method.
- Associate the **Record** with an existing data source, or create a new file or directory with the Open method.

The **Record** object is safe for scripting.

Record Object Properties, Methods, and Events

Properties/Collections

ActiveConnection Property

Fields Collection

Mode Property

ParentURL Property

Properties Collection

RecordType Property

Source Property (ADO Record)

State Property

Methods

Cancel Method

Close Method

CopyRecord Method

DeleteRecord Method

GetChildren Method

MoveRecord Method

Open Method (ADO Record)

Events

None.

ADO 2.8 API Reference

Stream Object

Represents a stream of binary data or text.

Remarks

In tree-structured hierarchies such as a file system or an e-mail system, a Record may have a default binary stream of bits associated with it that contains the contents of the file or the e-mail. A **Stream** object can be used to manipulate fields or records containing these streams of data. A **Stream** object can be obtained in these ways:

- From a URL pointing to an object (typically a file) containing binary or text data. This object can be a simple document, a **Record** object representing a structured document, or a folder.
- By opening the default **Stream** object associated with a **Record** object. You can obtain the default stream associated with a **Record** object when the **Record** is opened, to eliminate a round-trip just to open the stream.
- By instantiating a **Stream** object. These **Stream** objects can be used to store data for the purposes of your application. Unlike a **Stream** associated with a URL, or the default **Stream** of a **Record**, an instantiated **Stream** has no association with an underlying source by default.

With the methods and properties of a **Stream** object, you can do the following:

- Open a **Stream** object from a **Record** or URL with the Open method.
- Close a **Stream** with the Close method.
- Input bytes or text to a **Stream** with the Write and WriteText methods.
- Read bytes from the **Stream** with the Read and ReadText methods.
- Write any **Stream** data still in the ADO buffer to the underlying object with the Flush method.
- Copy the contents of a **Stream** to another **Stream** with the CopyTo method.
- Control how lines are read from the source file with the SkipLine method and the LineSeparator property.
- Determine the end of stream position with the EOS property and SetEOS method.
- Save and restore data in files with the SaveToFile and LoadFromFile methods.
- Specify the character set used for storing the **Stream** with the Charset property.
- Halt an asynchronous **Stream** operation with the Cancel method.
- Determine the number of bytes in a **Stream** with the Size property.
- Control the current position within a **Stream** with the Position property.
- Determine the type of data in a **Stream** with the Type property.
- Determine the current state of the **Stream** (closed, open, or executing) with the State property.
- Specify the access mode for the **Stream** with the Mode property.

Note URLs using the http scheme will automatically invoke the Microsoft OLE DB Provider for Internet Publishing. For more information, see Absolute and Relative URLs.

The **Stream** object is safe for scripting.

ADO 2.8 API Reference

Stream Object Properties, Methods, and Events

Properties

Charset Property

EOS Property

LineSeparator Property

Mode Property

Position Property

Size Property (ADO Stream)

State Property

Type Property (ADO Stream)

Methods

Cancel Method

Close Method

CopyTo Method

Flush Method

LoadFromFile Method

Open Method (ADO Stream)

Read Method

ReadText Method

SaveToFile Method

SetEOS Method

SkipLine Method

Stat Method

Write Method

WriteText Method

Events

None.