Introduction au langage SQL avec le SGBD Firebird

Serge Tahé, université d'Angers, janvier 2006

1 Introduction

Ce document reprend un document analogue écrit en 1991 pour Oracle et l'adapte au SGBD Firebird. En 1991, l'essence du document provenait de la documentation officielle d'ORACLE et pour certains points d'un excellent ouvrage de l'époque écrit par Christian MAREE et Guy LEDANT : SQL Initiation, Programmation et Maîtrise paru chez EYROLLES. Ce livre n'existe plus à ma connaissance. Les auteurs ont écrit un nouveau livre que je n'ai pas eu l'occasion de lire : SQL 2 Initiation / Programmation.

SQL (*Structured Query Language*) est un langage standard de création, de maintenance et d'interrogation de bases de données relationnelles. Il est largement indépendant du SGBD utilisé. Si les exemples de ce document ont été écrits à l'aide du SGBD Firebird, ils peuvent cependant, pour la plupart, être reproduits avec tout SGBD relationnel. Sous Windows, on trouvera divers SGBD : des produits libres tels MySQL, Postgres, Firebird, SQL Express 2005 et également des produits commerciaux : Access, SQL Server, Oracle. Ces SGBD acceptent tous le langage SQL mais parfois avec des variantes propriétaires, souvent des extensions apportées au langage SQL standard.

Le SGBD Firebird a été choisi parce qu'il présente la caractéristique d'encapsuler une base de données dans un unique fichier. Il est alors pratique pour des étudiants de mettre ce fichier sur une clé USB afin de l'exploiter sur leur ordinateur personnel ou ceux de leur école ou université.

Ce document est destiné à des débutants ou à des personnes désirant rafraîchir leurs souvenirs sur SQL. Il ne prétend pas couvrir SQL intégralement. Les notions de programmation (procédures stockées, programmation SQL, API SQL) ne sont par exemple pas abordées ni celles d'administration d'un SGBD.

Il n'est par ailleurs pas sûr qu'il soit exempt d'erreurs de fond ou de forme. Les suggestions constructives peuvent être envoyées à l'adresse serge.tahe@univ-angers.fr.

Serge Tahé, janvier 2006

Table des matières

2 Tutoriel Firebird

Avant d'aborder les bases du langage SQL, nous présentons au lecteur comment installer le SGBD Firebird ainsi que le client graphique IB-Expert.

2.1 Où trouver Firebird ?

Le site principal de Firebird est [http://firebird.sourceforge.net/]. La page de téléchargements offre les liens suivants (avril 2005) :

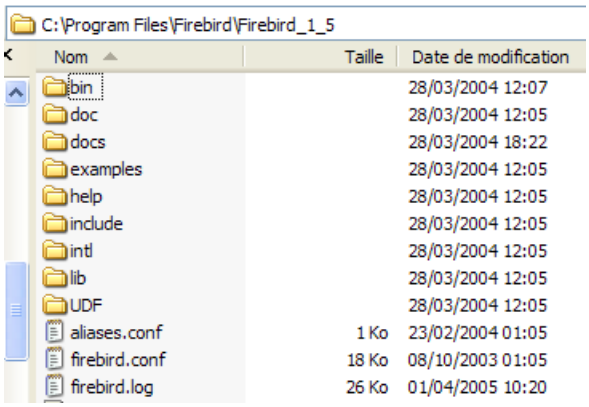
Latest File Releases

Package	Version	Date	Notes / Monitor	Download
firebird	1.5.2-Release (Source)	December 25, 2004	-	Download
firebird-addons	FbConfigManager for v1.5	April 9, 2003	-	Download
firebird-benchmarks	as3ap	February 4, 2003	-	Download
firebird-freebsd-i386	1.5.2-Release	January 4, 2005	-	Download
firebird-hpux	1.0.3-Release	May 12, 2004	-	Download
firebird-jca-jdbc-driver	1.5.5-release-src	December 5, 2004	-	Download
firebird-linux-i386	1.5.2-Release	December 25, 2004	-	Download
firebird-MacOS-X/darwin	1.5.1-Release	August 5, 2004	-	Download
firebird-net-provider	1.7 RC 2	February 25, 2005	-	Download
firebird-ODBC-driver	1.2-Release	August 19, 2004	-	Download
firebird-sinixz	1.5.0-Beta1	January 29, 2003	-	Download
firebird-solaris-sparc	1.0.0-Release	March 30, 2002	-	Download
firebird-solaris-x86	1.5.2-Release	February 3, 2005	-	Download
firebird-win32	1.5.2-Release	December 25, 2004	-	Download
jdbc-InterClient	2.01	November 13, 2001	-	Download

On téléchargera les éléments suivants :

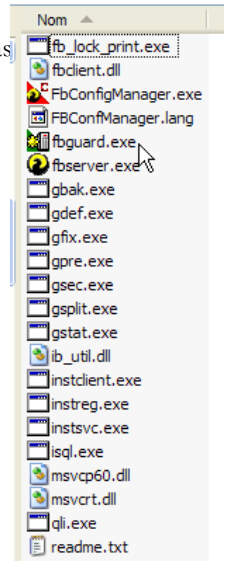
`firebird-win32`	le SGBD pour Windows
`firebird-net-provider`	une bibliothèque de classes pour les applications .NET qui permet d'accéder au SGBD sans passer par un pilote ODBC.
`firebird-ODBC-driver`	le pilote ODBC de Firebird

Faire l'installation de ces éléments. Le SGBD est installé dans un dossier dont le contenu est analogue au suivant :

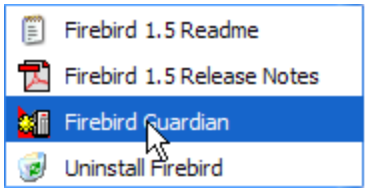


Les binaires sont dans le dossier [bin] :

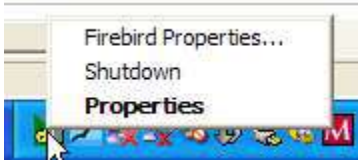
`fbguard.exe` permet de lancer/arrêter le SGBD
`isql.exe` client ligne permettant de gérer des bas



On notera que par défaut, l'administrateur du SGBD s'appelle [SYSDBA] et son mot de passe est [masterkey]. Des menus ont été installés dans [Démarrer] :



L'option [Firebird Guardian] permet de lancer/arrêter le SGBD. Après le lancement, l'icône du SGBD reste dans la barre des tâches de windows :



Pour créer et exploiter des bases de données Firebird avec le client ligne [isql.exe], il est nécessaire de lire la documentation livrée avec le produit dans le dossier [doc].

2.2 La documentation de Firebird

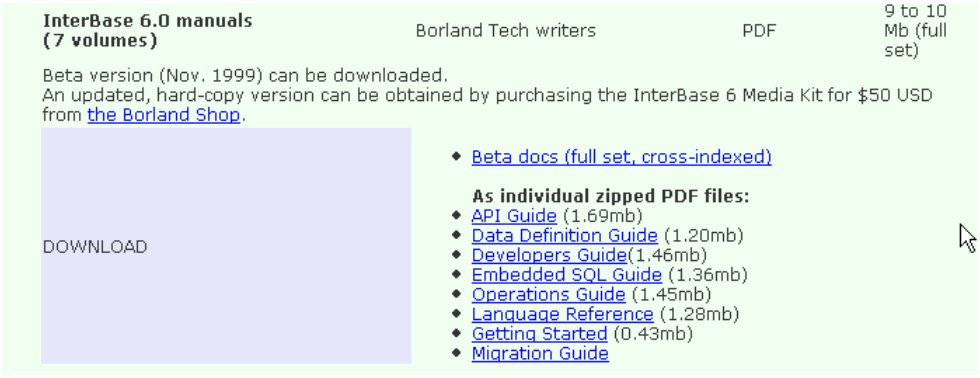
La documentation sur Firebird et sur le langage SQL peut être trouvée sur le site de Firebird (janvier 2006) :

Divers manuels sont disponibles en anglais :

`Firebird 1.5 Quick Start Guide` pour démarrer avec FB

`Firebird 1.5 Error Codes` pour comprendre les codes d'erreur renvoyés par FB

Des manuels de formation au langage SQL sont également disponibles :



