V.2.2. Création des tables

V.2.2.1. Principe

Nous allons créer une table contenant pour chaque formation :

- Un code, par exemple 3 ;
- Un Nom, par exemple « Delphi Interbase » ;
- Un nombre de jours par exemple 3.
- Un prix par exemple 15.000,00 DA.

Pour cela nous devons envoyer une requête de création en langage SQL vers le serveur Interbase, la syntaxe est la suivante :

```
CREATE TABLE formations
(f-numero INTEGER, f-nom CHARACTER (30),
f-jours INTEGER, f-Prix NUMERIC (5,2)) ;
```

Il suffit donc de choisir un nom de la table, et le nom de chaque colonne avec son type. Parmi les types autorisés par Interbase :

- **INTEGER** pour les valeurs entières 32 bits.
- **SMALLINT** pour les valeurs entières 16 bits.
- **Numeric** (décimales, précision) pour une valeur numérique Flottante.
- **Date** pour une date.
- **CHARACTER** (taille) pour des caractères.

Pour envoyer cette requête vers le serveur :

- Nous utilisons un TIBDatabase qui assura la connexion vers le serveur.
- Nous utilisons un TIBQuery.
- Nous le relions à TIBDatabase.
- Nous plaçons la requête SQL dans sa propriété TIBQuery.SQL via l'inspecteur d'objets ou par code.
- Nous exécutons la requête par TIBQuery.ExecSql.

Remarque :

La création n'est visible par tout le monde que si la transaction qui est utilisée pour la création de la table est confirmée.

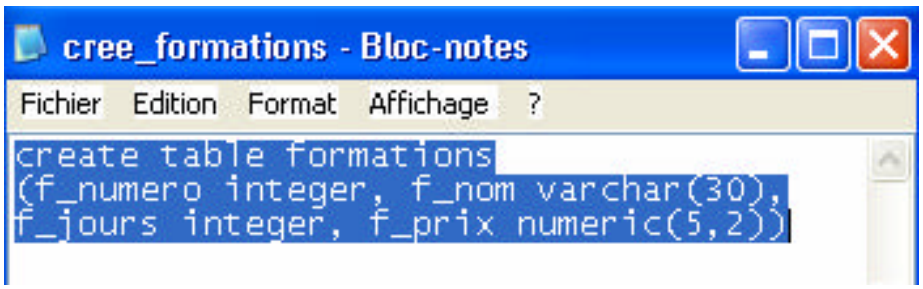
V.2.2.2. Utilisation de SQL

La requête à envoyer au serveur est placée dans `TIBQuery.Sql`.

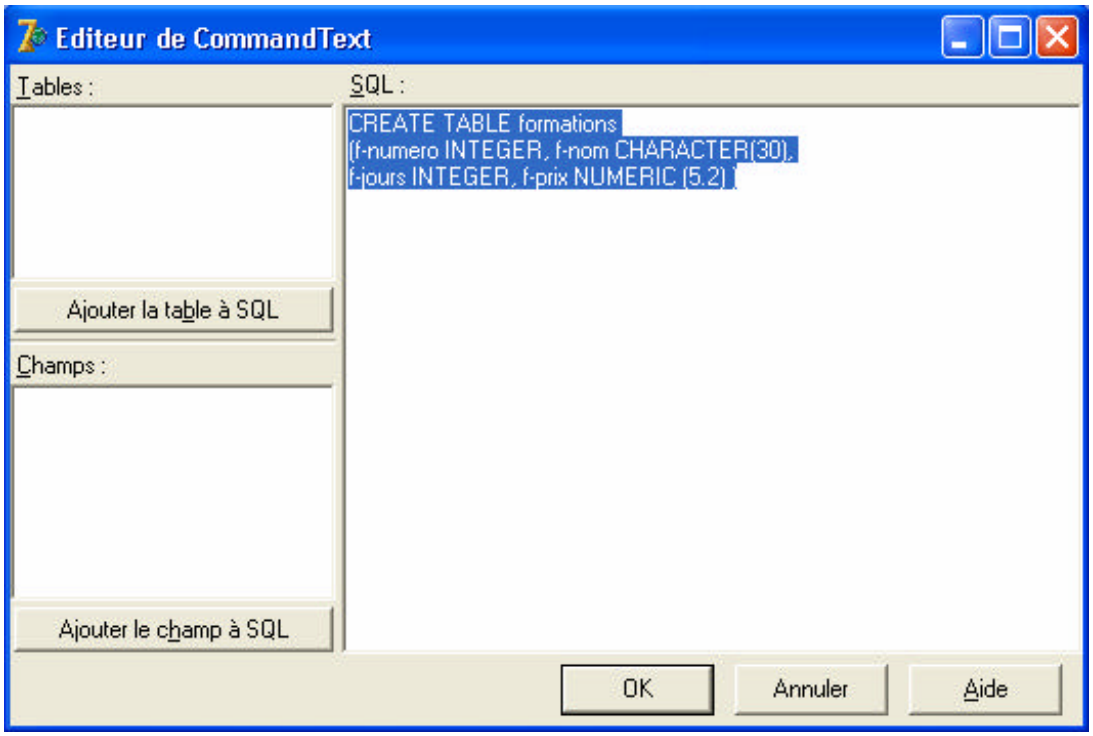
```
IBQuery1.SQL.Clear ;
IBQuery1.SQL.ADD( ' CREATE TABLE formations
    (f-numero INTEGER, f-nom CHARACTER (30),
 f-jours INTEGER, f-Prix NUMERIC (5,2)) ')
```

