DELPHI 6

Support de formation (2/3)

Les bases de données

1. CREATION D'UNE BASE

1.1 Rappels

Une base de données est une collection d'objets (exemple: recettes de cuisine, unités centrales, relations d'affaires, articles vendus, etc.). Pour que cette collection soit intéressante, il est souhaitable que les objets présentent au moins une propriété ou méthode commune. Une base de donnée est composée de champs (propriétés) et d'enregistrements (chaque objet). Pour créer une base, il est nécessaire de définir les champs (type, longueur, index, ...). Ensuite, il n'y a qu'à saisir les enregistrements.

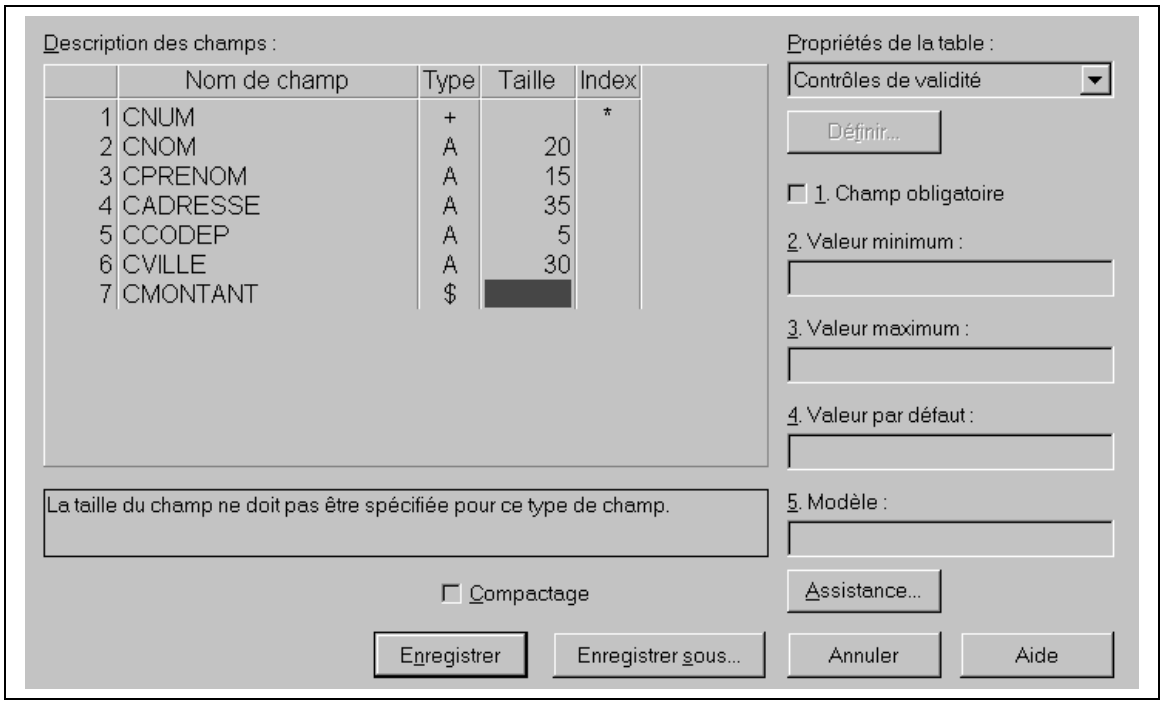
Les index permettent de retrouver rapidement un enregistrement. On distingue l'index principal (de préférence unique, c'est souvent un compteur) et les index secondaires qui évitent la lecture de tous les enregistrements lors de la recherche. Lors de l'affichage des enregistrements, le tri se fait sur l'index principal (ou secondaire si spécifié)

On distingue deux modes d'exploitation: mono-base (une seule base est exploitée) et multi-bases (plusieurs bases ou tables sont mises en relation).

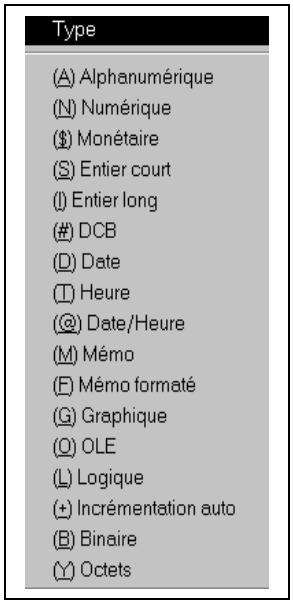
1.2 Utilisation de l'outil intégré

Dans le menu "Outils", choisir "Module Bases de données". Attendre le chargement de l'utilitaire qui est très proche de PARADOX 7 (Borland). Dans le nouveau menu "Fichier", choisir "Nouveau" puis "Table" et enfin "Paradox 7".

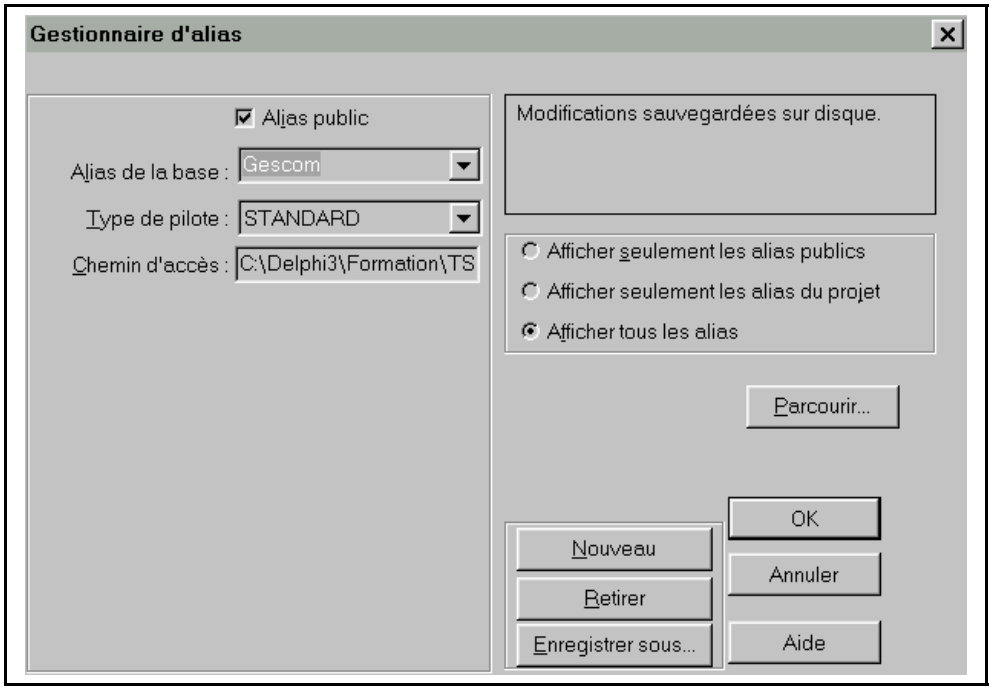
L'écran suivant s'affiche et permet de définir les champs.



Les types de champ peuvent être les suivants (table de type Paradox):



Une fois la table enregistrée, il est préférable de donner un alias ("Gescom") à son répertoire, ce qui permettra de la retrouver plus rapidement. Dans le menu "Outils" choisir "Gestionnaire d'alias" pour afficher la boîte de dialogue suivante. Pour le chemin d'accès, on peut cliquer sur le bouton "Parcourir".

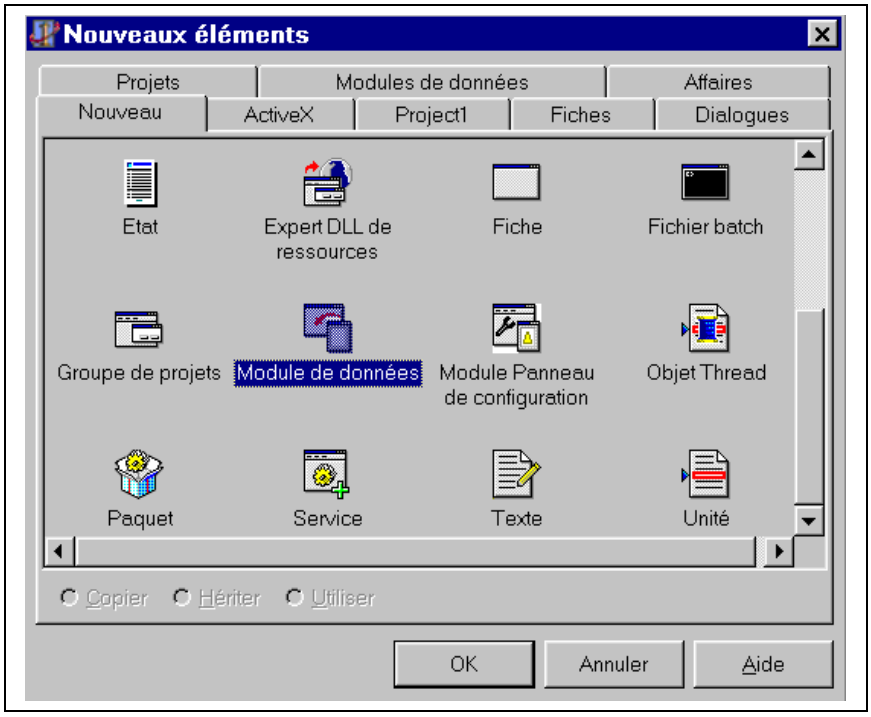


Valider. On peut saisir quelques enregistrements. Pour ce faire, choisir "Fichier" puis "Ouvrir" et "Table". L'alias créé nous sert déjà. Pour saisir, il suffit de passer en mode édition par l'option "Editer les données" dans le menu "Table". Après saisie, fermer et quitter.

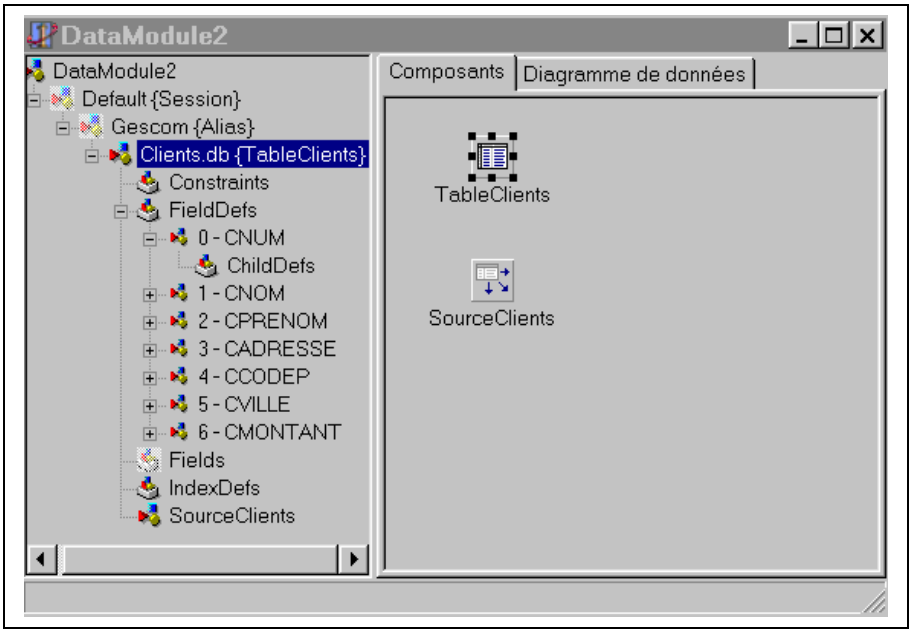
2. EXPLOITATION AVEC DELPHI

2.1 Visualisation de la base

Créer un nouveau projet Delphi.
Choisir Fichier - Nouveau - Module données



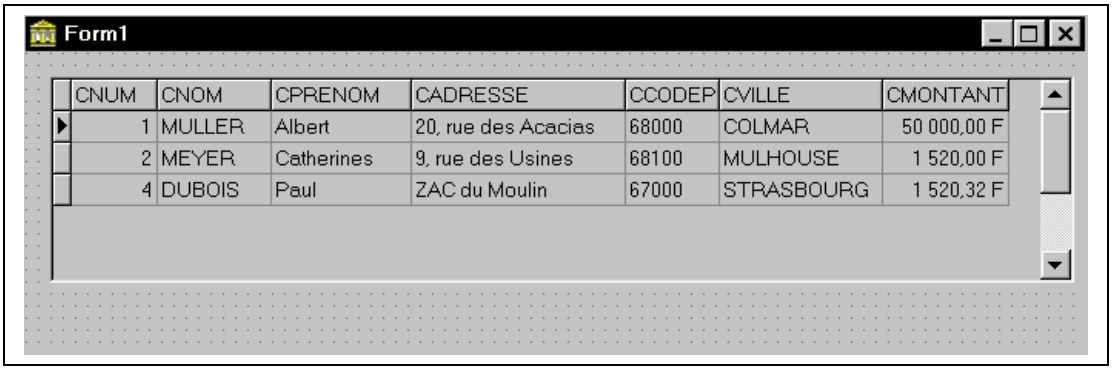
Y placer un composant "TTable" et "TDatasource"