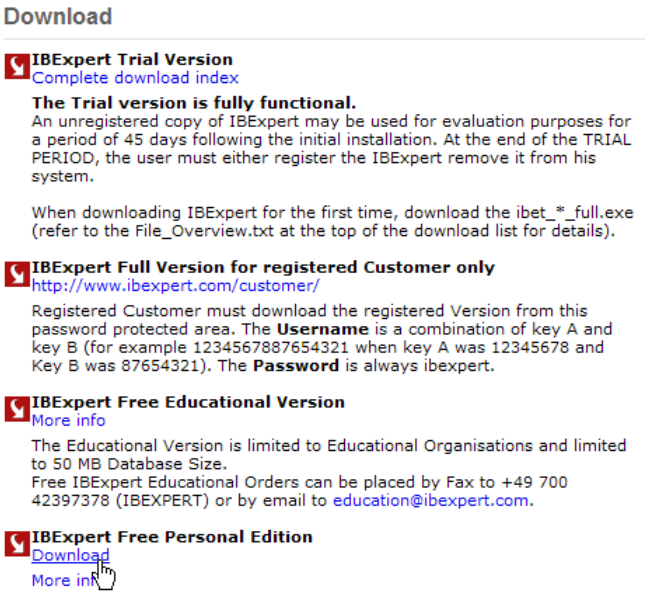
`Data Definition Guide` pour découvrir comment créer des tables, quels types de données sont utilisables, ...

`Language Reference` le guide de référence pour apprendre SQL avec Firebird

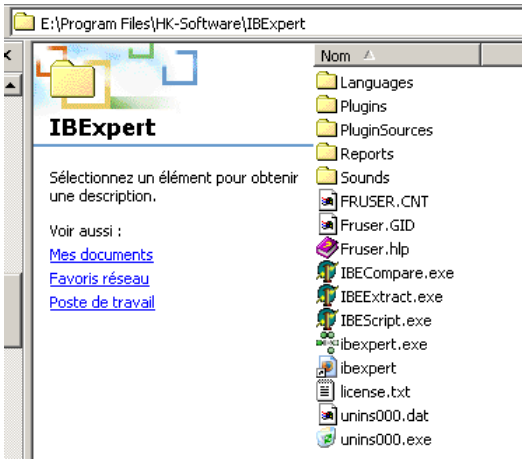
Une façon rapide de travailler avec Firebird et d'apprendre le langage SQL est d'utiliser un client graphique. Un tel client est IB-Expert décrit au paragraphe suivant.

2.3 Travailler avec le SGBD Firebird grâce à IB-Expert

Le site principal de IB-Expert est [http://www.ibexpert.com/]. La page de téléchargements offre les liens suivants :

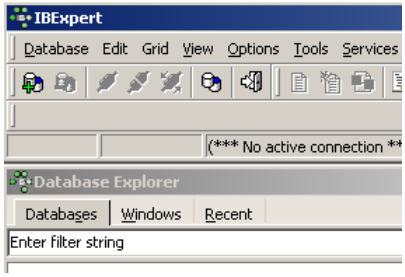


On choisira la version libre [Personal Edition]. Une fois celle-ci téléchargée et installée, on dispose d'un dossier analogue au suivant :



L'exécutable est [ibexpert.exe]. Un raccourci est normalement disponible dans le menu [Démarrer] :

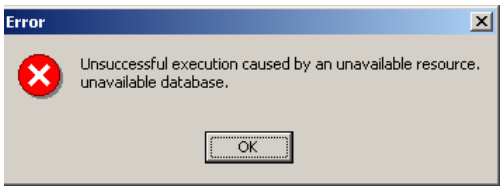
Une fois lancé, IBExpert affiche la fenêtre suivante :



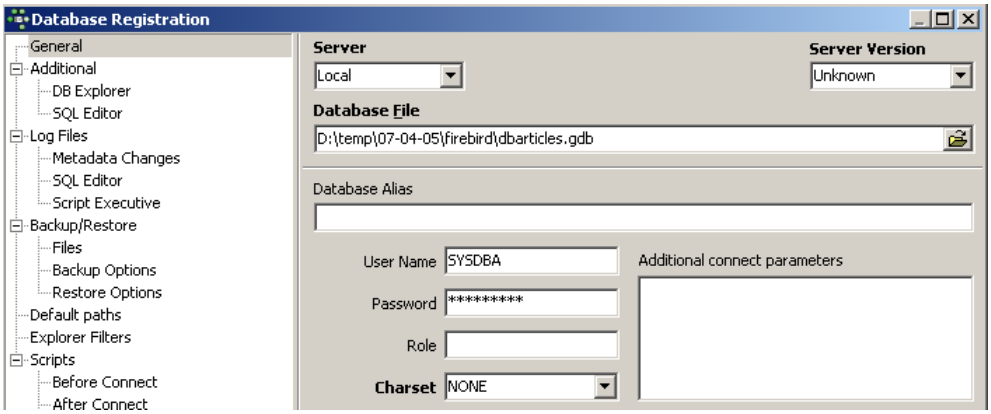
Utilisons l'option [Database/Create Database] pour créer une base de données :

`Server`	peut être [local] ou [remote]. Ici notre serveur est sur la même machine que [IBExpert]. On choisit donc [local]
`Database`	utiliser le bouton de type [dossier] du combo pour désigner le fichier de la base. Firebird met toute la base dans un unique fichier. C'est l'un de ses atouts. On transporte la base d'un poste à l'autre par simple copie du fichier. Le suffixe [.gdb] est ajouté automatiquement.
`Username`	**SYSDBA** est l'administrateur par défaut des distributions actuelles de Firebird
`Password`	**masterkey** est le mot de passe de l'administrateur SYSDBA des distributions actuelles de Firebird
`Dialect`	le dialecte SQL à utiliser
`Register Database`	si la case est cochée, IBExpert présentera un lien vers la base créée après avoir créé celle-ci

Si en cliquant le bouton [OK] de création, vous obtenez l'avertissement suivant :



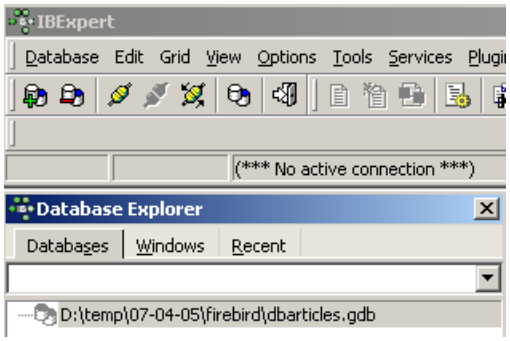
c'est que vous n'avez pas lancé Firebird. Lancez-le. On obtient une nouvelle fenêtre :



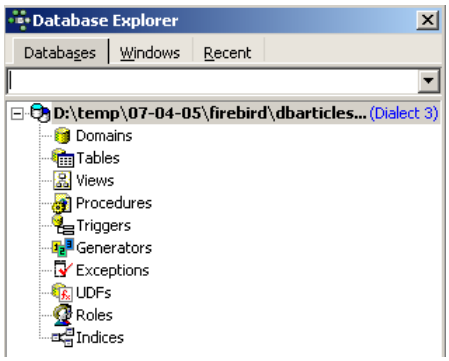
`Charset` Famille de caractères à utiliser. Bien que la copie d'écran ci-dessus n'indique aucune information, il est conseillé de prendre dans la liste déroulante la famille [ISO-8859-1] qui permet d'utiliser les caractères latins accentués.