Pour construire la requête, on peut utiliser `Add`, ou même `LoadFromFile` pour lire un fichier texte (`.txt`) contenant la requête :

```
IbQuery1.Sql.LoadFromFile ('Cree-formations.txt') ;
```



Nous pouvons entrer la requête en utilisant l'inspecteur d'objets.



La mise en page n'a aucune importance pour le serveur ; la requête peut être répartie en plusieurs Lignes :

```
IbQuery1.Sql.Add('CREATE TABLE') ;
IbQuery1.Sql.Add(' formations') ;
IbQuery1.Sql.Add('(f-numero INTEGER, f-nom CHARACTER (30),
f-jours INTEGER, f-Prix NUMERIC (5,2))') ;
```

Et vous pouvez taper manuellement la requête dans la partie « SQL ».

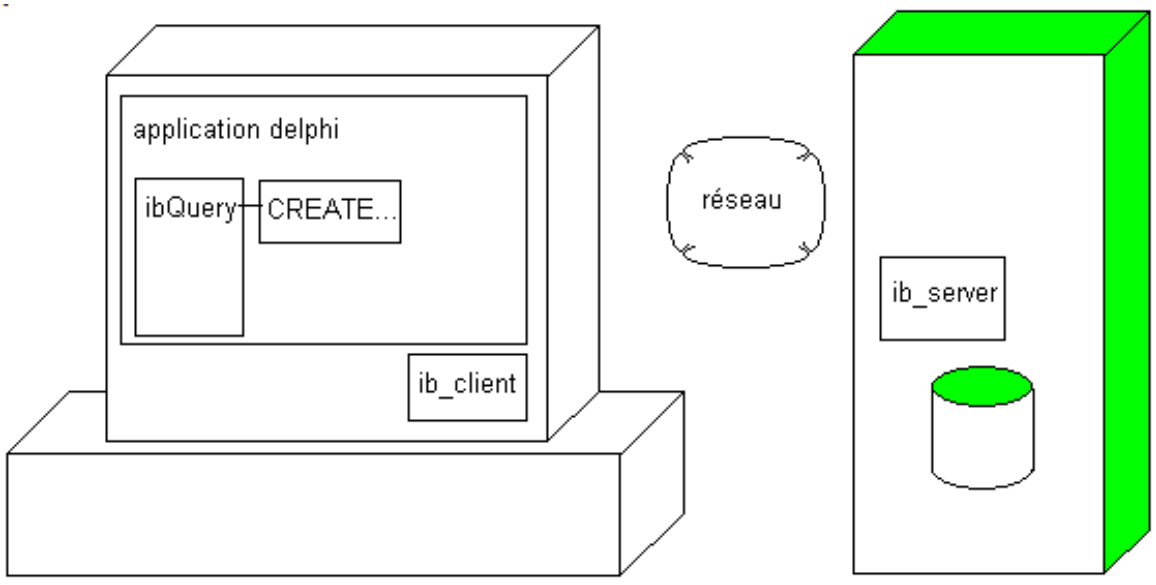
Nous pouvons aussi créer la `String` en plusieurs étapes par toutes les primitives de `String` telles que la concaténation, `Insert`, etc.…

```
Ma-requête : = 'CREATETABLE ' + Edit1.Text + ' (' ;
Ma-requête : = Ma-requête + Edit2.Text + ' )' ;
IBQuery1.SQL.Add(Ma-Requête) ;
```

V.2.2.3. Comment ça marche

Au niveau fonctionnement :