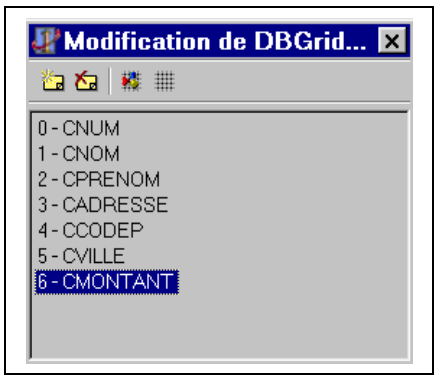
Attribuer au premier les propriétés "DatabaseName" à "Gescom" (alias choisi dans la liste déroulante), "TableName" à "Ficli.db" (choisi); "Name" à "TableClients".

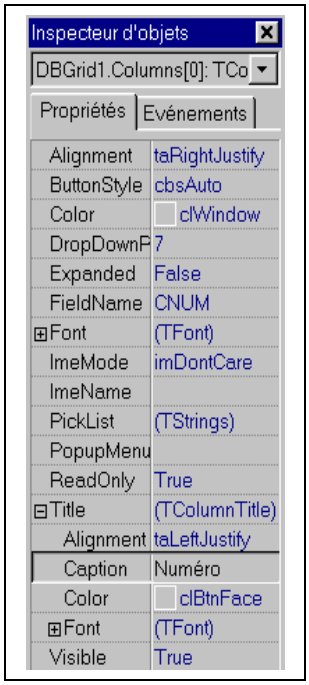
Pour le second, il suffit de mettre "DataSet" à "TableClients" et "Name" à "SourceClients".

A) Se placer dans la fiche (Form1).
B) Fichier Utiliser Unit2 (c'est le nom du module de données).
C) Placer un composant "TDBGrid" et en affecter la propriété "DataSource" à "Datasource1" (nom du composant précédent).
D) Sélectionner le composant "TableClients" dans le module de données et mettre "Active" à "true". Si tout est correct, les données apparaissent dans la grille.



Les colonnes peuvent être redimensionnées et déplacées à l'exécution. Pour le faire dans l'étape de conception, il faut utiliser le menu contextuel de la grille: éditeur de colonnes. Choisir "ajouter tous les champs" pour obtenir l'affichage suivant:





L'éditeur de propriétés permet de modifier certaines propriétés, notamment le nom du titre de la colonne. Il est également possible de prévoir des listes déroulantes (PickList) et des boutons d'action (événement OnEditButtonClick) pour chaque champ.

Exercice: changer le nom des colonnes et mettre une liste déroulante de villes (Mulhouse, Colmar, Strasbourg).

Enregistrer le projet, exécuter. Vérifier l'ajout et la modification.

2.1.1 Accès aux champs

On peut accéder au contenu d'un champ, soit par son indice, soit par son nom. Le résultat renvoyé est toujours de type **variant** (transtypage à effectuer). Pour utiliser le variant, il est possible de se référer à la propriété "value" du champ.

- Accès par indice: NomTable.Fields[Indice]
- Accès par nom: NomTableNomChamp ou NomTable.FieldByName('NomChamp')
  Exemple: *Edit1.Text:=Table1Nom.AsString;*

Transtypages: AsBoolean, AsCurrency, AsDateTime, AsFloat, AsInteger, AsString.

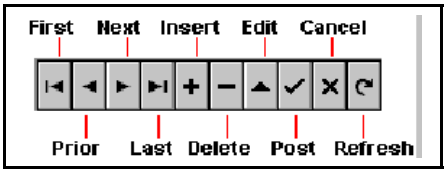
Positionnement sur un champ: Il se fait par la méthode FocusControl du champ.

Rq: accès au champ sélectionné dans 1 grille: DbGrid1.SelectedField.

2.2 Navigation dans une base

2.2.1 Avec le composant dédié

Placer un composant "TDBNavigator" et positionner sa propriété "Datasource" à "SourceClients" (nom de la source utilisée).



BOUTON	ROLE
Premier (first)	Le premier enregistrement de l'ensemble de données devient l'enregistrement courant, les boutons Premier et Précédent sont désactivés et les boutons Suivant et Dernier sont activés
Précédent (prior)	L'enregistrement précédent devient l'enregistrement courant, les boutons Dernier et Suivant sont activés
Suivant (next)	L'enregistrement suivant devient l'enregistrement courant et les boutons Premier et Précédent sont activés
Dernier (last)	Le dernier enregistrement de l'ensemble de données devient l'enregistrement courant, les boutons Dernier et Suivant sont désactivés et les boutons Premier and Précédent sont activés
Insérer (insert)	Insère un nouvel enregistrement qui précède l'enregistrement courant et bascule l'ensemble de données en mode insertion et modification
Supprimer (delete)	Supprime l'enregistrement courant et l'enregistrement suivant devient l'enregistrement courant
Modifier (edit)	Bascule l'ensemble de données en mode modification pour que l'enregistrement courant puisse être modifié
Emettre (post)	Ecrit les modifications de l'enregistrement courant dans la base de données
Annuler (cancel)	Annule les modifications dans l'enregistrement courant et restitue l'état de l'affichage de l'enregistrement tel qu'il était avant la modification, désactive les modes insertion et modification s'ils sont actifs
Rafraîchir (refresh)	Réaffiche l'enregistrement courant de l'ensemble de données, provoquant ainsi la mise à jour de l'affichage de l'enregistrement sur la fiche

Remarque: je n'ai pas de bouton pour ajouter immédiatement un enregistrement à la fin, mais la méthode existe (append).

2.2.2 En créant ses propres outils

Une base de donnée peut être exploitée en mode liste comme précédemment, mais également en mode formulaire plus propice à la saisie.

Créer un nouveau projet. A l'aide du gestionnaire de projet, ajouter le module de données défini précédemment au cours de l'exercice précédent. Faire utiliser ce module par la fiche principale (Form1). Demander un menu contextuel sur le composant TTable et choisir l'éditeur de champs. Dans le menu contextuel de ce dernier, choisir "ajouter champs". On peut choisir ici les champs qui seront utilisés dans la fiche (les choisir tous).

Il est alors enfantin de placer les champs dans la fiche par cliquer-glisser, individuellement ou collectivement (selon sélection).

Placer des boutons pour exploiter la table: premier, précédent, suivant, dernier. Les procédures existent et peuvent être attribuées à DataModule2.TableClients (ex: DataModule2.TableClients.First) : first, prior, next, last.

Rajouter des boutons:

- insérer (insert) et ajouter à la fin (append),
- modifier (edit)
- valider (post),
- abandonner (cancel),
- détruire (delete),
- mise à jour (refresh),
- et un bouton pour quitter l'application.

Enregistrer le projet et exécuter.
Options:

- rajouter un menu et une barre d'outils qui vont remplacer les boutons
- écrire un gestionnaire commun unique pour les événements
- sécuriser la suppression ("MessageDlg")
- afficher en rouge les montants supérieurs à 5000 francs

2.3 Etats des tables

Suivant les opérations effectuées (par exemple avec le navigateur), la table (plutôt le "Dataset") se place dans différents états renvoyés par sa propriété "State".

VALEUR	SIGNIFICATION
dsInactive	L'ensemble de données est fermé ; ses données sont donc inaccessibles.
dsBrowse	Il est possible de consulter mais pas de modifier les données. C'est la valeur par défaut à l'ouverture d'un ensemble de données.
dsEdit	L'enregistrement actif peut être modifié.
dsInsert	Il est possible d'insérer un nouvel enregistrement.
dsSetKey	TTable uniquement. La recherche d'enregistrement est activée ou une opération SetRange est en cours. Seule la visualisation d'un sous-ensemble des enregistrements est possible, mais pas la modification ou l'ajout de données.
dsCalcFields	Un événement OnCalcFields est en cours. Les champs non-calculés ne peuvent être modifiés et il n'est pas possible d'ajouter de nouveaux enregistrements.
dsFilter	Un événement OnFilterRecord a lieu. Seule la visualisation d'un sous-ensemble des enregistrements est possible, mais pas la modification ou l'ajout de données.

2.4 Tri sur index dans une table

Le tri se fait automatiquement sur l'index principal. Si l'on désire utiliser un index secondaire, il faut fermer préalablement la table. Exemple:

```
Table1.close;
Table1.IndexName:='NomIndexSecondaire';
Table1.open;
```

2.5 Recherche dans une table sur un champ indexé

Il existe différentes instructions. Parmi elles, on note "SetKey" qui permet d'attribuer une valeur de recherche à la propriété "Fields" de la table suivie de "GotoKey" pour une recherche exacte et " GotoNearest" pour une recherche approchée. Dans tous les cas, une valeur "false" est retournée si la recherche n'a pas pu aboutir. Exemples:

```
Table1.SetKey;
Table1.['NomChampIndex']:=Edit1.Text;
if not Table1.GotoKey then showmessage('pas vu');
```

```
Table1.Setkey;
Table1.['NomChampIndex']:=Edit1.Text;
Table1.GotoNearest;
```

2.6 Etablissement d'un filtre

On peut utiliser la propriété "Filter" pour définir le filtre (ex: *NOM='B*'*). Pour lancer le filtrage, il faut mettre "Filtered" à "true".

2.7 Utilisation de signets

Les signets permettent de mémoriser un emplacement dans la table et d'y revenir rapidement:

- Déclaration: *var Signet1:TBookmark;*
- Pose: S*ignet1:=NomTable.GetBookmark;*
- Accès: *NomTable.GotoBookmark(Signet1);*
- Suppression: *NomTable.FreeBookmark(Signet1);*

3. REQUETES ET TRIS

3.1 Préparation du projet

Le mot requête vient de question. Les requêtes servent donc à poser des questions (ex: quelles personnes habitent à COLMAR?). Delphi comporte une classe "TQuery" qui facilite les requêtes.

Pour disposer d'une base importante, charger la base "baseadr.dbf" (format DBASE). Lui donner un alias "baseadr"(standard) avec l'utilitaire BDE.

Créer un nouveau projet avec un module de données comportant les composants: TQuery, TDatasource.

Placer une grille de données et un navigateur sur la fiche. Attribuer les propriétés nécessaires.

Il ne manque plus que le texte SQL (Structured Query Langage)

3.2 Quelques instructions SQL

SELECT permet d'extraire des champs ou des parties de champ d'une ou de plusieurs tables.