`Server version` [IBExpert] est capable de gérer différents SGBD dérivés d'Interbase. Prendre la version de Firebird que vous avez installée :

Une fois cette nouvelle fenêtre validée par [Register], on a le résultat suivant :

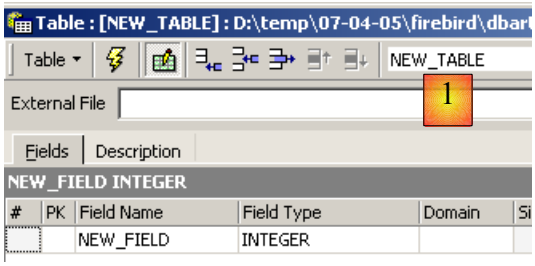


Pour avoir accès à la base créée, il suffit de double-cliquer sur son lien. IBExpert expose alors une arborescence donnant accès aux propriétés de la base :

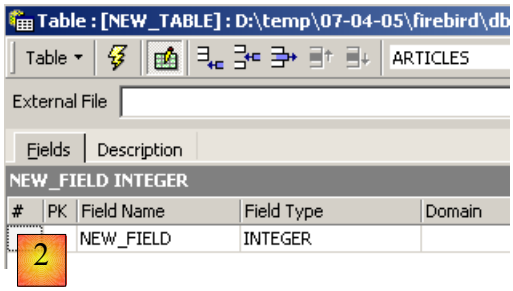


2.4 Création d'une table de données

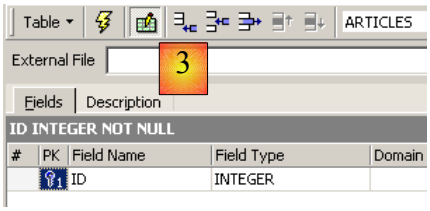
Créons une table. On clique droit sur [Tables] (cf fenêtre ci-dessus) et on prend l'option [New Table]. On obtient la fenêtre de définition des propriétés de la table :



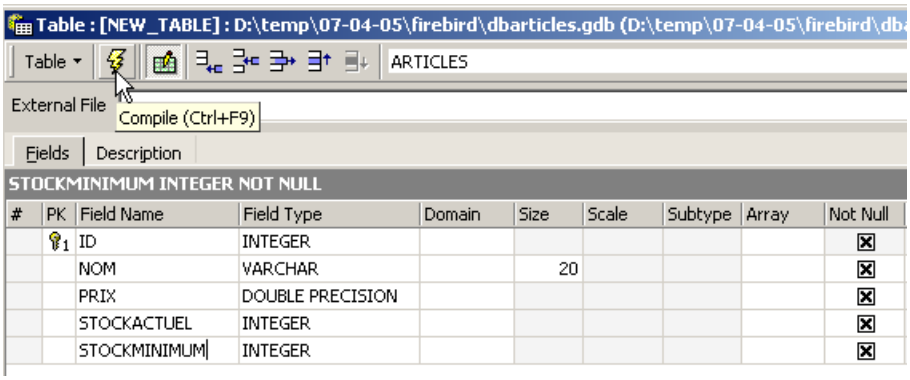
Commençons par donner le nom [ARTICLES] à la table en utilisant la zone de saisie [1] :



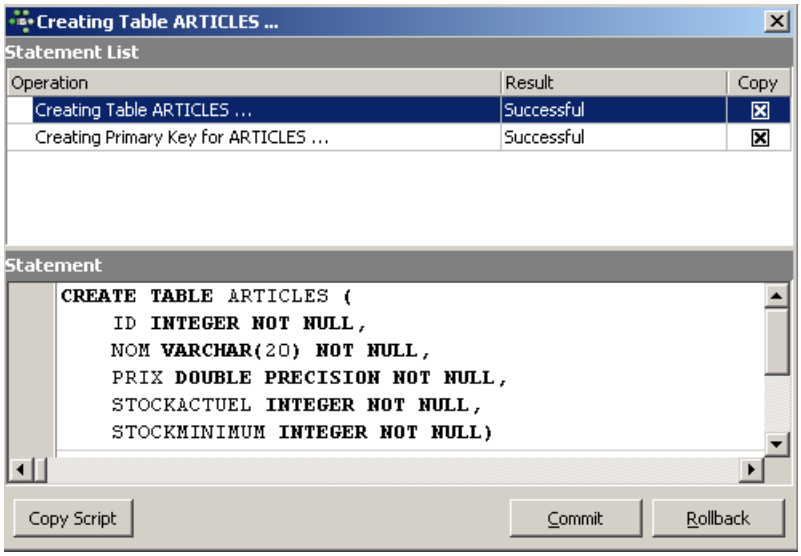
Utilisons la zone de saisie [2] pour définir une clé primaire [ID] :



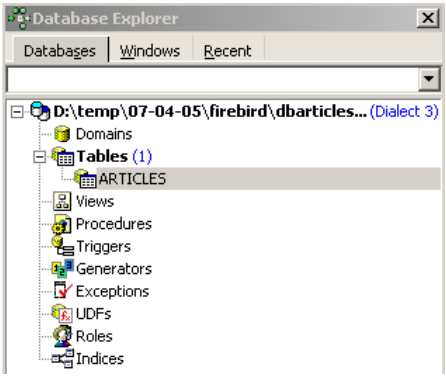
Un champ est fait clé primaire par un double-clic sur la zone [PK] (Primary Key) du champ. Ajoutons des champs avec le bouton situé au-dessus de [3] :



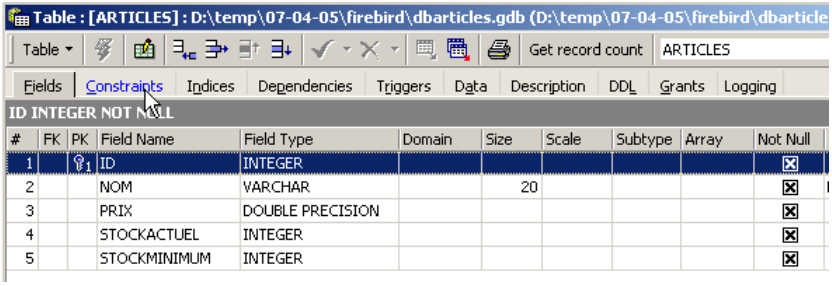
Tant qu'on n'a pas " compilé " notre définition, la table n'est pas créée. Utilisons le bouton [Compile] ci-dessus pour terminer la définition de la table. IBExpert prépare les requêtes SQL de génération de la table et demande confirmation :



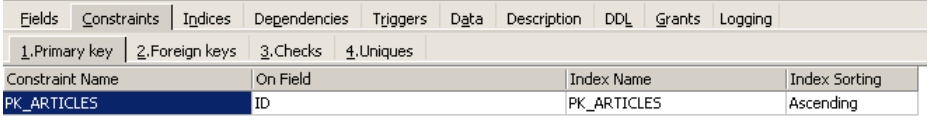
De façon intéressante, IBExpert affiche les requêtes SQL qu'il a exécutées. Cela permet un apprentissage à la fois du langage SQL mais également du dialecte SQL éventuellement propriétaire utilisé. Le bouton [Commit] permet de valider la transaction en cours, [Rollback] de l'annuler. Ici on l'accepte par [Commit]. Ceci fait, IBExpert ajoute la table créée, à l'arborescence de notre base de données :



En double-cliquant sur la table, on a accès à ses propriétés :



Le panneau [Constraints] nous permet d'ajouter de nouvelles contraintes d'intégrité à la table. Ouvrons-le :



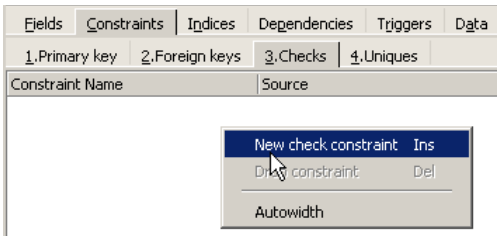
On retrouve la contrainte de clé primaire que nous avons créée. On peut ajouter d'autres contraintes :

- des clés étrangères [Foreign Keys]
- des contraintes d'intégrité de champs [Checks]
- des contraintes d'unicité de champs [Uniques]

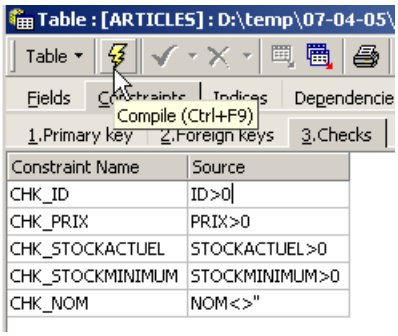
Indiquons que :

- les champs [ID, PRIX, STOCKACTUEL, STOKMINIMUM] doivent être >0
- le champ [NOM] doit être non vide et unique