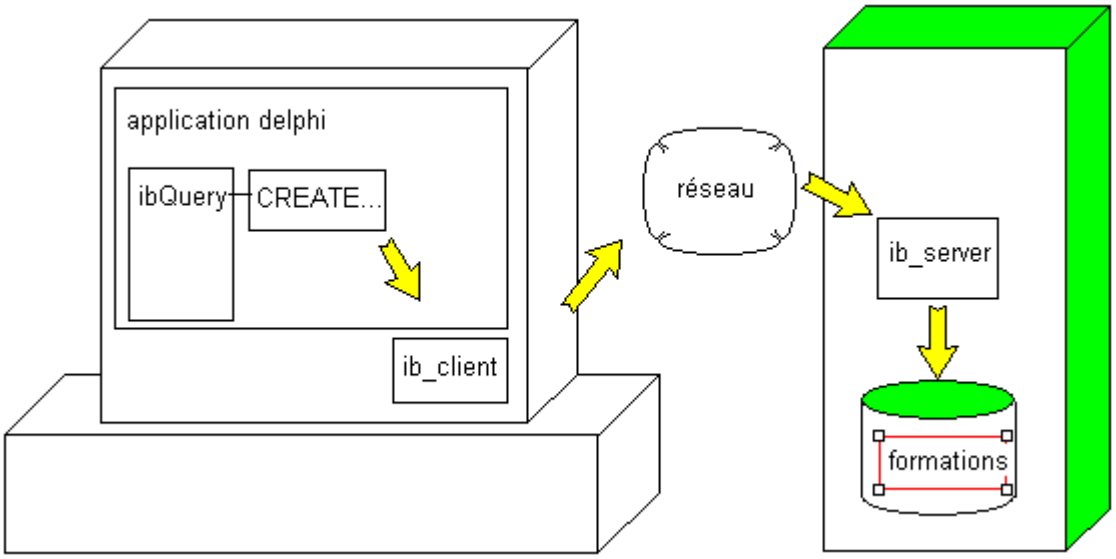
Lorsque nous construisons la requête par `IbQuery1.Sql.Add`, ce texte est à ce stade une `TString` en mémoire.



A ce niveau aucune vérification de syntaxe n'est effectuée. Lorsque nous exécutons :

`IbQuery1.ExecSql;` alors :

- La requête est envoyée au serveur via le client,
- Le serveur traite la requête et crée la table si celle ci est correcte; Retourne une erreur transformée par Delphi en exception en cas de problème.



Remarque :

L'envoi de toute requête qui modifie des données du serveur (la création par exemple) ne peut se faire que par code (.Pas en basculent IbQuery1.Active sur True en mode conception).

V.2.3. Suppression d'une base de données

Pour supprimer une base de données, il suffit de spécifier son nom dans la propriété DatabaseName, et après vérification de sa Non ouverture par personne, appelez la méthode DropDataBase.

V.3. TIBTransaction, gestion des transactions

En Interbase toutes les requêtes sont crées dans le cadre d'une transaction. Les transactions sont un mécanisme qui garantit que plusieurs requêtes sont réalisées en entier ou annulées. Un exemple simple est le transfert Bancaire, si nous débitons Khaled pour créditer cherif, le système doit garantir que soit les deux modifications sont réalisées on aucune ne l'est.

En Delphi, les composants de bases de données utilisent par défaut des transactions transparentes pour le programmeur, cependant on peut gérer les transactions d'une manière explicite, et c'est ce qui est recommandé pour Interbase.

En interbase, on utilise un composant TIBTransaction qu'on connecte à TIBDatabase.DefaultTransaction.

Propriétés du composant TIBTransaction	
Propriété	**Description**
DefaultDatabase	Pointe vers le composant TIBDatabase, dont il est la transaction par défaut. Cette information semble faire doublon avec la propriété DefaultTransaction du TIBDatabase mais il est en effet nécessaire de les renseigner toutes les deux.
IdleTimer	Il est possible d'automatiser le Rollback ou le commit après un certain temps d'activité. Pour cela indiquez un temps en seconde différent de 0 et mentionnez l'action à remplir dans DefaultAction.
DefaultAction	Permet d'indiquer l'action qui sera exécutée par le composant lorsque le temps d'inactivité programmé par IdleTimer expirera, les possibilités offertes sont le commit et le Rollback.
Params	Permet de paramétrer finement la gestion de la transaction. On utilise généralement l'éditeur de propriété qui met à notre disposition un dialogue simplifiant la saisie.

Les paramètres de la propriété Params d'une transaction permettent de fixer le niveau d'isolation entre elle et les autres transactions.

Niveaux d'isolation d'une transaction	
Niveau d'Isolation	**Description**
SnapShot (par défaut) (C'est le mode le plus standard)	Il offre à l'application une photo Instantanée de la base de données telle qu'elle est au moment de l'ouverture de la transaction. Les autres transactions peuvent faire des insertions, des suppressions et être validées, l'application verra la base comme si aucune activité n'avait eu lieu depuis le lancement de la transaction. C'est grâce à ce système qu'Interbase peut autoriser des sauvegardes consistantes même lorsque des utilisateurs sont connectés à la base et la modifient.
Read Commited	La transaction verra toutes les modifications apportées par les autres transactions dés quelles seront validées.