INSTRUCTION	SIGNIFICATION
SELECT	permet de choisir les champs désirés.
FROM	est suivi par le nom de la table.
WHERE	donne les conditions de recherche
GROUP BY	regroupe des enregistrements; est utilisé avec HAVING.
HAVING	restreint les colonnes générées par GROUP BY.
UNION	permet d'utiliser plusieurs instructions SELECT pour générer une seule table dynamique sans colonnes dupliquées.
ORDER BY	spécifie les champs de tri ascendant (ASC) ou descendant (DESC)

3.3 Mise en place de la requête

Cliquer sur les points de suspension de la propriété SQL de Query1. L'éditeur de listes de chaînes s'affiche. On veut afficher tous les champs et tous les enregistrements, ce qui donne l'instruction SQL suivante:

*select * from baseadr*

Si l'on met la propriété "Active" à "True", le résultat s'affiche dans la grille.

La propriété "RequestLive" est positionnée à "false" (lecture seule). Il est possible, sous certaines conditions, de renvoyer des données dans la table en la mettant à "true".

3.4 Modification de la requête à l'exécution

Enregistrer le projet sous un nouveau répertoire

Placer 4 boutons sur la fiche.

Ils provoqueront respectivement l'affichage:
- des habitants de Colmar
- des personnes d'Andolsheim ayant payé plus de 3000 francs par chèque
- des personnes dont le nom commence par "M"
- de toute la table triée par ordre alphabétique de nom et de prénom.

Pour modifier une requête en phase d'exécution, il faut tout d'abord la fermer avec l'instruction "close" (ex: DataModule2.Query1.close).

Ensuite, effacer le texte SQL (ex: DataModule2.Query1.SQL.clear). Il suffit alors d'ajouter les nouvelles instructions (ex: DataModule2.Query1.SQL.Add('*texte'*)) et de lancer la requête (ex: DataModule2.Query1.open).

Les listings nous donnent:

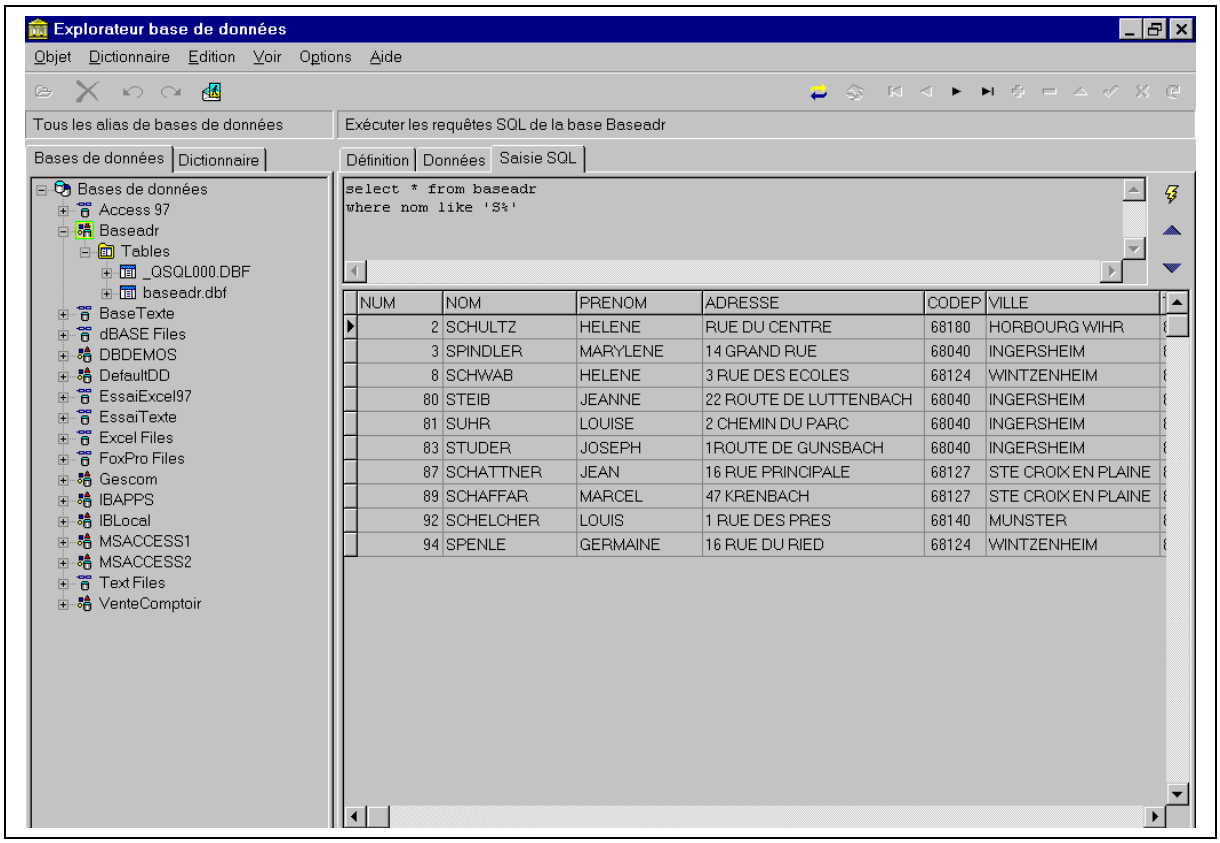
```
procedure TForm1.Button1Click(Sender: TObject);
begin
        DataModule2.Query1.close;
        DataModule2.Query1.Sql.clear;
        DataModule2.Query1.Sql.Add('select * from baseadr where
        VILLE=''COLMAR''');
        DataModule2.Query1.Open;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
        with DataModule2.Query1 do
          begin
          close;
          Sql.clear;
          Sql.Add('select * from baseadr where
        VILLE=''ANDOLSHEIM''');
          Sql.Add('and FINANCEMEN=''CHEQUE'' and
        MONTANT>3000');
          Open;
          end;
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
   DataModule2.Query1.close;
   DataModule2.Query1.Sql.clear;
   DataModule2.Query1.Sql.Add('select * from baseadr where NOM
   like ''M%'' ');
   DataModule2.Query1.Open;
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
        DataModule2.Query1.close;
        DataModule2.Query1.Sql.clear;
        DataModule2.Query1.Sql.Add('select * from baseadr order by
        NOM,PRENOM');
        DataModule2.Query1.Open;
end;
```

Il est très facile de tester le fonctionnement d'une requête avec l'explorateur de bases de données (Base de données - explorateur). Ouvrir l'alias désiré puis choisir la table. L'onglet "Données" présente les enregistrements (modifiables) et l'onglet "Saisie SQL" permet de saisir le texte SQL des requêtes. Pour lancer une exécution, cliquer sur le bouton affichant un éclair. Quand la requête est correcte, l'enregistrer au moyen du menu contextuel (donner un nom significatif).



La propriété "Sql" du composant "Query" est de type "TStrings": liste de chaînes comme les lignes d'un "Memo". Je vois une méthode intéressante "LoadFromFile"…

Modifier le projet précédent pour utiliser cette technique plus élégante. En effet, les requêtes sont maintenant stockées sous la forme de fichiers texte dont la maintenance et la modification sont faciles (par exemple avec le bloc-notes).

3.5 Requêtes paramétrées

Pour définir une requête paramétrée, il suffit de préciser dans l'instruction SQL le nom du paramètre précédé par un double point (ex: "select * from clients where nom = :param1).

L'accès aux paramètres se fait par un tableau indicé à partir de zéro. La première valeur sera donc "Params[0]".

Rajouter un composant de type TTabControl et mettre 26 onglets de A à Z. Le but est de n'afficher que les enregistrements dont le nom commence par le lettre de l'onglet. Le listing du programme peut être le suivant:

```
procedure TForm1.TabControl1Change(Sender: TObject);
begin
with DataModule2.Query1 do
        begin
        close;
        Sql.clear;
        Sql.Add('select * from baseadr');
        Sql.Add('where NOM like :truc');
        Params[0].AsString:=
          Form1.TabControl1.Tabs[Form1.TabControl1.Tabindex]+'%';
        open;
        end;
end;
```

Il est nécessaire de spécifier le type de paramètre ("AsString"). Le numéro de l'onglet sélectionné est donné par la propriété "Tabindex" et la chaîne associée par la propriété "Tabs".

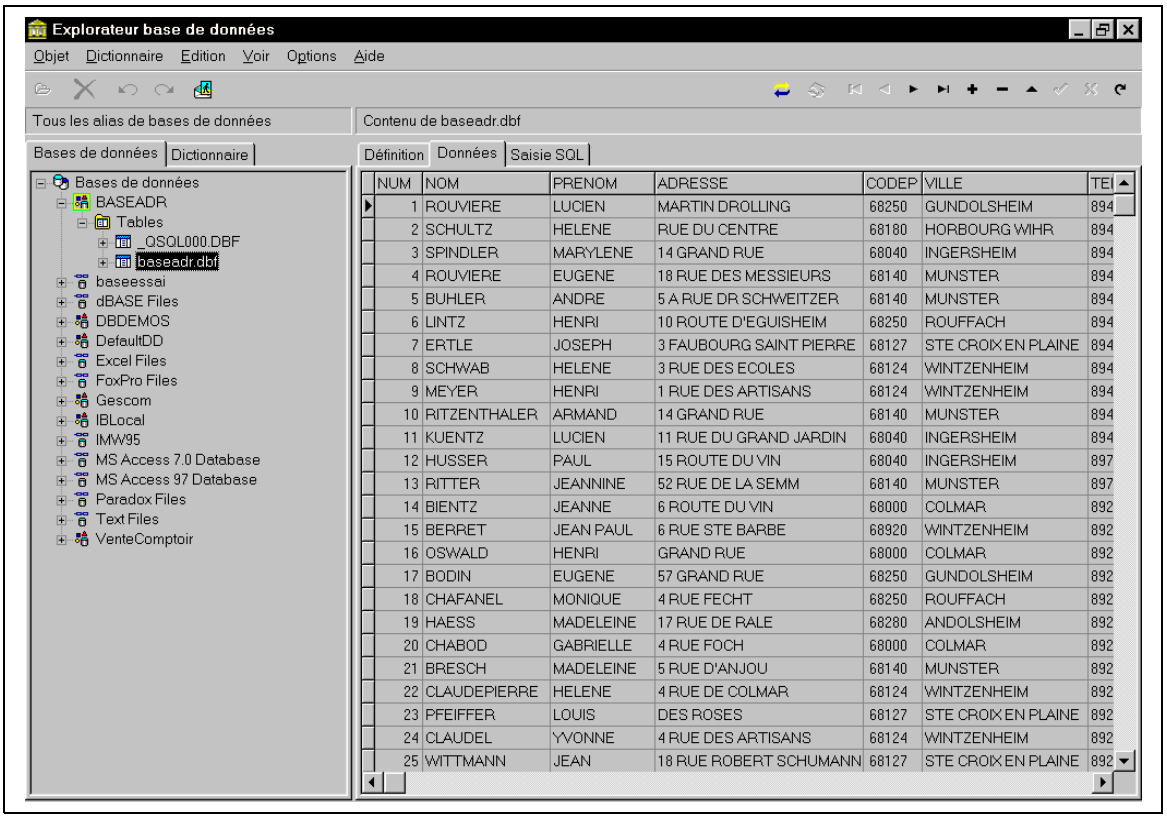
4. ETATS

Les états sont les impressions obtenues à partir de bases de données. Delphi possède un outils puissant: Quickreport (consulter le manuel en ligne pour plus de précisions). C'est un ensemble de composants prêts à l'emploi disponibles dans la palette.