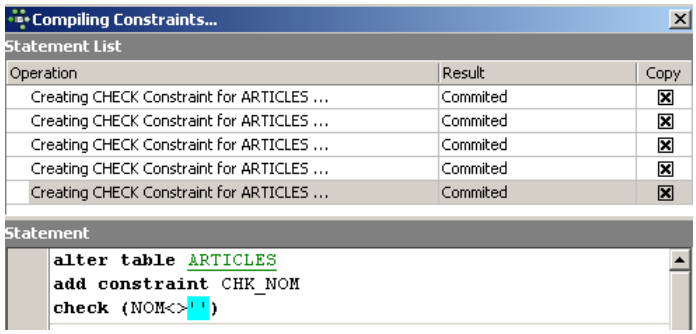
Ouvrons le panneau [Checks] et cliquons droit dans son espace de définition des contraintes pour ajouter une nouvelle contrainte :



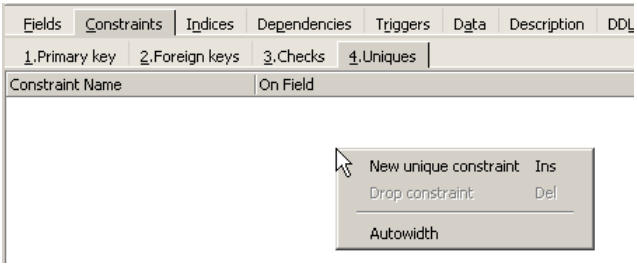
Définissons les contraintes souhaitées :



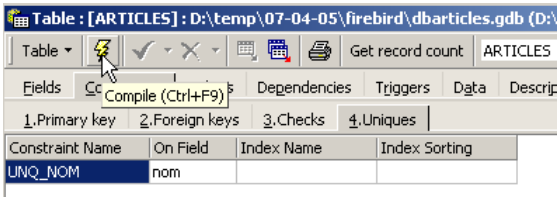
On notera ci-dessus, que la contrainte [NOM<>''] utilise deux apostrophes et non des guillemets. Compilons ces contraintes avec le bouton [Compile] ci-dessus :



Là encore, IBExpert fait preuve de pédagogie en indiquant les requêtes SQL qu'il a exécutées. Passons maintenant au panneau [Constraints/Uniques] pour indiquer que le nom doit être unique. Cela signifie qu'on ne peut pas avoir deux fois le même nom dans la table.



Définissons la contrainte :



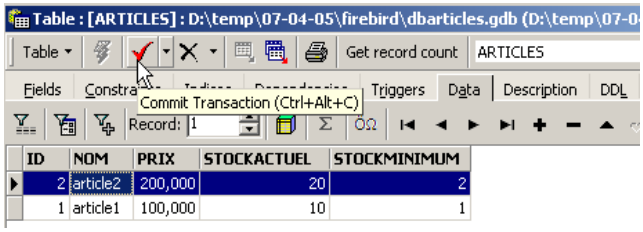
Puis compilons-la. Ceci fait, ouvrons le panneau [DDL] (Data Definition Language) de la table [ARTICLES] :

Celui-ci donne le code SQL de génération de la table avec toutes ses contraintes. On peut sauvegarder ce code dans un script afin de le rejouer ultérieurement :

```
SET SQL DIALECT 3;
SET NAMES NONE;
CREATE TABLE ARTICLES (
    ID             INTEGER NOT NULL,
    NOM            VARCHAR(20) NOT NULL,
    PRIX           DOUBLE PRECISION NOT NULL,
    STOCKACTUEL    INTEGER NOT NULL,
    STOCKMINIMUM   INTEGER NOT NULL
);
ALTER TABLE ARTICLES ADD CONSTRAINT CHK_ID check (ID>0);
ALTER TABLE ARTICLES ADD CONSTRAINT CHK_PRIX check (PRIX>0);
ALTER TABLE ARTICLES ADD CONSTRAINT CHK_STOCKACTUEL check (STOCKACTUEL>0);
ALTER TABLE ARTICLES ADD CONSTRAINT CHK_STOCKMINIMUM check (STOCKMINIMUM>0);
ALTER TABLE ARTICLES ADD CONSTRAINT CHK_NOM check (NOM<>'');
ALTER TABLE ARTICLES ADD CONSTRAINT UNQ_NOM UNIQUE (NOM);
ALTER TABLE ARTICLES ADD CONSTRAINT PK_ARTICLES PRIMARY KEY (ID);
```

2.5 Insertion de données dans une table

Il est maintenant temps de mettre des données dans la table [ARTICLES]. Pour cela, utilisons son panneau [Data] :



Les données sont entrées par un double-clic sur les champs de saisie de chaque ligne de la table. Une nouvelle ligne est ajoutée avec le bouton [+], une ligne supprimée avec le bouton [-]. Ces opérations se font dans une transaction qui est validée par le bouton [Commit Transaction] (cf ci-dessus). Sans cette validation, les données seront perdues.

2.6 L'éditeur SQL de [IB-Expert]

Le langage SQL (Structured Query Language) permet à un utilisateur de :

1. créer des tables en précisant le type de données qu'elle va stocker, les contraintes que ces données doivent vérifier
2. d'y insérer des données
3. d'en modifier certaines
4. d'en supprimer d'autres
5. d'en exploiter le contenu pour obtenir des informations
6. ...

IBExpert permet à un utilisateur de faire les opérations 1 à 4 de façon graphique. Nous venons de le voir. Lorsque la base contient de nombreuses tables avec chacune des centaines de lignes, on a besoin de renseignements difficiles à obtenir visuellement.

Supposons par exemple qu'un magasin virtuel sur le web ait des milliers d'acheteurs par mois. Tous les achats sont enregistrés dans une base de données. Au bout de six mois, on découvre qu'un produit « X » est défaillant. On souhaite contacter toutes les personnes qui l'ont acheté afin qu'elles renvoient le produit pour un échange gratuit. Comment trouver les adresses de ces acheteurs ?

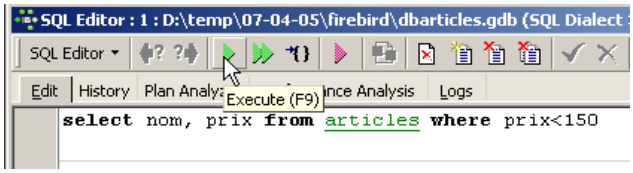
1. On peut consulter visuellement toutes les tables et chercher ces acheteurs. Cela prendra quelques heures.
2. On peut émettre un ordre SQL qui va donner la liste de ces personnes en quelques secondes

Le langage SQL est utile dès

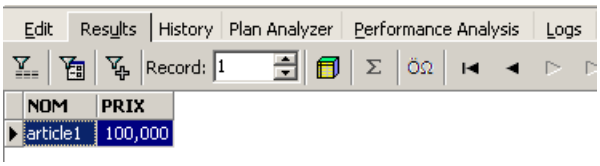
- que la quantité de données dans les tables est importante
- qu'il y a beaucoup de tables liées entre-elles
- que l'information à obtenir est répartie sur plusieurs tables
- ...

Nous présentons maintenant l'éditeur SQL d'IBExpert. Celui-ci est accesible via l'option [Tools/SQL Editor] ou [F12] :

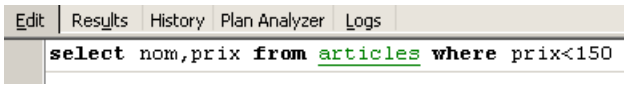
On a alors accès à un éditeur de requêtes SQL évolué avec lequel on peut jouer des requêtes. Tapons une requête :



On exécute la requête SQL avec le bouton [Execute] ci-dessus. On obtient le résultat suivant :

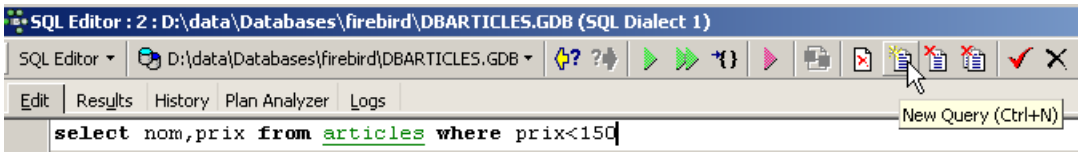


Ci-dessus, l'onglet [Results] présente la table résultat de l'ordre SQL [Select]. Pour émettre une nouvelle commande SQL, il suffit de revenir sur l'onglet [Edit]. On retrouve alors l'ordre SQL qui a été joué.