Niveaux d'isolation d'une transaction (Suite)	
Read Only Table Stability Read Write Table Stability	Les autres transactions ne peuvent pas modifier les tables que votre transaction a lues ou changées (Read ou Read – Write) tant que celle-ci n'aura pas pris fin (commit ou Rollback). Ce mode est rarement utilisé puisqu'il est bloquant et pénalise l'ensemble des autres transactions.

Le paramètre par défaut est le niveau le plus standard à savoir SnapShot, en mode sans attente (nowait). Face à une situation de concurrence, la transaction peut soit attendre que la ou les transactions qui lui bloquent l'accès à une ressource se terminent (mode wait qui est presque jamais utilisé), soit s'arrêter immédiatement et retourner une erreur (mode nowait c'est le mode qui est presque toujours utilisé).

Les primitives utilisées avec TIBTransaction :

- TIBTransaction.StartTransaction pour démarrer une nouvelle transaction avant une ou plusieurs opérations.
- TIBTransaction.commit pour confirmer la suite d'opérations.
- TIBTransaction.RollBack pour tout Annuler.

Remarque :

StartTransaction et commit ou RollBack sont souvent utilisées dans TRY..EXCEPT ;

```
TRY
      IBTransaction1.StartTransaction ;
      Débité Khaled ;
      Crédité Cherif ;
      IBTransaction1.commit ;
EXCEPT
      IBTransaction1.Rollback ;
END ;
```

On ne peut appeler IBTransaction1.StartTransaction si la transaction attachée à IBTransaction1 est fermée (commited ou RollBack).

Pour savoir si une transaction est active, on utilise la fonction InTransaction :

```
IF NOT IBTransaction1.InTransaction Then
                  IBTransaction1.StartTransaction ;
```

Si nous réalisons le traitement sans cette IBTransaction1 :

- Pendant l'exécution de l'application d'autres applications (l'explorateur de base de données, un autre exécutable, etc.…) ne verraient pas la table.
- Un autre IBQuery de notre propre application pourrait ne pas voir la table.
- Lorsque l'exécutable sera fermé, la transaction sera automatiquement fermée.

V.3.1. La logique des transactions

Les transactions sous Interbase sont obligatoires. C'est à dire qu'un simple SELECT pour retourner quelques enregistrements qui seront juste lus doit être compris dans une transaction. Si une transaction n'est pas débutée explicitement par le programmeur, Interbase le fera. Si cela est parfais dans la plupart des cas pour un SELECT, il n'en va pas de même s'il s'agit d'insertions ou d'autres manipulations modifiant la base de données. Si les modifications concernent plusieurs enregistrements ou plusieurs tables il est préférables d'entourer tout le bloc par une seule et même transaction pour en assurer la consistance.

Pour Interbase une transaction porte un numéro séquentiel qui lui est attribué au moment de son activation. La plus récente des transaction portant le numéro le plus élevée. Interbase effectue une copie des pages d'inventaire de transaction (liste de toutes les transactions en cours avec leurs paramètres) qu'il stocke dans les informations de la transaction qui vient de débuter.

De ce fait, Interbase peut savoir dans quel état été exactement la base de données au moment où cette transaction à été lancée. Par la suite le serveur pourra déterminer quelles versions des enregistrements seront visibles ou modifiables.

En simplifiant, disons que chaque enregistrement dans la base contient le numéro de version au moment de sa création. Interbase retournera ainsi le version de l'enregistrement qui possède le plus grand numéro parmi toutes les versions ayant été validées au démarrage de la transaction en cours.

C'est grâce à cette gestion de versions différentes pour chaque enregistrement, que le système de transactions d'Interbase s'oppose au système de verrous utilisé par la majorité des autres bases de données et permet, notamment, de pouvoir faire des sauvegardes cohérentes sans déconnecter les utilisateurs. L'arbitrage final à lieu au moment de la validation de chaque transaction.

Certes les transactions d'Interbase sont plus puissantes et plus souples, elles réclament en contrepartie des traitements plus complexes lors de leur démarrage. Il est donc judicieux de regrouper le maximum de choses à l'intérieur d'une transaction.

V.3.2. Les accès concurrents

Avec les transaction, on doit à des moments, gérer le conflit entre cohérence des données et concurrence d'accès. Supposons que l'application doive faire des SELECT sur quelques tables, puis, en fonction des valeurs lues, qu'elle fasse une série de mises à jour dans l'autres tables et que ces modifications soient suivies de quelques longues requêtes retournant des totaux.

En se plaçant du côté de la concurrence d'accès aux données, il serait préférable de faire un commit après les modifications, de telle sorte que les autres transactions puissent voir ces nouvelles informations le plus tôt possible. Mais les grandes requêtes finales seront alors exécutées dans une nouvelle transaction et pourront voir toutes les modifications intervenues entre temps. Il se peut donc que les données lues par ces requêtes différent de celles lues par les requêtes de sélection en début de traitement (enregistrements supprimés, modifiés, insérés).