4.1 Définition de l'état

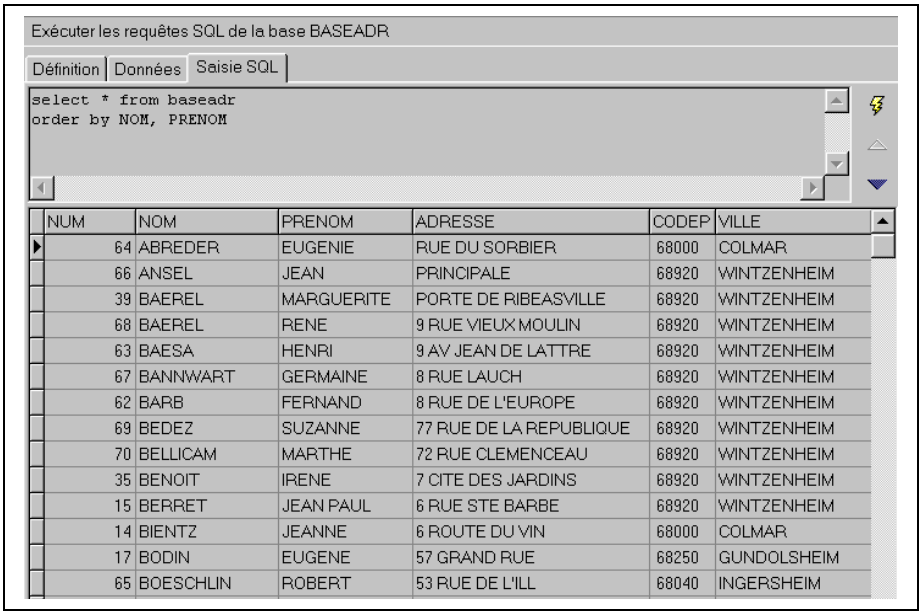
On veut imprimer une liste des clients avec nom, prénom, code postal, ville et montant.

Créer un nouveau projet. Placer un composant "TQuery" relié à "Baseadr". Dans le menu contextuel, choisir explorer. L'explorateur de bases de données s'ouvre.



NUM	NOM	PRENOM	ADRESSE	CODEP	VILLE	TEI
1	ROUVIERE	LUCIEN	MARTIN DROLLING	68250	GUNDOLSHEIM	894
2	SCHULTZ	HELENE	RUE DU CENTRE	68180	HORBOURG WIHR	894
3	SPINDLER	MARYLENE	14 GRAND RUE	68040	INGERSHEIM	894
4	ROUVIERE	EUGENE	18 RUE DES MESSIEURS	68140	MUNSTER	894
5	BUHLER	ANDRE	5 A RUE DR SCHWEITZER	68140	MUNSTER	894
6	LINTZ	HENRI	10 ROUTE D'EGUISHEIM	68250	ROUFFACH	894
7	ERTLE	JOSEPH	3 FAUBOURG SAINT PIERRE	68127	STE CROIX EN PLAINE	894
8	SCHWAB	HELENE	3 RUE DES ECOLES	68124	WINTZENHEIM	894
9	MEYER	HENRI	1 RUE DES ARTISANS	68124	WINTZENHEIM	894
10	RITZENTHALER	ARMAND	14 GRAND RUE	68140	MUNSTER	894
11	KUENTZ	LUCIEN	11 RUE DU GRAND JARDIN	68040	INGERSHEIM	894
12	HUSSER	PAUL	15 ROUTE DU VIN	68040	INGERSHEIM	897
13	RITTER	JEANNINE	52 RUE DE LA SEMM	68140	MUNSTER	897
14	BIENTZ	JEANNE	6 ROUTE DU VIN	68000	COLMAR	892
15	BERRET	JEAN PAUL	6 RUE STE BARBE	68920	WINTZENHEIM	892
16	OSWALD	HENRI	GRAND RUE	68000	COLMAR	892
17	BODIN	EUGENE	57 GRAND RUE	68250	GUNDOLSHEIM	892
18	CHAFANEL	MONIQUE	4 RUE FECHT	68250	ROUFFACH	892
19	HAESS	MADELEINE	17 RUE DE RALE	68280	ANDOLSHEIM	892
20	CHABOD	GABRIELLE	4 RUE FOCH	68000	COLMAR	892
21	BRESCH	MADELEINE	5 RUE D'ANJOU	68140	MUNSTER	892
22	CLAUDEPIERRE	HELENE	4 RUE DE COLMAR	68124	WINTZENHEIM	892
23	PFEIFFER	LOUIS	DES ROSES	68127	STE CROIX EN PLAINE	892
24	CLAUDEL	YVONNE	4 RUE DES ARTISANS	68124	WINTZENHEIM	892
25	WITTMANN	JEAN	18 RUE ROBERT SCHUMANN	68127	STE CROIX EN PLAINE	892

Il est ici facile de tester directement des instructions SQL.



NUM	NOM	PRENOM	ADRESSE	CODEP	VILLE
64	ABREDER	EUGENIE	RUE DU SORBIER	68000	COLMAR
66	ANSEL	JEAN	PRINCIPALE	68920	WINTZENHEIM
39	BAEREL	MARGUERITE	PORTE DE RIBEASVILLE	68920	WINTZENHEIM
68	BAEREL	RENE	9 RUE VIEUX MOULIN	68920	WINTZENHEIM
63	BAESA	HENRI	9 AV JEAN DE LATTRE	68920	WINTZENHEIM
67	BANNWART	GERMAINE	8 RUE LAUCH	68920	WINTZENHEIM
62	BARB	FERNAND	8 RUE DE L'EUROPE	68920	WINTZENHEIM
69	BEDEZ	SUZANNE	77 RUE DE LA REPUBLIQUE	68920	WINTZENHEIM
70	BELLICAM	MARTHE	72 RUE CLEMENCEAU	68920	WINTZENHEIM
35	BENOIT	IRENE	7 CITE DES JARDINS	68920	WINTZENHEIM
15	BERRET	JEAN PAUL	6 RUE STE BARBE	68920	WINTZENHEIM
14	BIENTZ	JEANNE	6 ROUTE DU VIN	68000	COLMAR
17	BODIN	EUGENE	57 GRAND RUE	68250	GUNDOLSHEIM
65	BOESCHLIN	ROBERT	53 RUE DE L'ILL	68040	INGERSHEIM

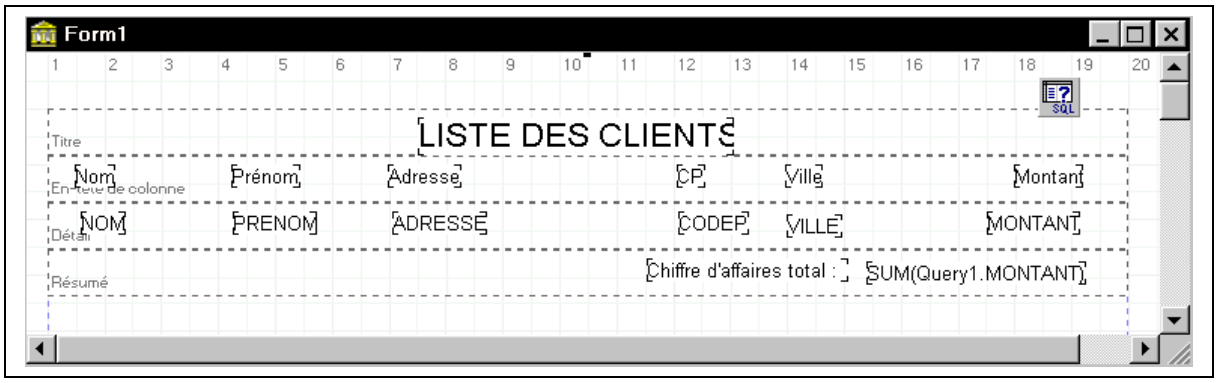
Placer ces instructions dans le composant Query1 et le rendre actif.

Placer sur la fiche un composant TQuickRep. Jouer sur le zoom pour tout voir en largeur. **Attribuer "Dataset" à "Query1"** (c'est ce que l'on oublie trop souvent).

Développer la propriété "Bands" pour ajouter des bandes: titre (title), entête de page (page header), entête de colonnes (column header), pied de page (page footer), récapitulatif (summary). Placer titre, entête de colonne, détail et récap.

Le composant QrLabel permet d'ajouter du texte, QrDbText des champs de base de données, QrExpr des calculs (en particulier pour les récaps).

Placer ces composants et affecter les propriétés nécessaires pour obtenir l'état suivant:



Le menu contextuel de Report1 nous fournit une option "prévisualiser" fort utile.

Option: présenter les montants au format monétaire (mask # ##0.00 F).

4.2 Exécution de l'état

Placer deux boutons sur la fiche, l'un pour lancer un aperçu (prewiew), l'autre pour imprimer (print). Enregistrer et exécuter. Essayer de créer une planche d'étiquettes.

4.3 Etats avec sous-totaux

L'état doit être basé sur une requête triée selon la rupture désirée. Ici nous désirons totaliser le chiffre d'affaires par ville. Pour réaliser des sous-totaux, il est nécessaire de placer dans l'état (en plus de la bande détail) un composant "TQRGroup".

On peut rajouter une bande de bas de groupe en placant une bande "TQRBand", puis en affectant la propriété "FooterBand" de "TQRGroup" à cette nouvelle bande. La rupture se fera sur le changement de valeur de la propriété "Expression" de "TQRGroup". Un assistant propose de choisir parmi des champs.

LISTE DES CLIENTS

Nom	Prénom	Adresse	CP	Ville	Montant
GRAFF	ALFRED	20 RUE MARECHAL LECLERC	68280	ANDOLSHEIM	4 600,00 F
HAEFFELE	HENRI	2 CHEMIN DE LA PEPINIERE	68280	ANDOLSHEIM	4 470,00 F
GROLLEMUND	EMILE	2 RUE DU SORBIER	68280	ANDOLSHEIM	4 470,00 F
WEREY	ALFRED	3 RUE DES AULNES	68000	ANDOLSHEIM	5 520,00 F
GOETSCH	JEANNE	20 RUE POTIERS	68280	ANDOLSHEIM	4 470,00 F
CASALINI	JOSEPH	44 RUE CLEMENCEAU	68280	ANDOLSHEIM	4 600,00 F
HICKENBICK	DENISE	16 RUE PRINCIPALE	68280	ANDOLSHEIM	4 600,00 F
HADEY	ANDRE	2 RUE DE LA WEISS	68280	ANDOLSHEIM	4 470,00 F
HEYMANN	ANNETTE	169 ROUTE DE COLMAR	68280	ANDOLSHEIM	4 470,00 F
HAESS	MADELEINE	17 RUE DE RALE	68280	ANDOLSHEIM	2 300,00 F
		Sous-total de la ville de	ANDOLSHEIM		43 970,00 F

5. RELATIONS ENTRE TABLES

Les bases de données multi-tables comportent généralement des relations. Elle peuvent être de 3 types: un à un, un à plusieurs et plusieurs à plusieurs. Dans ce dernier cas, il est nécessaire de disposer d'une table intermédiaire pour n'effectuer que des liens de 1 à plusieurs. Par exemple, pour une facturation, la relation entre clients et articles est de ce type. En effet, chaque clients peut acheter plusieurs articles et un type d'article peut être acheté par plusieurs clients. La solution est ici une table qui comporte le détail des opérations: les lignes de facture.