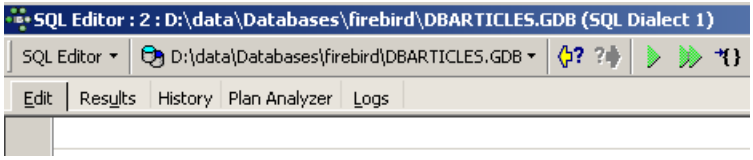


Plusieurs boutons de la barre d'outils sont utiles :

- le bouton [New Query] permet de passer à une nouvelle requête SQL :

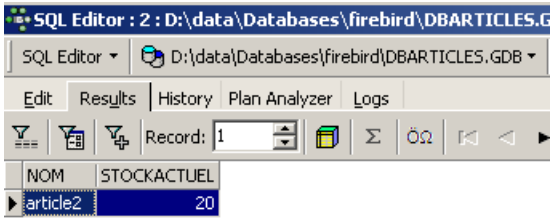


On obtient alors une page d'édition vierge :



On peut alors taper un nouvel ordre SQL :

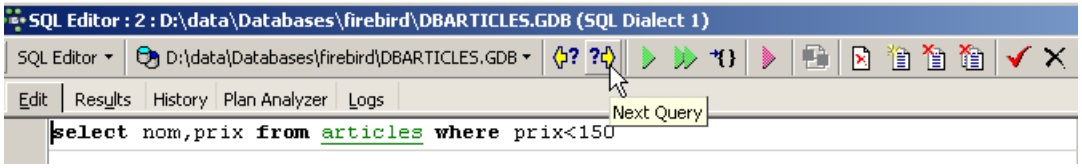
et l'exécuter :



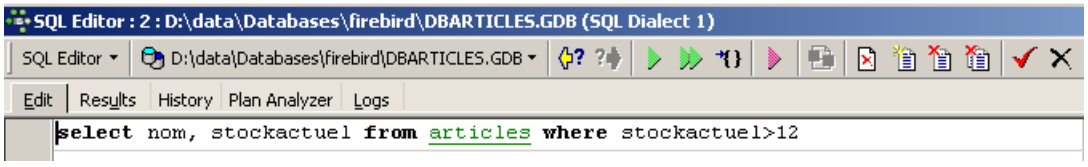
Revenons sur l'onglet [Edit]. Les différents ordre SQL émis sont méorisés par [IB-xpert]. Le bouton [Previous Query] permet de revenir à un ordre SQL émis antérieurement :

On revient alors à la requête précédente :

Le bouton [Next Query] permet lui d'aller à l'ordre SQL suivant :

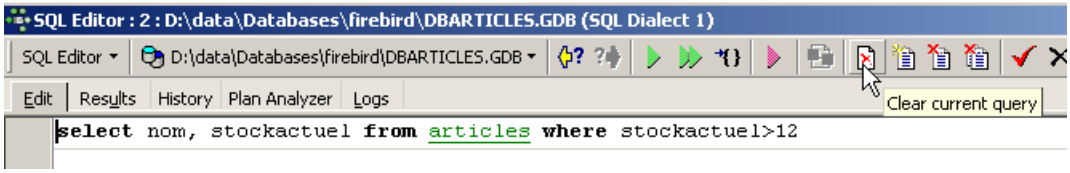


On retrouve alors l'ordre SQL qui suit dans la liste des ordres SQL mémorisés :

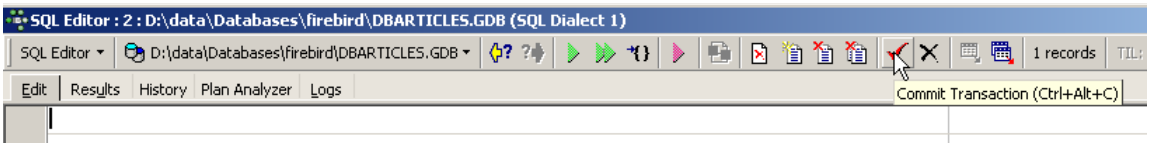


Le bouton [Delete Query] permet de supprimer un ordre SQL de la liste des ordres mémorisés :

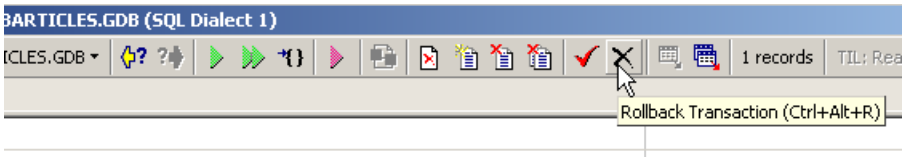
Le bouton [Clear Current Query] permet d'effacer le contenu de l'éditeur pour l'ordre SQL affiché :



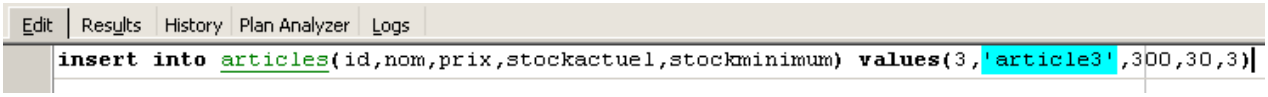
Le bouton [Commit] permet de valider définitivement les modifications faites à la base de données :



Le bouton [RollBack] permet d'annuler les modifications faites à la base depuis le dernier [Commit]. Si aucun [Commit] n'a été fait depuis la connexion à la base, alors ce sont les modifications faites depuis cette connexion qui sont annulées.

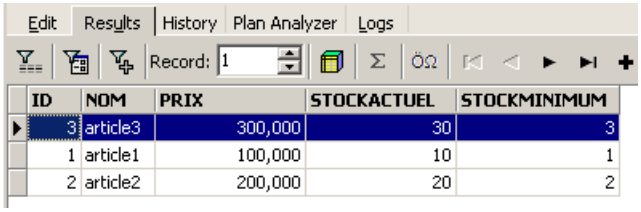


Prenons un exemple. Insérons une nouvelle ligne dans la table :

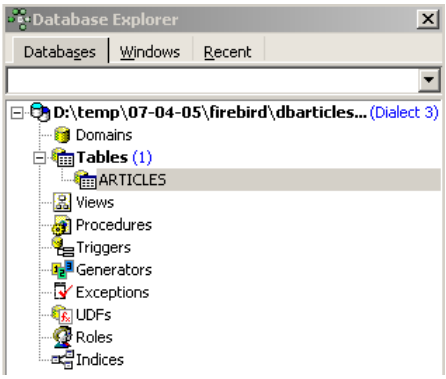


L'ordre SQL est exécuté mais aucun affichage ne se produit. On ne sait pas si l'insertion a eu lieu. Pour le savoir, exécutons l'orde SQL suivant [New Query] :

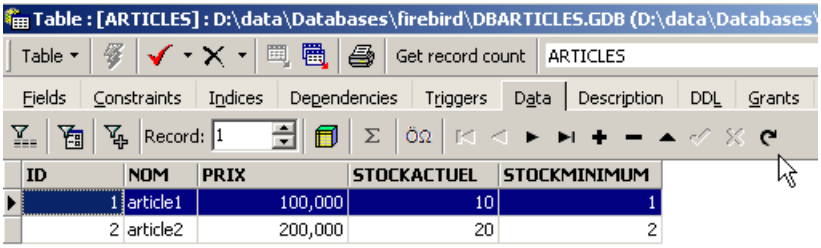
On obtient [Execute] le résultat suivant :



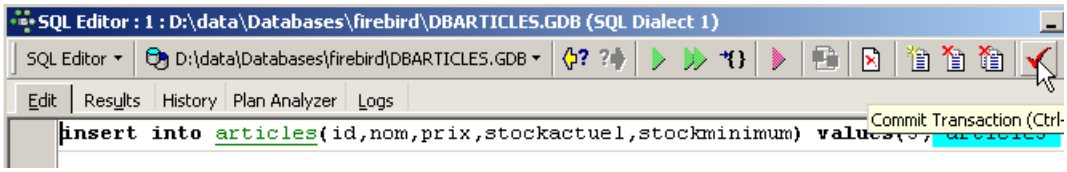
La ligne a donc bien été insérée. Examinons le contenu de la table d'une autre façon maintenant. Double-cliquons sur la table [ARTICLES] dans l'explorateur de bases :