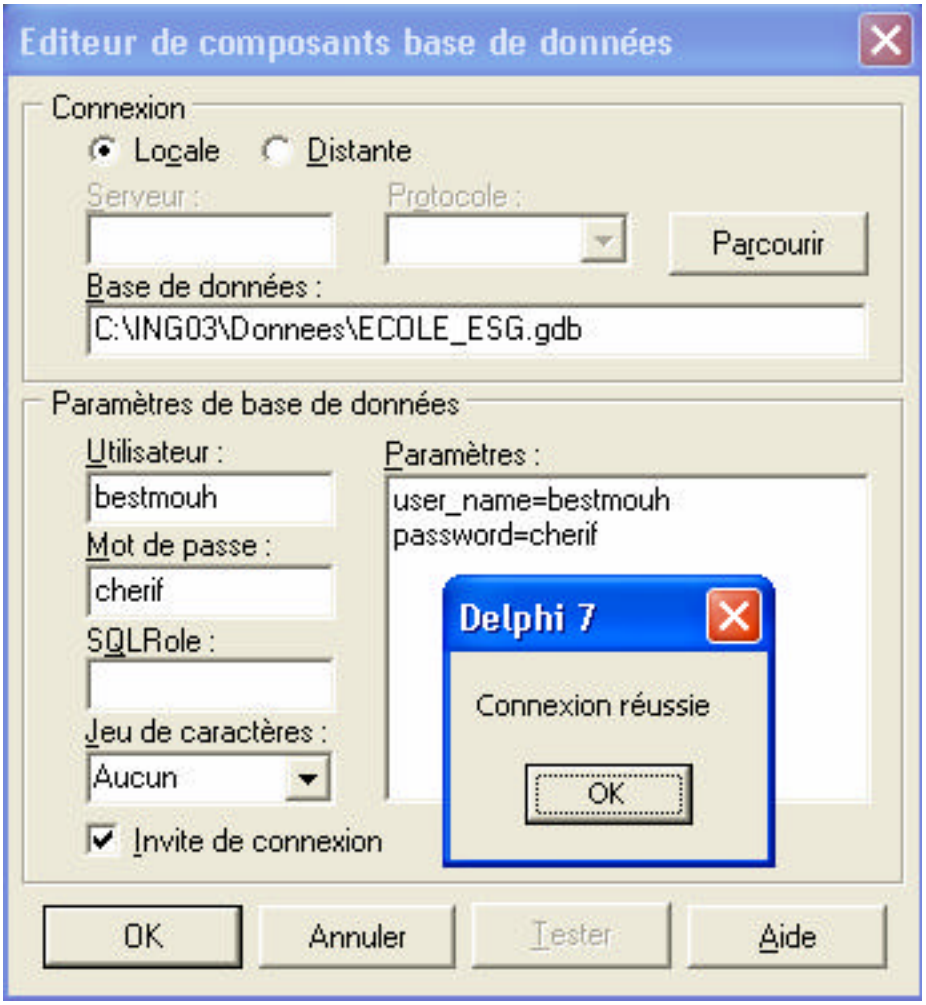
Dans un tel cas, les résultats risquent de ne plus être cohérents. Si on se place sur le plan de la cohérence des données, il faut entourer toutes les opérations dans une seule transaction. Mais bien entendu ce sera au détriment de la concurrence des accès.

Interbase autorise que plusieurs transactions soient ouvertes en même temps par le même client, cela s'effectue en utilisant autant de composants TIBTransaction que nécessaire.

V.4. Application :

V.4.1. Création d'une table

- Créez une nouvelle application.
- Placez un TIbDatabase sur la Form, double cliquez sur IBDatabase1. Et renseigner le dialogue qui s'affiche. Puis tester la connexion avec le bouton « Tester ».



- Vérifiez la connexion en basculant IBDataBase1.Connected sur True, puis fermez la connexion (pour éviter de monopoliser un utilisateur).
- Placez un TIBTransaction sur la Form et mettre IBDatabase1.DefaultTransaction à IBTransaction1.
- De même mettre IBTransaction1.DefaultDatabase à IBDatabase1.
- Placez un TIBQuery sur la Form et mettre sa propriété DatabaseName à IBDatebase1.
- Placez un TButton sur la Form et crée sa méthode OnClick(), placez y les instructions de création :

```
procedure TForm1.Cree_TableClick(Sender: TObject);
begin
IBQuery1.Close;
IBQuery1.SQL.Clear;
IBQuery1.SQL.Add('CREATE TABLE formations ');
IBQuery1.SQL.Add('(f_numero integer, f_nom varchar(30), ');
IBQuery1.SQL.Add('f_jours integer, f_prix numeric(5,2))');
  try
    if IBTransaction1.InTransaction then IBTransaction1.Commit;
    IBTransaction1.StartTransaction;
    IBQuery1.ExecSQL;
    IBTransaction1.Commit;
  except on e:Exception do
    showmessage('problème de création');
  end;
end;
```