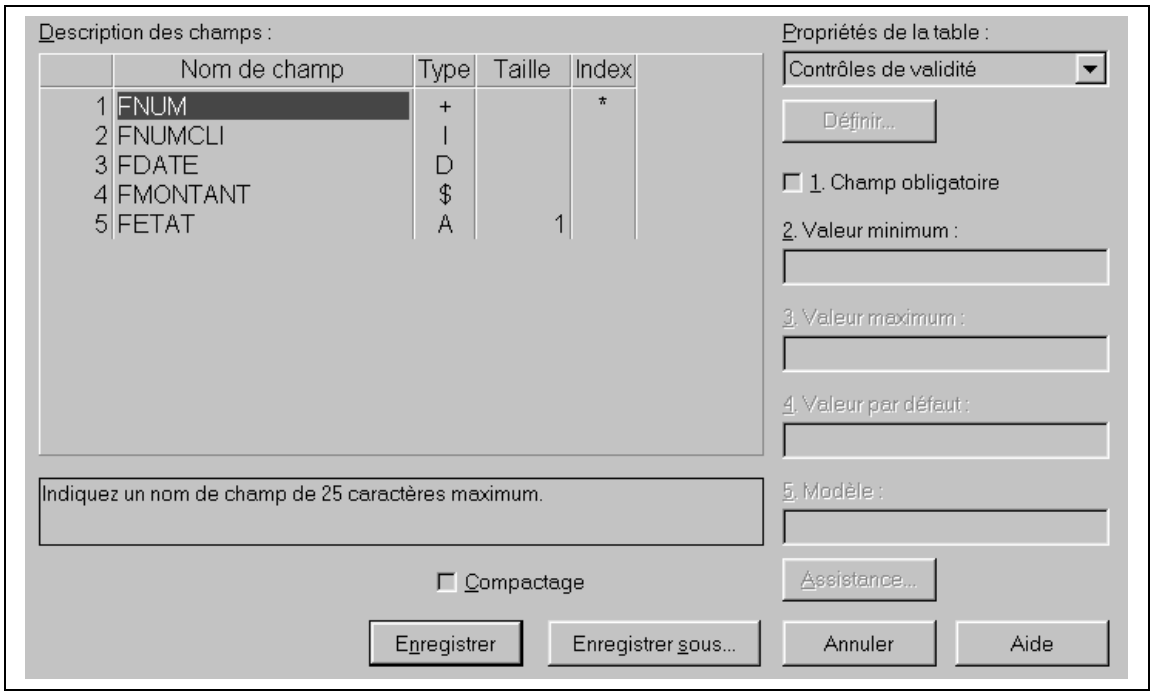
Les relations de un à un sont peu intéressantes sauf pour les ajouts optionnels de renseignements (champs). Il est souvent possible de tout regrouper dans une seule table.

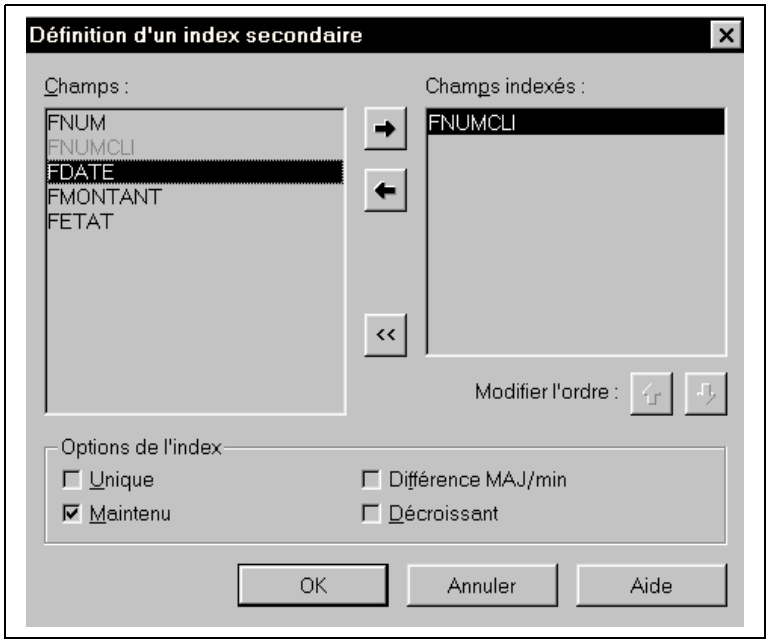
Nous verrons comment établir des relations de type un à plusieurs.

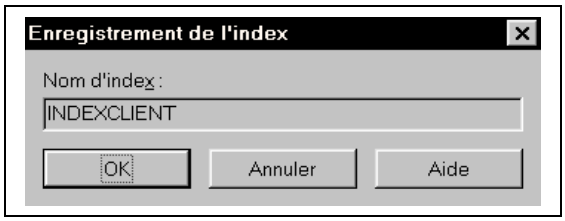
5.1 Définition des tables

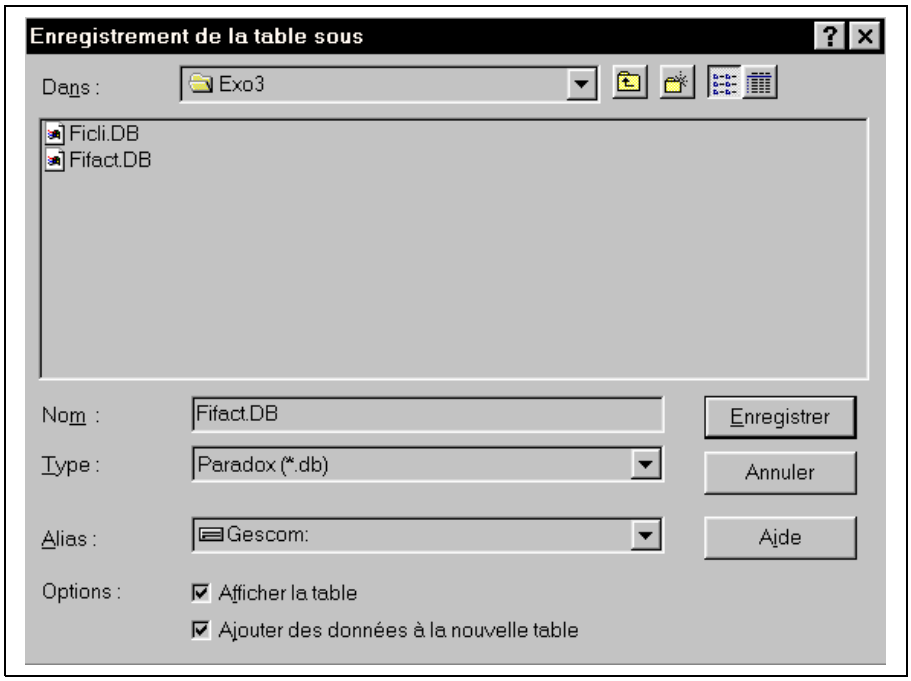
Nous avons déjà créé une table "clients", nous allons maintenant créer une table "factures" comportant le n° facture, n° client, date, montant et état (I=impayé, ...).

L'index principal sera toujours le n° de la facture, mais il faudra en plus créer un index secondaire pour le n°client (afin de pouvoir créer un lien).

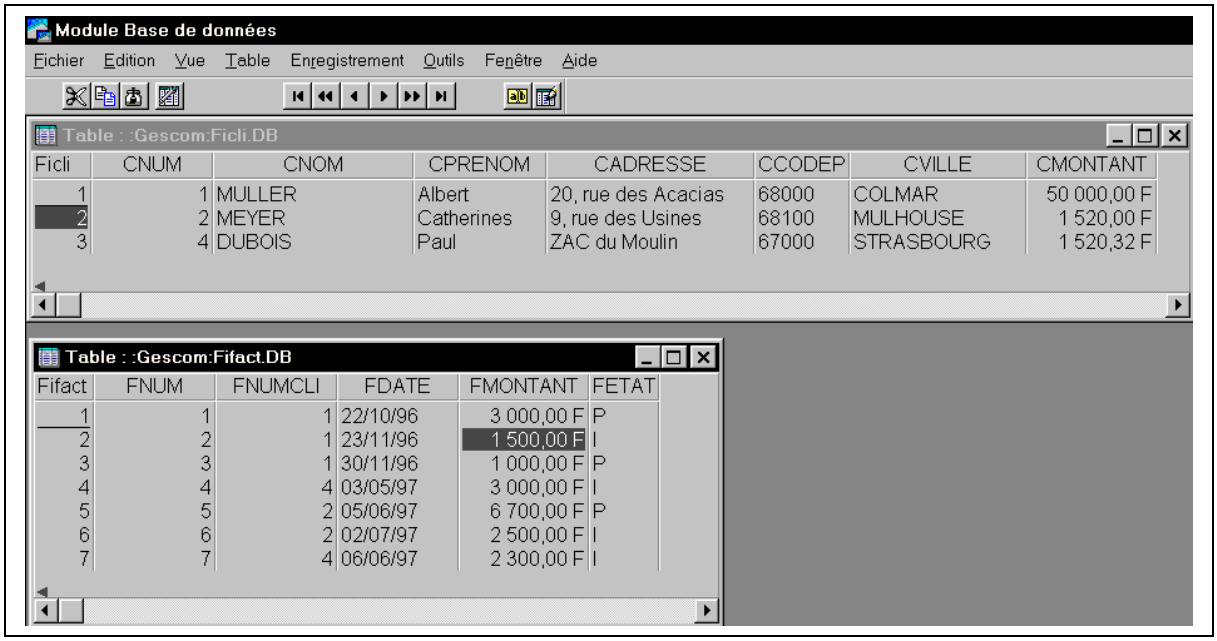






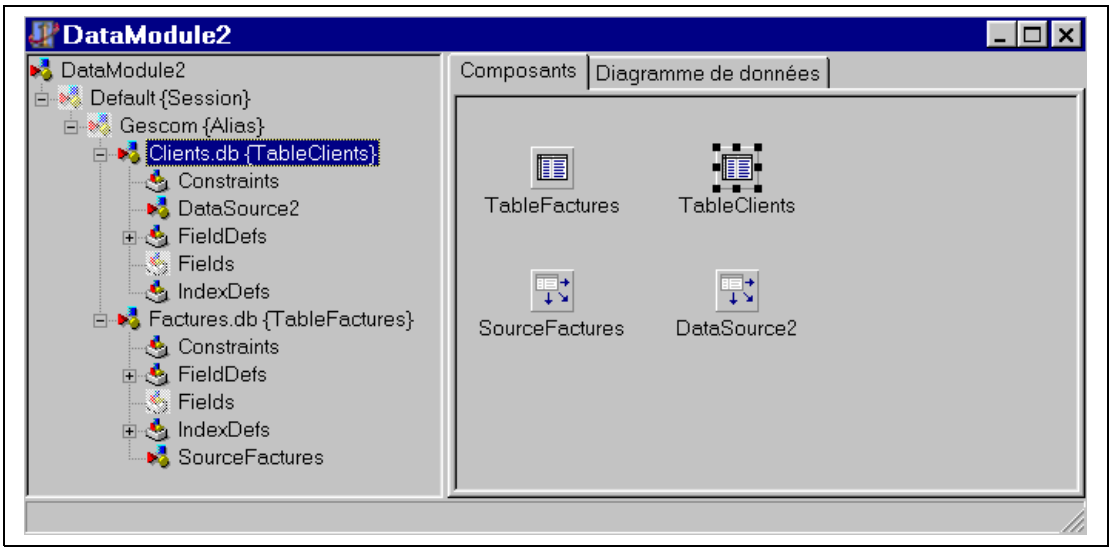


Ouvrir les 2 tables de cet alias et enregistrer manuellement quelques factures: il faut que les numéros client correspondent!!!