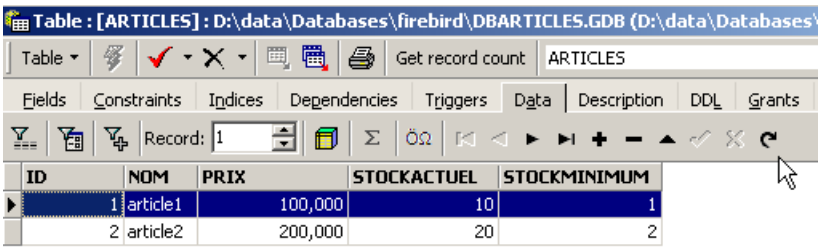
On obtient la table suivante :



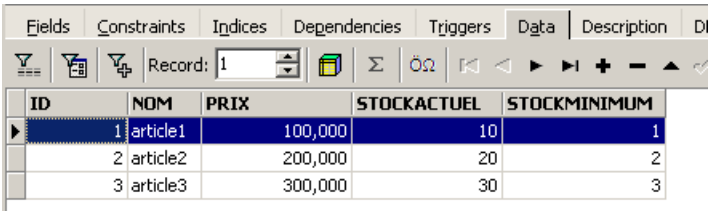
Le bouton fléché ci-dessus permet de rafraîchir la table. Après rafraîchissement, la table ci-dessus ne change pas. On a l'impression que la nouvelle ligne n'a pas été insérée. Revenons à l'éditeur SQL (F12) puis validons l'ordre SQL émis avec le bouton [Commit] :



Ceci fait, revenons sur la table [ARTICLES]. Nous pouvons constater que rien n'a changé même en utilisant le bouton [Refresh] :

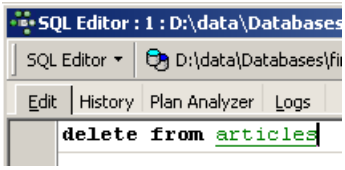


Ci-dessus, ouvrons l'onglet [Fields] puis revenons sur l'onglet [Data]. Cette fois-ci la ligne insérée apparaît correctement :



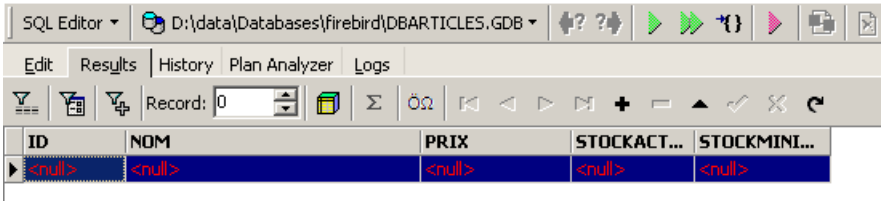
Quand commence l'émission des différents ordres SQL, l'éditeur ouvre ce qu'on appelle une **transaction** sur la base. Les modifications faites par ces ordres SQL de l'éditeur SQL ne seront visibles que tant qu'on reste dans le même éditeur SQL (on peut en ouvrir plusieurs). Tout se passe comme si l'éditeur SQL travaillait non pas sur la base réelle mais sur une copie qui lui est propre. Dans la réalité, ce n'est pas exactement de cette façon que cela se passe mais cette image peut nous aider à comprendre la notion de transaction. Toutes les modifications apportées à la copie au cours d'une transaction ne seront visibles dans la base réelle que lorsqu'elles auront été validées par un [Commit Transaction]. La transaction courante est alors terminée et une nouvelle transaction commence.

Les modifications apportées au cours d'une transaction peuvent être annulées par une opération appelée [Rollback]. Faisons l'expérience suivante. Commençons une nouvelle transaction (il suffit de faire [Commit] sur la transaction courante) avec l'ordre SQL suivant :

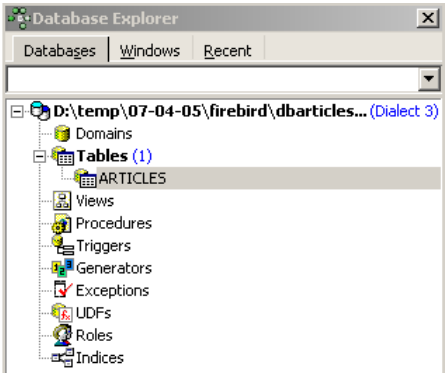


Exécutons cet ordre qui supprime toutes les lignes de la table [ARTICLES], puis exécutons [New Query] le nouvel ordre SQL suivant :

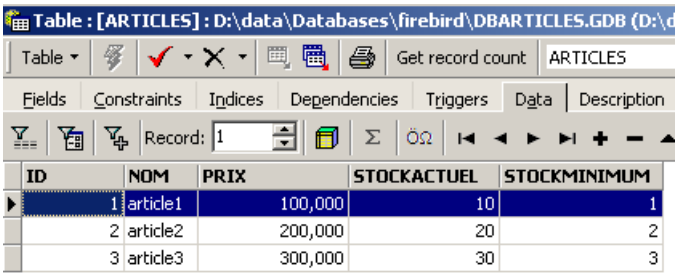
Nous obtenons le résultat suivant :



Toutes les lignes ont été détruites. Rappelons-nous que cela a été fait sur une copie de la table [ARTICLES]. Pour le vérifier, double-cliquons sur la table [ARTICLES] ci-dessous :

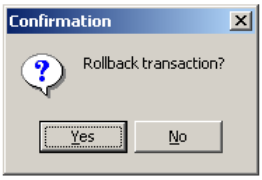


et visualisons l'onglet [Data] :

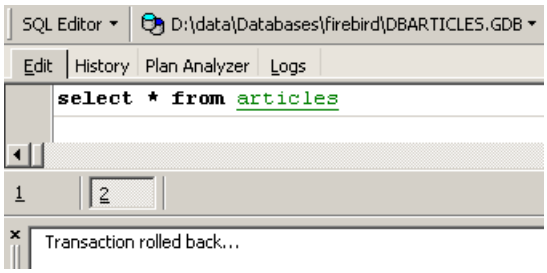


Même en utilisant le bouton [Refresh] ou en passant à l'onglet [Fields] pour revenir ensuite à l'onglet [Data], le contenu ci-dessus ne bouge pas. Ceci a été expliqué. Nous sommes dans une autre transaction qui travaille sur sa propre copie. Maintenant revenons à l'éditeur SQL (F12) et utilisons le bouton [RollBack] pour annuler les suppressions de lignes qui ont été faites :

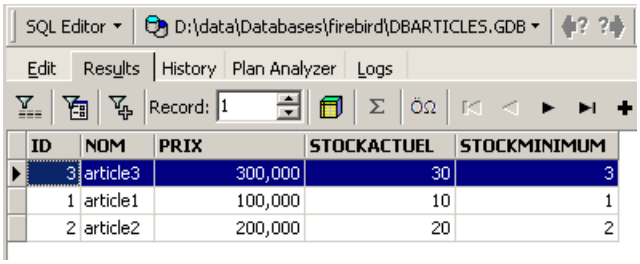
Confirmation nous est demandée :



Confirmons. L'éditeur SQL confirme que les modifications ont été annulées :



Rejouons la requête SQL ci-dessus pour vérifier. On retrouve les lignes qui avaient été supprimées :



ID	NOM	PRIX	STOCKACTUEL	STOCKMINIMUM
3	article3	300,000	30	3
1	article1	100,000	10	1
2	article2	200,000	20	2

L'opération [Rollback] a ramené la copie sur laquelle travaille l'éditeur SQL, dans l'état où elle était au début de la transaction.

3 Introduction au langage SQL

Dans cette section chapitre nous présentons les premières commandes SQL permettant de **créer** et **d'exploiter** une unique table. Nous en donnons en général une version simplifiée. Leur syntaxe complète est disponible dans les guides de référence de Firebird (cf paragraphe 2.2, page 6).