V.4.2. Vérifier la création

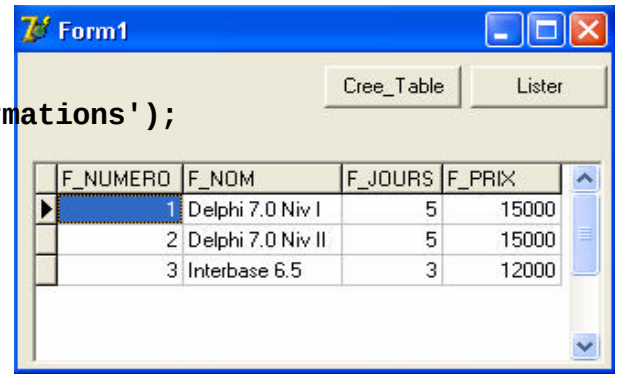
Pour vérifier que la création de la table à été effectuée, soit on utilise l'Explorateur de bases de données Delphi, soit en lisant les données de cette table dans l'application.

Pour lire les données, il suffit d'envoyer la requête :

```
SELECT * From formations
```

- Ajouter un second TIBQuery sur la Form, et mettre sa propriété DatabaseName à IBDatabase1.
- Placez un second TButton sur la Form et placez-y la requête de lecture.
- Insérez un DataSource ; mettre sa propriété DataSet à IBQuery2.
- Insérez un DBGrid ; initialisez sa propriété DataSource à DataSource1.

```
IBQuery2.Close;
IBQuery2.SQL.Clear;
IBQuery2.SQL.Add('select * from formations');
IBQuery2.Open;
```



V.4.3. Effacer une table

Pour supprimer une table, il faut exécuter la requête :

```
DROP TABLE formations
```

- Ajouter un troisième TIBQuery sur la Form, et mettre sa propriété DatabaseName à IBDatabase1.
- Placez un autre TButton sur la Form et placez-y la requête de suppression.

```
procedure TForm1.Drop_TableClick(Sender: TObject);
begin
IBQuery3.Close;
IBQuery3.SQL.Clear;
IBQuery3.SQL.Add('DROP TABLE formations ');
  try
    if IBTransaction1.InTransaction then IBTransaction1.Commit;
    IBTransaction1.StartTransaction;
    IBQuery3.ExecSQL;
    IBTransaction1.Commit;
  except on e:Exception do
    showmessage('Problème de Suppression');
  end;
end;
```

Remarques :

- Nous avons utilisé trois IBQuery, nous aurions pu utiliser le même.
- On à utilisé un IBQuery au lieu du IBSql, car c'est le premier qui permet à la fois la modification de données du serveur (Create, Drop, etc.…) et la lecture de données depuis le serveur (SELECT).
- Pour envoyer une requête de modification il faut utiliser TIBQuery.ExecSql, et pour la lecture c'est TIBQuery.Open (qui est équivalent à TIBQuery.Active : = True ;)
- On ne peut pas créer une table à partir de l'inspecteur d'objets en mode conception (il faut exécuter du code).

V.4.4. Ajouter des données

Pour ajouter un enregistrement dans une table, l'instruction SQL est :

```
    INSERT INTO formations
     (F-numero, F-nom, F-jours, F-prix)
     values (6, 'Interbase Delphi', 3, 15000.00)
```

- Placez un TIBTransaction sur une Form.
- Placez un TIBDatabase sur la Form, double cliquez dessus, renseignez les champs.
- Vérifier ensuite la connexion en basculant IBDatabase1.Connected sur True, puis fermez la connexion.
- Mettre IBDatabase1.DefaultTransaction à IBTransaction1.
- Placez un TIBQuery sur la form.
- Initialisez sa propriété DatabaseName à IBDatabase1.
- Placez un TButton sur la Form, et créez sa méthode OnClick().

```
procedure TForm1.Insert_DataClick(Sender: TObject);
begin
IBQuery4.Close;
IBQuery4.SQL.Clear;
IBQuery4.SQL.Add('INSERT INTO formations ');
IBQuery4.SQL.Add('(f_numero, f_nom, f_jours, f_prix)');
IBQuery4.SQL.Add(' VALUES(4,"Visual C++ Init",10,30000)');
  try
    if not IBTransaction1.InTransaction then
IBTransaction1.StartTransaction;
    //IBTransaction1.StartTransaction;
    IBQuery4.ExecSQL;
    IBTransaction1.Commit;
  except on e:Exception do
    showmessage('Problème d"insertion');
  end;
end;
```