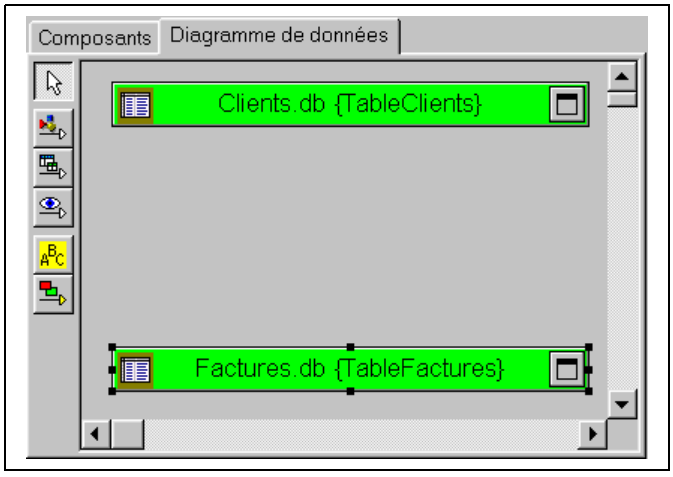


5.2 Exploitation dans un projet

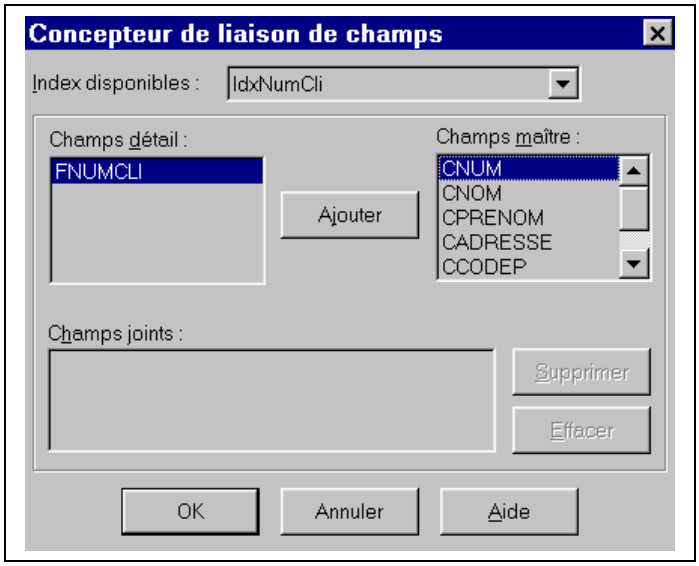
Créer un nouveau projet. Placer deux composants "Ttable", deux "DataSource" dans un module puis deux "TDbGrid" et deux "TDBnavigator" sur la fiche principale. Attribuer les propriétés pour visualiser clients et factures.



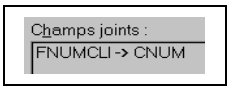
Cliquer ensuite sur l'onglet "diagramme de données" puis faire passer les 2 tables dans la partie de droite à l'aide d'un cliquer-glisser. Redimensionner si nécessaire.

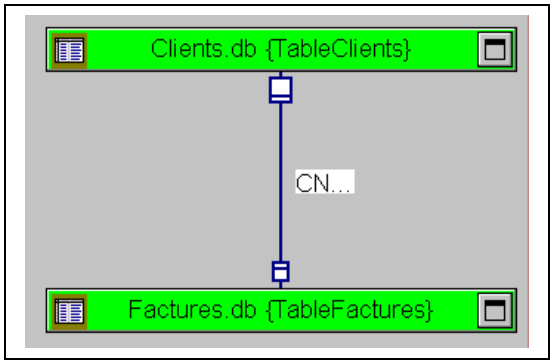


Cliquer sur le bouton "maître-détail" puis cliquer-glisser de la table maître vers la table détail. Le concepteur de liaison s'ouvre. Choisir l'index secondaire créé précédemment et cliquer sur les champs à mettre en relation.

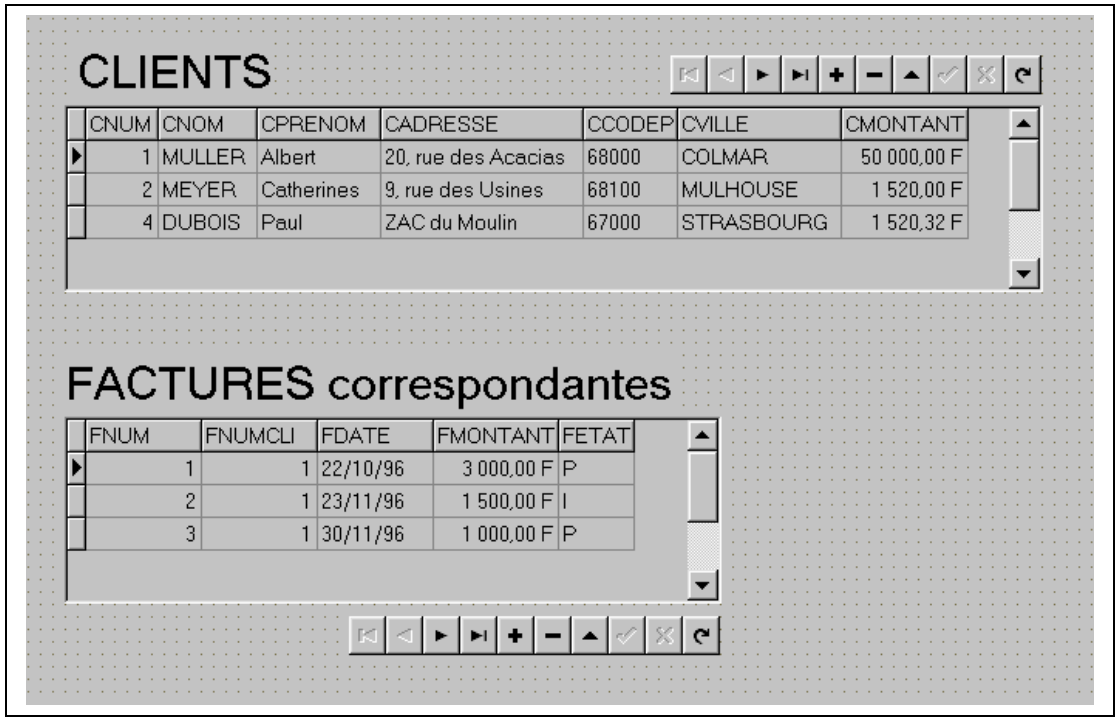


Il suffit ensuite de cliquer sur le bouton "ajouter" pour créer la relation.





Ensuite, on peut créer une fiche de ce type:



Options:

- essayer l'expert fiche de données
- totaliser les factures impayées

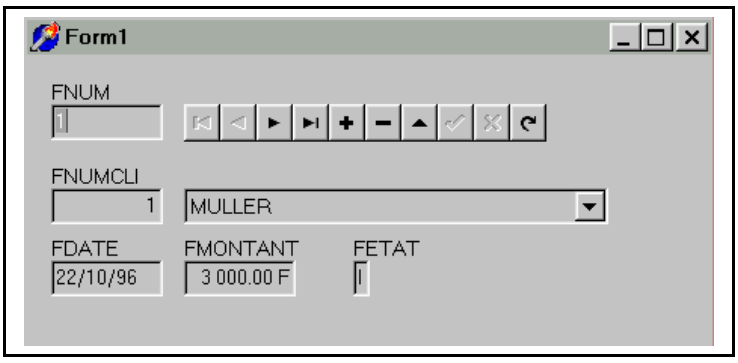
5.3 Utilisation des composants de recherche dans une autre table

Deux composant sont très utiles quand il s'agit de rechercher une valeur dans une autre table: "DBLookupCombo" et "DBLookupList".

Créer une nouveau projet. Y mettre 2 composants "Ttable" et "TDatasource" pour accéder aux factures et aux clients. Placer tous les champs "facture" et y mettre un navigateur.

Placer un composant "TDBLookupCombo" avec les propriétés suivantes:

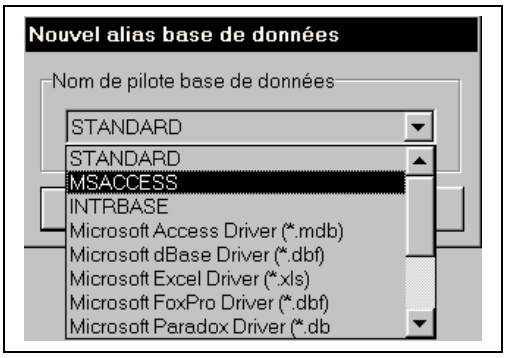
- Datasource: celle des factures,
- DataField: le n° client des factures,
- ListSource: la source des clients,
- KeyField (le champ utilisé): CNUM,
- ListField (les champs affichés): CPRENOM; CNOM.
- ListFieldIndex: 1

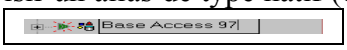


6. Utilisation de tables ACCESS 97 et 2000

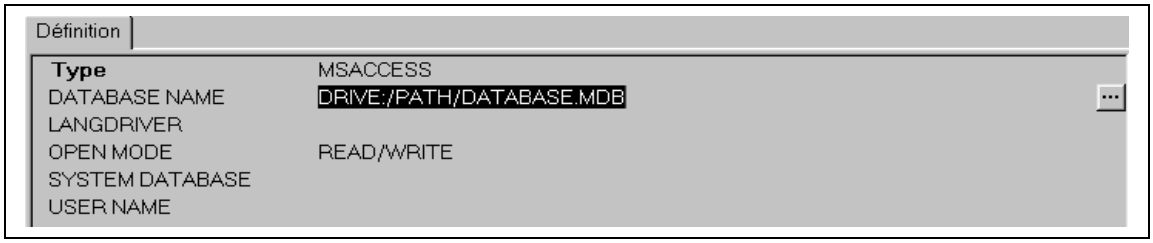
Pour pouvoir utiliser des tables crées avec Access, il suffit de définir un alias pointé sur la base (DATABASENAME).

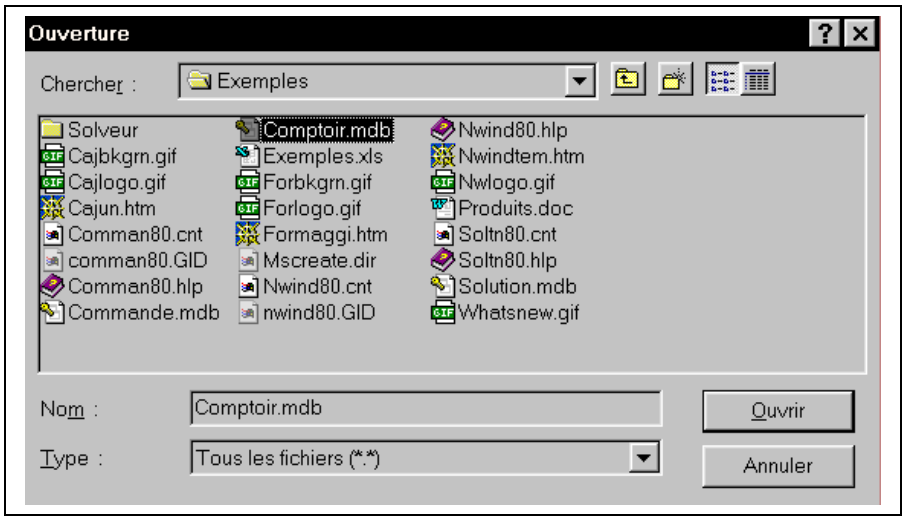
Lancer l'explorateur de bases de données et choisir Objet – Nouveau.