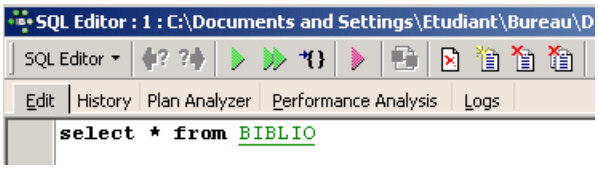
Une base de données est utilisée par des personnes ayant des compétences diverses :

- **l'administrateur** de la base est en général quelqu'un maîtrisant le langage SQL et les bases de données. C'est lui qui crée les tables car cette opération n'est en général faite qu'une fois. Il peut au cours du temps être amené à en modifier la structure. Une base de données est un ensemble de tables liées par des relations. C'est l'administrateur de la base qui définira ces relations. C'est également lui qui donnera des droits aux différents utilisateur de la base. Ainsi il indiquera que tel utilisateur a le droit de visualiser le contenu d'une table mais pas de la modifier.
- **l'utilisateur** de la base est quelqu'un qui fait vivre les données. Selon les droits accordés par l'administrateur de la base, il va ajouter, modifier, supprimer des données dans les différentes tables de la base. Il va aussi les exploiter pour en tirer des informations utiles à la bonne marche de l'entreprise, de l'administration, ...

Au paragraphe 2.6, page 13, nous avons présenté l'éditeur SQL de l'outil [IB-Expert]. C'est cet outil que nous allons utiliser. Rappelons quelques points :

- L'éditeur SQL s'obtient via l'option de menu [Tools/SQL Editor], soit via la touche [F12]

Nous obtenons alors une fenêtre [SQL Editor] dans laquelle nous pouvons taper un ordre SQL :



La copie d'écran ci-dessus sera souvent représentée par le texte ci-dessous :

```
SQL> select * from BIBLIO
```

3.1 Les types de données de Firebird

Lors de la création d'une table, il nous faut indiquer le type des données que peut contenir une colonne de table. Nous présentons ici, les types Firebird les plus courants. Signalons que ces types de données peuvent varier d'un SGBD à l'autre.

`SMALLINT`	nombre entier dans le domaine [-32768, 32767] : 4
`INTEGER`	nombre entier dans le domaine [–2 147 483 648, 2 147 483 647] : -100
`NUMERIC(n,m)` `DECIMAL(n,m)`	nombre réel de n chiffres dont m après la virgule NUMERIC(5,2) : -100.23, +027.30
`FLOAT`	nombre réel approché avec 7 chiffres significatifs : 10.4
`DOUBLE PRECISION`	nombre réel approché avec 15 chiffres significatifs : -100.89
`CHAR(N)` `CHARACTER(N)`	chaîne de N caractères exactement. Si la chaîne stockée a moins de N caractères, elle est complétée avec des espaces. CHAR(10) : 'ANGERS ' (4 espaces de fin)
`VARCHAR(N)` `CHARACTER VARYING(N)`	chaîne d'au plus N caractères

	VARCHAR(10) : 'ANGERS'
DATE	une date : '2006-01-09' (format YYYY-MM-DD)
TIME	une heure : '16:43:00' (format HH:MM:SS)
TIMESTAMP	date et heure à la fois : '2006-01-09 16:43:00' (format YYYY-MM-DD HH:MM:SS)

La fonction **CAST()** permet de passer d'un type à l'autre lorsque c'est nécessaire. Pour passer une valeur V déclarée comme étant de type T1 à un type T2, on écrit : CAST(V,T2). On peut opérer les changements de type suivants :

- **nombre** vers **chaîne de caractères**. Ce changement de type se fait implicitement et ne nécessite pas l'utilisation de la fonction CAST. Ainsi l'opération 1 + '3' ne nécessite pas de conversion du caractère '3'. Son résultat est le nombre 4.
- **DATE**, **TIME**, **TIMESTAMP** vers **chaînes de caractères** et vice-versa. Ainsi
- **TIMESTAMP** vers **TIME** ou **DATE** et vice-versa

Dans une table, une ligne peut avoir des colonnes sans valeur. On dit que la valeur de la colonne est la constante NULL. On peut tester la présence de cette valeur à l'aide des opérateurs

IS NULL / IS NOT NULL

3.2 Création d'une table

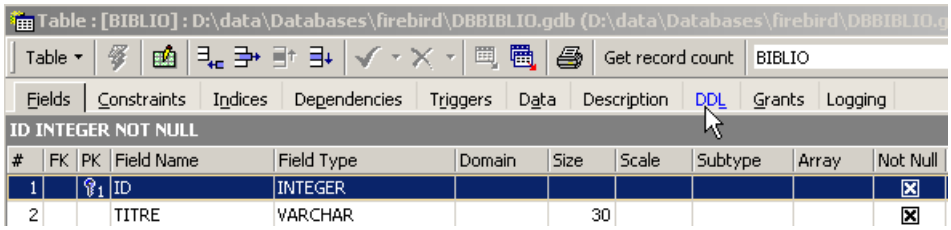
Pour découvrir comment créer une table, nous commençons par en créer une en mode [Design] avec IBExpert. Nous suivons pour cela la méthode décrite au paragraphe 2.3, page 8. Nous créons ainsi la table suivante :

#	FK	PK	Field Name	Field Type	Domain	Size	Scale	Subtype	Array	Not Null	Charset
1		1	ID	INTEGER						☒	
2			TITRE	VARCHAR		30				☒	ISO8859_1
3			AUTEUR	VARCHAR		20				☒	ISO8859_1
4			GENRE	VARCHAR		30				☒	ISO8859_1
5			ACHAT	DATE						☒	
6			PRIX	NUMERIC		6	2			☒	
7			DISPONIBLE	CHAR		1				☒	ISO8859_1

Cette table servira à enregistrer les livres achetés par une bibliothèque. La signification des champs est la suivante :

Name	Type	Contrainte	Signification
ID	INTEGER	Primary Key	Identifiant du livre
TITRE	VARCHAR(30)	NOT NULL UNIQUE	Titre du livre
AUTEUR	VARCHAR(20)	NOT NULL	Son auteur
GENRE	VARCHAR(30)	NOT NULL	Son genre (Roman, Poésie, Policier, BD, ..)
ACHAT	DATE	NOT NULL	Date d'achat du livre
PRIX	NUMERIC6,2)	NOT NULL	Son prix
DISPONIBLE	CHAR(1)	NOT NULL	Est-il disponible ? O (oui), N (non)

Cette table qui a été créée avec l'outil IBEXPERT comme assistant aurait pu être créée directement par des ordres SQL. Pour connaître ceux-ci, il suffit de consulter l'onglet [DDL] de la table :



Le code SQL qui a permis de créer la table [BIBLIO] est le suivant :

```
1.SET SQL DIALECT 3;
2.
3.SET NAMES ISO8859_1;
4.
5.
6.CREATE TABLE BIBLIO (
7.    ID INTEGER NOT NULL,
```

```
8.    TITRE VARCHAR(30) NOT NULL,
9.    AUTEUR VARCHAR(20) NOT NULL,
10.   GENRE VARCHAR(30) NOT NULL,
11.   ACHAT DATE NOT NULL,
12.   PRIX NUMERIC(6,2) NOT NULL,
13.   DISPONIBLE  CHAR(1) NOT NULL
14.);
15.
16.ALTER TABLE BIBLIO ADD CONSTRAINT UNQ1_BIBLIO UNIQUE (TITRE);
17.ALTER TABLE BIBLIO ADD CONSTRAINT PK_BIBLIO PRIMARY KEY (ID);
```

- ligne 1 : propriétaire Firebird - indique le niveau de dialecte SQL utilisé
- ligne 2 : propriétaire Firebird - indique la famille de caractères utilisée
- lignes 6 - 14 : standard SQL : crée la table BIBLIO en définissant le nom et la nature de chacune de ses colonnes.
- ligne 16 : standard SQL : crée une contrainte indiquant que la colonne TITRE n'admet pas de doublons
- ligne 17 : standard SQL : indique que la colonne [ID] est clé primaire de la table. Cela signifie que deux lignes de la table ne peuvent avoir le même ID. On est proche ici de la contrainte [UNIQUE NOT NULL] de la colonne [TITRE] et de fait la colonne TITRE aurait pu servir de clé primaire. La tendance actuelle est d'utiliser des clés primaires qui n'ont pas de signification et qui sont générées par le SGBD.