Les valeurs a insérer ont été figées, nous pouvons automatiser cet ajout par :

- Soit des scripts.
- Soit en mode Interactif.
- Soit une procédure paramétrée

V.4.5. Les types de données

V.4.5.1. Type CHARACTER

Pour spécifier le type CHARACTER, SQL exige que la chaîne soit entourée de guillemets. Suivant les serveurs, il faut utiliser un guillemet simple ou double.

```
VALUES (6, 'Interbase Delphi', 3, 15000)
```

Ou

```
VALUES (6, " Interbase Delphi ", 3, 15000)
```

Si la valeur est incluse dans une string Pascal, il faut dédoubler les guillemets.

```
IbQuery1.Sql.Add(' Values(6, "Interbase Delphi", 3, 15000)')
```

Pour simplifier Delphi propose la méthode QuotedStr.

```
IbQuery1.Sql.Add('Values(6, '+QuotedStr('Interbase Delphi')+',3, 15000)').
```

V.4.5.2. Type Numeric

Pour les valeurs numériques avec décimales :

- Au US la partie entière est séparée de la partie décimale par un point 3.14, alors qu'en Europe c'est une virgule 3,14.
- Les valeurs utilisées en Pascal objet doivent respecter la syntaxe US (c à d utilisé le point '.') : `A : = 3.14 ;`
- Les composants visuels (Tedit, …) et les primitives de conversion (FlotToStr, …) utilisent la virgule ','.
- On peut imposer le séparateur à utiliser : `DecimalSeparator : = '.' ;`

V.4.5.3. Le type DATE

- SQL utilise le format US « Année mois jour » **EXCLUSIVEMENT**.
- Delphi utilise un format US ou européanisé suivant le type de primitive.
- Le séparateur doit être un Slache « / » : 2004/08/18, pour désigner le 18 Août 2004.
- Les dates doivent être entourée de guillemets :'2004/08/18' sinon Delphi effectue une division et traduit le résultat en une date (ce qui est erroné).

```
INSERT INTO Dates (d-numero, d-date)
Values (6, '2004/09/30')
```

Ou

```
IbQuery1.Sql.Add('INSERT INTO Dates') ;
IbQuery1.Sql.Add(' (d-numero, d-date)') ;
IbQuery1.Sql.Add(' values (6, "2004/09/30")');
```

V.4.6. Affichage des données

V.4.6.1. Principe

Pour afficher un enregistrement, on doit d'abord récupérer ses valeurs du serveur. Pour lire les données contenues dans une table, SQL utilise l'instruction SELECT ;

```
SELECT f-numero, f-nom
From Formations
```

Lorsque le serveur reçoit cette requête :

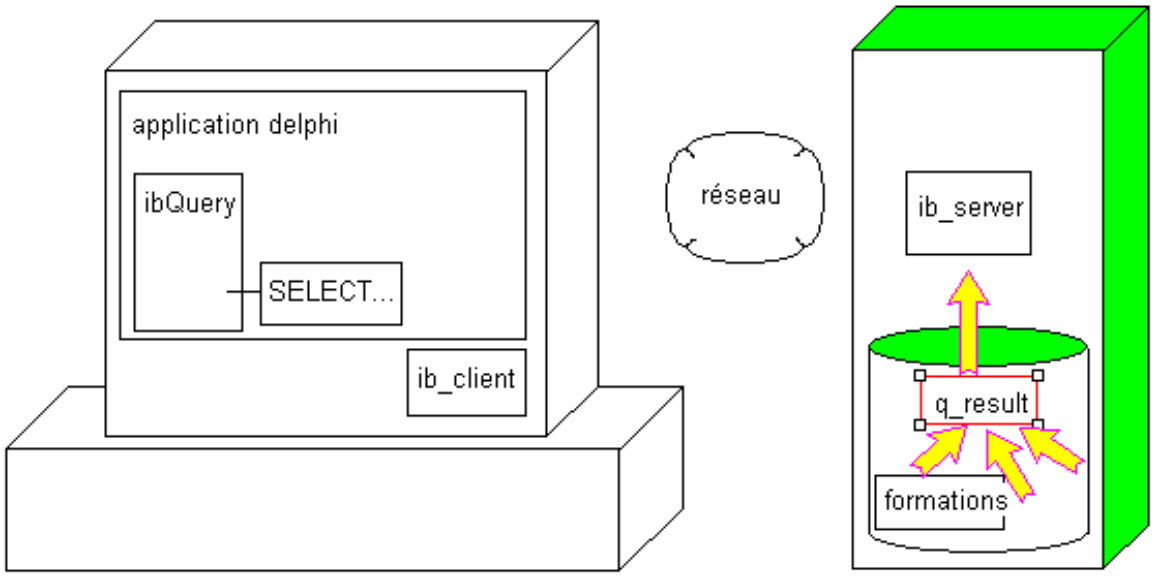
- Il vérifie sa syntaxe.
- Il construit une table contenant les valeurs demandées.
- Ces données sont envoyé au client.