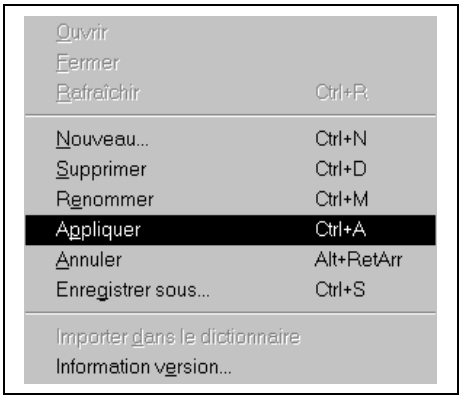
Choisir un alias de type natif (on ne passe pas par ODBC) "MSACCESS". Taper le nom de l'alias: 

Dans l'onglet définition, montrer le chemin d'accès en cliquant sur le bouton …

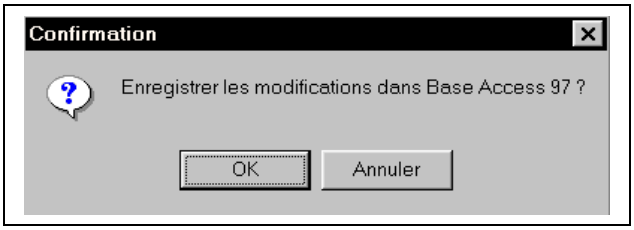




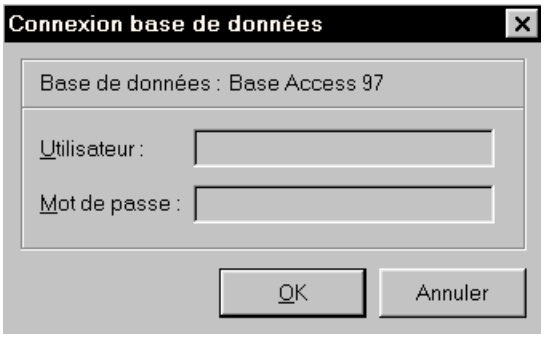
Dans le menu contextuel de l'alias que nous venons de créer, choisir appliquer:



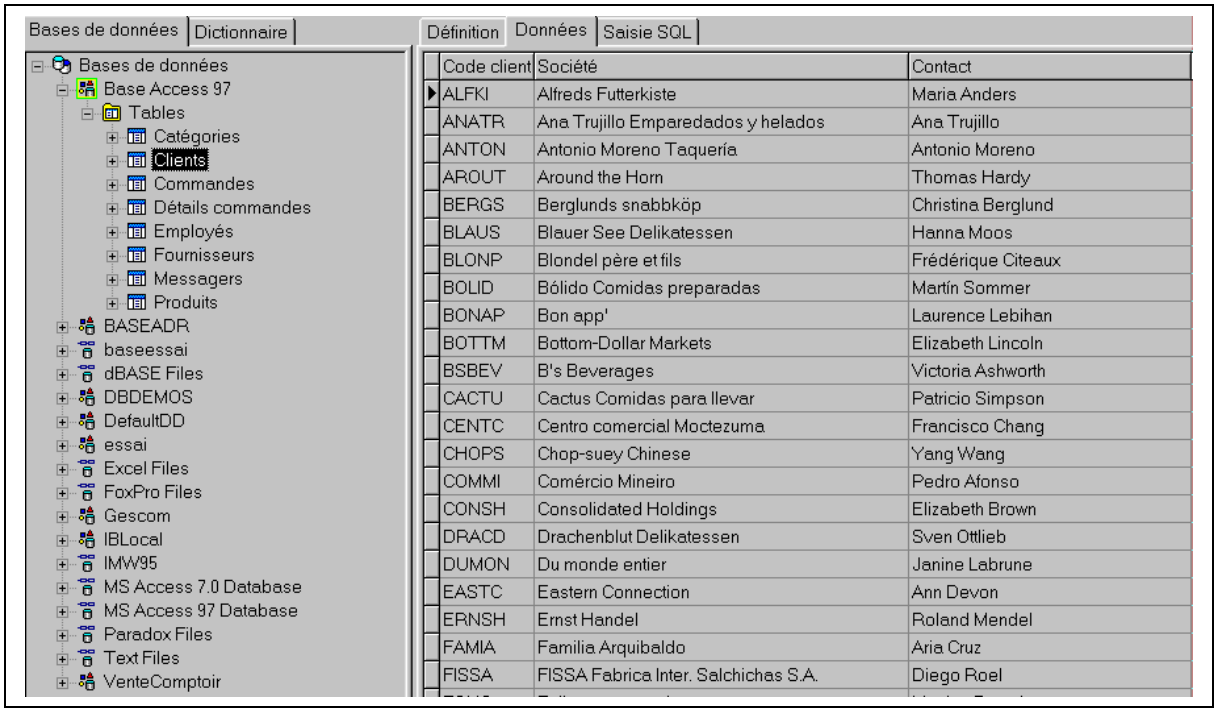
Valider:



Valider la connexion sans rien saisir si la base n'est pas protégée:



Une base ACCESS se manipule maintenant comme d'habitude, sans difficulté. Avec ODBC, il est possible de faire des alias sur tous types de bases de données dont on possède les pilotes. Par exemple, l'installation de "Client Access" génère un pilote capable d'attaquer un AS400…



Ensuite on peut accéder à toutes les tables de cette base en définissant les propriétés du composant "TTable":

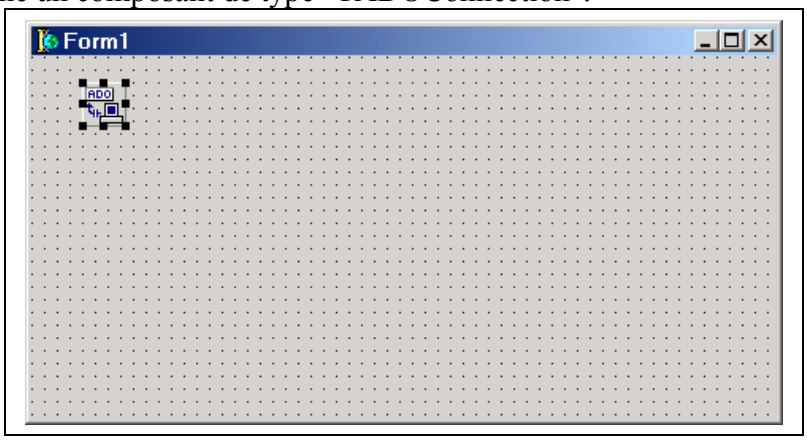
- DataBaseName: choisir le nom de l'alias dans la liste déroulante,
- TableName: choisir le nom de la table comme précédemment.

C'est très simple à cause du driver natif intégré: plus besoin d'ODBC, donc un gain de temps appréciable.

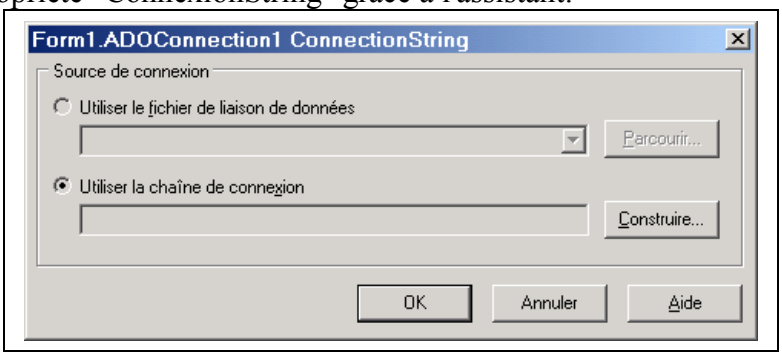
On peut faire plus fort, c'est à dire rajouter des tables dans une base "ACCESS" avec l'utilitaire BDE (et les visualiser avec l'explorateur de bases). La base doit exister et être pointée par un alias.

7. Utilisation des composants ADO (Microsoft)

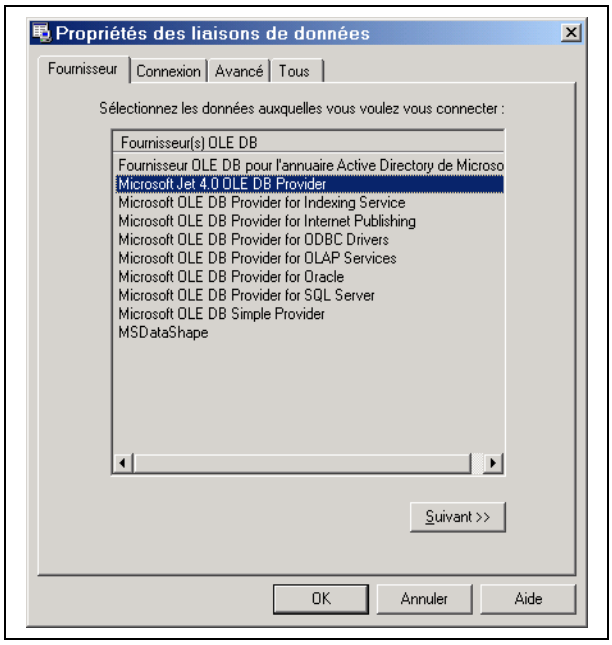
Créer un nouveau projet.
Placer sur la fiche un composant de type "TADOConnection".



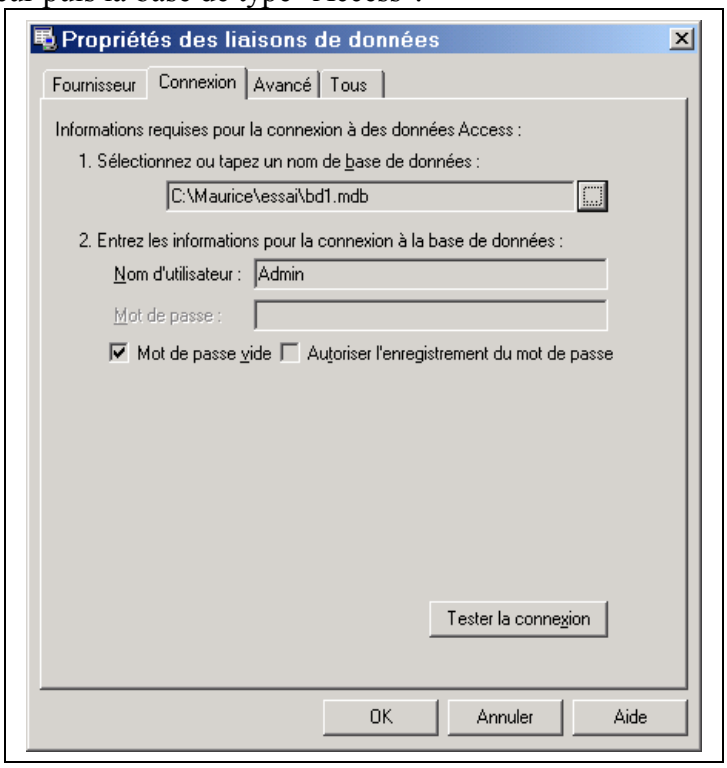
Renseigner la propriété "ConnexionString" grâce à l'assistant:



Cliquez sur "construire".



Choisir le fournisseur puis la base de type "Access".



Il est intéressant de tester ici la connexion grâce au bouton.
Valider et mettre la propriété "connected" à "true" pour la phase de développement.
Vous pouvez ensuite utiliser les composants de type TADOTable ou TADOQuery associés à un TDataSource.
C'est ensuite un jeu d'enfant de renseigner les propriétés et de faire fonctionner le projet!!!

8. PROJET: GESTION DE BIBIOTHEQUE

Réaliser un projet permettant de gérer une petite bibliothèque.

Chaque livre est porteur d'un numéro unique attribué dans l'ordre d'achat.

Chaque lecteur reçoit une carte de lecteur comportant un numéro unique. Il est possible qu'il puisse oublier sa carte.

Fonctionnalités souhaitées:

- Prêt et retour de livres
- Liste des livres prêtés plus de 15 jours et non rendus
- Liste des lecteurs
- Liste des livres (par titre, par auteur)
- Liste des livres de plus de 5 ans

9. ANNEXE: LE LANGAGE SQL

Le langage SQL est très important car il permet des requêtes sur tous types de bases de données. Il est souvent utilisé pour des applications client-serveur et permet d'aller au delà des outils visuels habituels (de type Query par exemple).

9.1 SELECT

L'instruction SELECT est utilisée pour retrouver des données à partir d'une ou de plusieurs tables. Une instruction SELECT qui retrouve des données depuis plusieurs tables est appelée une "jointure". SQL local supporte la forme suivante de l'instruction SELECT.

```
SELECT [DISTINCT] liste_colonnes
        FROM référence_table
        [WHERE condition_recherche]
        [ORDER BY liste_tri]
        [GROUP BY liste_groupe]
        [HAVING condition_having]
        [UNION expr_select]
```

Toutes les clauses sont gérées comme dans la norme ANSI de SQL, à l'exception de ce qui est indiqué ci-après. Les clauses entre crochets sont facultatives.

La liste_de_colonnes indique les colonnes dans lesquelles les données doivent être recherchées. Ainsi, l'instruction suivante recherche les données dans deux colonnes :

```
SELECT PART_NO, PART_NAME
        FROM PARTS
```

9.1.1 FROM

La clause FROM spécifie la ou les tables dans lesquelles les données doivent être recherchées. référence_table peut être une table unique, une liste de tables séparées par des virgules ou encore une jointure interne ou externe comme cela est spécifié dans la norme SQL-92. L'exemple suivant spécifie une table unique :

```
SELECT PART_NO
        FROM "PARTS.DBF"
```

L'instruction suivante spécifie pour table une jointure externe gauche :

```
SELECT * FROM PARTS LEFT OUTER JOIN INVENTORY
        ON PARTS.PART_NO = INVENTORY.PART_NO
```

9.1.2 WHERE

La clause facultative WHERE réduit le nombre de lignes renvoyées par une requête uniquement à celles qui correspondent au critère spécifié dans condition_de_recherche. L'instruction suivante, par exemple, ne retrouve que les lignes pour lesquelles ART_NO est supérieur à 543 :

```
SELECT * FROM PARTS
    WHERE PART_NO > 543
```

La clause WHERE peut contenir le prédicat IN, suivi d'une liste de valeurs entre parenthèses. Ainsi, l'instruction suivante ne retrouve que les lignes dans lesquelles le numéro d'article fait partie des éléments de la liste qui suit IN :

```
SELECT * FROM PARTS
    WHERE PART_NO IN (543, 544, 546, 547)
```

Outre les opérateurs de comparaison scalaires (=, <, > ...), d'autres prédicats utilisant IN, ANY, ALL, EXISTS sont supportés.

9.1.3 ORDER BY

La clause ORDER BY spécifie l'ordre des lignes. Par exemple, la requête suivante ramène une liste de tous les articles par ordre alphabétique des noms d'article :

```
SELECT * FROM PARTS
    ORDER BY PART_NAME ASC
```

La requête suivante présente toutes les informations sur les articles, classées en ordre numérique décroissant des numéros d'article :

```
SELECT * FROM PARTS
    ORDER BY PART_NO DESC
```

Les champs calculés peuvent être classés par nom de corrélation ou par position numérique. Par exemple, la requête suivante ordonne les lignes selon le champ calculé NOM_COMPLET :

```
SELECT NOM || ', ' || PRENOM AS NOM_COMPLET, TELEPHONE,
    FROM CLIENTS
    ORDER BY NOM_COMPLET
```

La projection de toutes les colonnes de groupement ou de tri n'est pas obligatoire .

9.1.4 GROUP BY

La clause GROUP BY spécifie la façon dont les lignes récupérées sont regroupées vis à vis des fonctions statistiques.

9.1.5 HAVING

La clause HAVING spécifie des conditions que les enregistrements doivent satisfaire pour être renvoyés par la requête. C'est une expression conditionnelle utilisée avec la clause GROUP BY. Les groupes qui ne correspondent pas à l'expression de la clause HAVING sont omis de l'ensemble des résultats.

Les sous-requêtes sont supportées dans la clause HAVING. Une sous-requête fonctionne comme une condition de recherche limitant le nombre de lignes renvoyées par la requête parent. Voir la Clause WHERE

Outre les opérateurs de comparaison scalaires (=, <, > ...), d'autres prédicats utilisant IN, ANY, ALL, EXISTS sont supportés.

9.1.6 UNION

La clause UNION combine le résultat de plusieurs instructions SELECT pour obtenir une seule table.

9.2 JOINTURES HETEROGENES

SQL local supporte les jointures de tables de formats différents de bases de données. C'est ce qu'on appelle une "jointure hétérogène".

L'instruction suivante récupère les données d'une table Paradox et d'une table dBASE :

```
SELECT DISTINCT C.CUST_NO, C.STATE, O.ORDER_NO
      FROM "CUSTOMER.DB" C, "ORDER.DBF" O
      WHERE C.CUST_NO = O.CUST_NO
```

Vous pouvez également utiliser des alias de moteur de base de données Borland en conjonction avec des noms de tables (ex: *SELECT * FROM :PDOX:TABLE1*).

9.3 INSERT

Cette instruction permet d'insérer des enregistrements, par exemple:

```
INSERT INTO CUSTOMER (FIRST_NAME, LAST_NAME, PHONE)
    VALUES(:fnom, :lnom, :tel_no)
```

L'insertion depuis une table dans une autre à partir d'une sous-requête n'est pas autorisée.

L'instruction suivante ajoute une ligne à une table, en assignant des valeurs à deux colonnes :

```
INSERT INTO EMPLOYEE_PROJECT (EMP_NO, PROJ_ID) VALUES (52, "DGPII");
```

L'instruction suivante spécifie des valeurs à insérer dans une table avec une instruction SELECT :

```
INSERT INTO PROJECTS
    SELECT * FROM NEW_PROJECTS
        WHERE NEW_PROJECTS.START_DATE > "6-JUN-1994";
```

9.4 UPDATE

Modification d'enregistrements

9.5 DELETE

Destruction d'enregistrements

9.6 REQUETES PARAMETREES

Dans les instructions de manipulation des données, on peut utiliser des variables ou marqueurs de paramètre (?) à la place des valeurs. Les variables doivent toujours être précédées du signe deux-points (:), comme dans cet exemple :

```
SELECT NOM, PRENOM
    FROM "CLIENT.DB"
    WHERE NOM > :var1 AND PRENOM < :var2
```

9.7 FONCTIONS RECAPITULATIVES

Les fonctions récapitulatives suivantes de la norme ANSI de SQL sont disponibles dans la recherche de données avec SQL local :

- SUM(), pour totaliser toutes les valeurs numériques d'une colonne
- AVG(), pour faire la moyenne de toutes les valeurs numériques différentes de NULL dans une colonne
- MIN(), pour déterminer la valeur minimale d'une colonne
- MAX(), pour déterminer la valeur maximale d'une colonne
- COUNT(), pour compter le nombre de valeurs d'une colonne satisfaisant un critère donné

Les expressions récapitulatives complexes sont supportées, par exemple :

- SUM(Champ * 10)
- SUM(Champ) * 10
- SUM(Champ1 + Champ2)

9.8 CHAINES

SQL local supporte les fonctions de manipulation de chaînes de la norme ANSI de SQL pour la recherche, l'insertion et la mise à jour des données :

- UPPER(), pour convertir une chaîne en majuscules
- LOWER(), pour convertir une chaîne en minuscules
- TRIM(), pour supprimer des répétitions d'un caractère spécifié à gauche, à droite, ou aux deux extrémités d'une chaîne
- SUBSTRING(), pour créer une sous-chaîne à partir d'une chaîne

9.9 DATE

SQL local supporte la fonction EXTRACT() pour isoler un champ numérique simple d'un champ date/heure, en utilisant la syntaxe suivante :

```
EXTRACT (champ_extrait FROM champ_nom)
```

Ainsi, l'instruction suivante extrait la valeur de l'année d'un champ date :

```
SELECT EXTRACT(YEAR FROM HIRE_DATE)
    FROM EMPLOYEE
```

Vous pouvez extraire de la même façon le mois (MONTH), le jour (DAY), l'heure (HOUR), les minutes (MINUTE) et les secondes (SECOND).

Note: EXTRACT ne supporte pas les clauses TIMEZONE_HOUR ni TIMEZONE_MINUTE.

9.10 OPERATEURS

SQL local supporte les opérateurs suivants :

TYPE	OPERATEUR
Arithmétique	+ - * /
Comparaison	< > = <> >= =< IS NULL IS NOTNULL
Logique	AND OR NOT
Concaténation de chaîne	||

9.11 MODIFIABLES

SQL Links offre un support étendu pour les requêtes modifiables (ou reliées) monotables et multitables.

Ces restrictions s'appliquent aux mises à jour :
- Les champs de la liaison ne peuvent être mis à jour
- Changer d'index provoque une erreur.

Restrictions sur les requêtes reliées:

La sémantique des requêtes reliées, pour toutes les méthodes de manipulation des données, renvoie des curseurs semblables par leurs fonctions et leur sémantique aux curseurs renvoyés par la fonction DbiOpenTable de BDE.

Les vues ou les requêtes monotables sont modifiables aux conditions suivantes :

- Pas d'opération JOIN, UNION, INTERSECT ni MINUS.
- Pas de mot réservé DISTINCT dans l'instruction SELECT. (Cette restriction peut être levée si tous les champs d'un index unique sont projetés.)
- Tout élément d'une clause SELECT est une référence de colonne simple ou un champ calculé (opérateur récapitulatif non autorisé).

La table référencée dans la clause FROM est soit une table de base de données modifiable, soit une vue modifiable:
- Pas de clause GROUP BY ni HAVING.
- Pas de sous-requêtes référençant la table dans la clause FROM et pas de sous-requêtes corrélées.
- Une clause ORDER BY peut être satisfaite par un index.

Restrictions sur les jointures reliées:

Les jointures reliées dépendent des curseurs composites. Les jointures reliées peuvent seulement être utilisées si :
- Toutes les jointures sont des jointures externes gauche-droite.
- Toutes les jointures sont des équijointures.
- Toutes les conditions de jointure peuvent être satisfaites par des index (pour Paradox et dBASE).
- L'ordre des sorties n'est pas défini.
- Chaque table de la jointure est une table de base de données.
- La requête ne contient aucun des éléments cités ci-dessus qui empêcherait la possibilité de mise à jour monotable.

9.12 EXEMPLES

Exemple 1 : UPDATE

```
update marchand
        set ville = 'Santa Cruz'
        where marchand.ville = 'Scotts Valley'
```

Exemple 2 : INSERT

```
insert
        into marchand ( no_art, ville )
        values ( 'aa0094', 'San Jose' )
```

Exemple 3 : DELETE

```
delete
        from marchand
        where no_art = 'aa0093'
```

Exemple 4 : SELECT utilisée pour une jointure

L'exemple suivant illustre comment l'instruction SELECT est supportée comme équivalence à JOIN :

```
select distinct a.no_art, m.quant, m.ville
        from arts a, marchand m
        where a.no_art = m.no-art
        and a.quant > 20
        order by a.quant, m.ville, a.no_art
```

Une instruction SELECT qui contient une jointure doit posséder une clause WHERE dans laquelle au moins un champ de chaque table est impliqué dans un contrôle d'égalité.

Exemple 5 : sous-sélections

Les requêtes de sous-sélection sont supportées. L'exemple suivant illustre cette syntaxe :

```
select a.no_art
        from arts a
        where a.quant in
                (select i.quant
                        from inventaire i
                        where  i.part_no = 'aa9393')
```

Exemple 6 : GROUP BY

Les exemples suivants illustrent la clause GROUP BY :

```
select no_art, sum(quant) as PQTE
        from arts
        group by no_art
```

Note: Les champs récapitulatifs de la clause SELECT doivent disposer d'une clause GROUP BY si un champ projeté est utilisé, comme l'illustre le premier exemple ci-dessus.

Exemple 7 : ORDER BY

L'exemple suivant illustre ORDER BY avec une clause DESCENDING :

```
select distinct no_client
        from c:\donnees\clients
        order by no_client descending
```

9.13 OPERATIONS SUR TABLES (PARADOX et DBASE)

9.13.1 ALTER TABLE

Cette instruction permet d'ajouter et d'enlever des champs à la table.

Exemples:

ajout d'un champ:
ALTER TABLE "employes.dbf" ADD BUILDING_NO SMALLINT

destruction de champs:
ALTER TABLE "employes.dbf" DROP NOM, DROP PRENOM

combinaison:
ALTER TABLE "employes.dbf" DROP NOM, DROP PRENOM, ADD NOM_COMPLET CHAR[30]

9.13.2 DROP TABLE

Suppression de table.
Exemple: *DROP TABLE "employes.db"*

9.13.3 CREATE INDEX

Création d'index (secondaire sur PARADOX)
Exemple: *CREATE INDEX NOMINDEX ON "employes.dbf" (NOM)*

9.13.4 DROP INDEX

Destruction d'index
Exemples:

- *DROP INDEX "employes.db" .PRIMARY*
- *DROP INDEX "employes.db" .INDEXSEC*

TABLE DES MATIERES