- Placer un TIBTransaction sur une Form.
- Placez un TIBDatabase sur la Form, double cliquez dessous, renseignez les champs. Vérifier ensuite la connexion en basculant IBDatabase1.Connected sur True, puis fermez la connexion.
- Mettre IbDatabase1.DefaultTransaction à IBTransaction1.
- Placez un TIBQuery sur la Form. Initialisez sa propriété DatabaseName à IbDatabase1.
- Placez un TButton sur la Form, et Créez sa méthode OnClick().

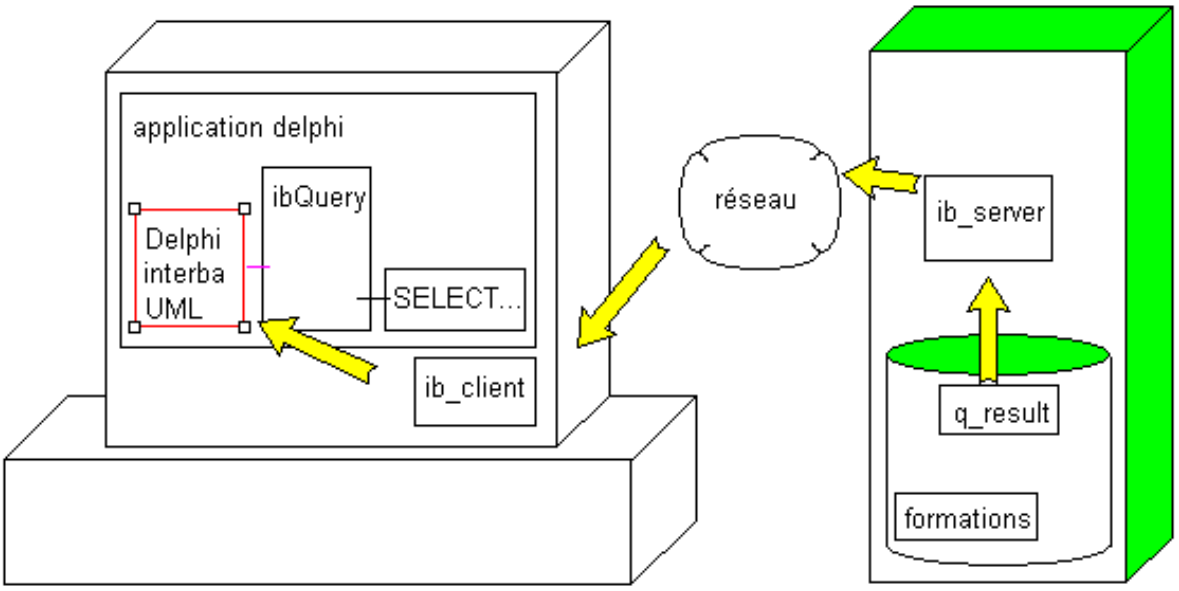
```
procedure TForm1.ListerClick(Sender: TObject);
begin
     IBQuery2.Close;
     IBQuery2.SQL.Clear;
     IBQuery2.SQL.Add('select * from formations');
     IBQuery2.Open;
end;
```

L'execution de **IBQuery1.Open;**

- Delphi envoie la requête Select au serveur.
- Celui ci construit une table résultat, en utilisant la table formations mais éventuellement des tables annexes (index, contraintes etc.…).



- Cette table résultat est envoyée via le réseau au client Interbase, celui ci la transmet ensuite à l'application. Delphi place alors ce résultat dans un tampon mémoire associé à TIBQuery.



V.4.6.2. Affichage des résultats

Les données étant récupérées par IBQuery, on peut les afficher ou les traiter en mémoire.

Pour afficher les données on utilise :

- Un TDataSource qui récupère les données du TIBQuery.
- Un composant d'affichage relié au TDataSorce par exemple un TDBGrid.

Par conséquent :

- Sélectionnez dans la palette des composants, le composant TDataSource et placez le sur la From. Sélectionnez ensuite un TDBGrid.
- Mettez sa propriété DataSource à DataSource1.
- Cliquez sur le bouton qui lance la requête.

Remarques :

- L'éditeur de SQL peut être utilisé.
- Certes « select * From formations » retourne bien les mêmes données que celle de la table sur disque « formations ». Mais ce n'est qu'un instantané réalisé lorsque la requête à été lancé. Il s'agit d'une copie. Si un autre utilisateur ajoute une nouvelle formation, la copie que nous avons dans la cache attaché à IBQuery ne le reflète pas.
- Pour rafraîchir notre cache, la seule façon de faire est de relancer la requête (en fermant et rouvrant TIBQuery).