La syntaxe de la commande [CREATE TABLE] est la suivante :

`syntaxe` **CREATE TABLE** *table (nom_colonne1 type_colonne1 contrainte_colonne1, nom_colonne2 type_colonne2 contrainte_colonne2, ..., nom_colonnen type_colonnen contrainte_colonnen, autres contraintes)*

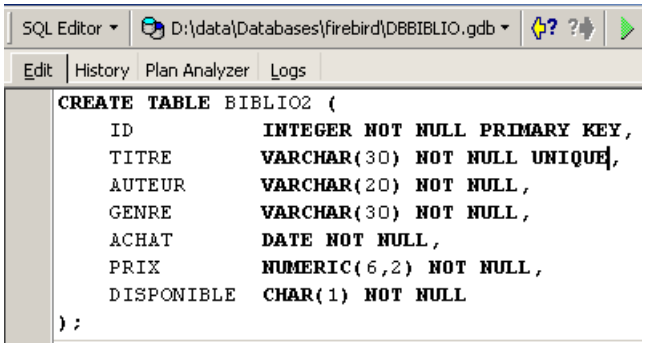
`action` crée la table *table* avec les colonnes indiquées

`nom_colonnei`	nom de la colonne i à créer
`type_colonnei`	type des données de la colonne i : char(30) numeric(6,2) date timestamp ...
`contrainte_colonnei`	contrainte que doivent respecter les données de la colonne i. En voici quelques unes : **PRIMARY KEY** : la colonne est clé **primaire**. Cela signifie que deux lignes de la table n'ont jamais la même valeur dans cette colonne et par ailleurs qu'une valeur est obligatoire dans cette colonne. Une clé primaire sert principalement à identifier une ligne de façon unique. **NOT NULL** : aucune valeur nulle n'est permise dans la colonne. **UNIQUE** : aucune valeur ne peut apparaître plusieurs fois dans la colonne. **CHECK (condition)** : la valeur de la colonne doit vérifier **condition**.
`autres contraintes`	on peut placer ici - des contraintes sur plusieurs colonnes : check(col1>col2) - des contraintes de clés étrangères

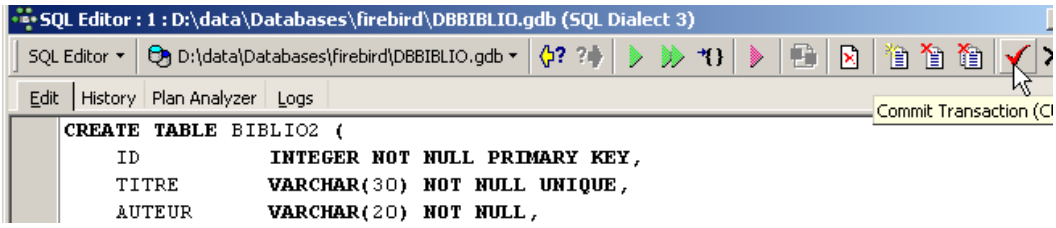
La table [BIBLIO] aurait pu également être construite avec l'ordre SQL suivant :

```
1.  CREATE TABLE BIBLIO (
2.      ID INTEGER NOT NULL PRIMARY KEY,
3.      TITRE VARCHAR(30) NOT NULL UNIQUE,
4.      AUTEUR VARCHAR(20) NOT NULL,
5.     GENRE VARCHAR(30) NOT NULL,
6.     ACHAT DATE NOT NULL,
7.     PRIX NUMERIC(6,2) NOT NULL,
8.     DISPONIBLE  CHAR(1) NOT NULL
9.  );
```

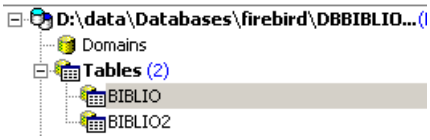
Montrons-le. Reprenons cet ordre dans un éditeur SQL (F12) pour créer une table que nous appellerons [BIBLIO2] :



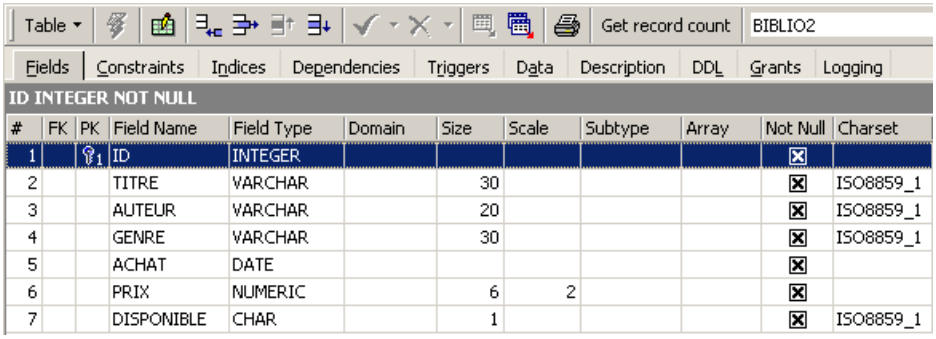
Après exécution, il faut valider la transaction afin de voir le résultat dans la base :



Ceci fait, la table apparaît dans la base :



En double-cliquant sur son nom, on peut avoir accès à sa structure :



#	FK	PK	Field Name	Field Type	Domain	Size	Scale	Subtype	Array	Not Null	Charset
1		1	ID	INTEGER						☒	
2			TITRE	VARCHAR		30				☒	ISO8859_1
3			AUTEUR	VARCHAR		20				☒	ISO8859_1
4			GENRE	VARCHAR		30				☒	ISO8859_1
5			ACHAT	DATE						☒	
6			PRIX	NUMERIC		6	2			☒	
7			DISPONIBLE	CHAR		1				☒	ISO8859_1

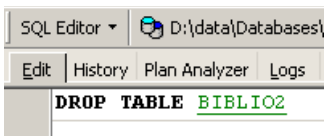
On retrouve bien la définition que nous avons faite de la table [BIBLIO2]

3.3 Suppression d'une table

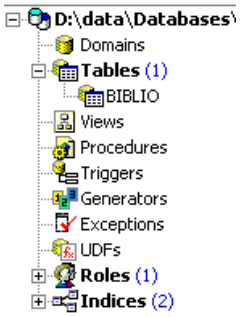
L'ordre SQL pour supprimer une table est le suivant :

`syntaxe` **DROP TABLE** table
`action` Supprime [table]

Pour supprimer la table [BIBLIO2] que nous venons de créer, nous exécutons maintenant la commande SQL suivante :

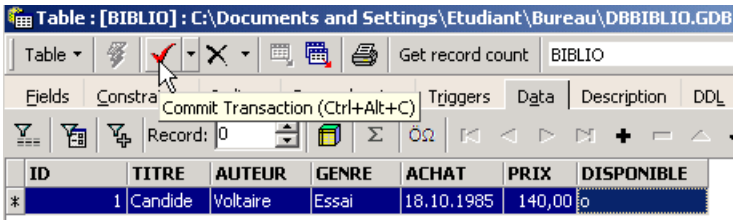


et nous la validons par [Commit]. La table [BIBLIO2] est supprimée :

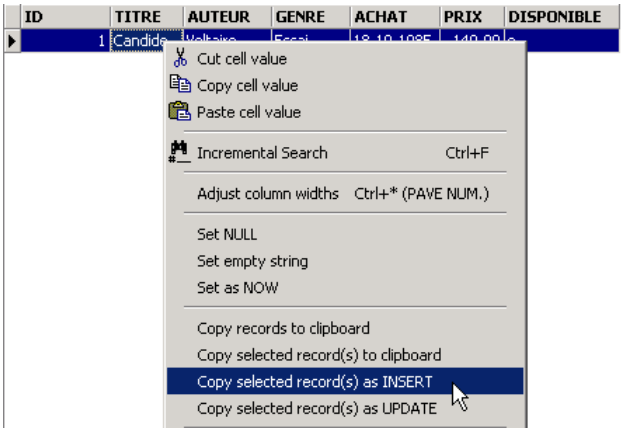


3.4 Remplissage d'une table

Insérons une ligne dans la table [BIBLIO] que nous venons de créer :



Validons l'ajout de la ligne par [Commit] puis cliquons droit sur la ligne ajoutée :



et demandons, comme il est montré ci-dessus, la copie de la ligne insérée dans le presse-papiers sous la forme d'un ordre SQL INSERT. Prenons ensuite n'importe quel éditeur de texte et collons (Coller / Paste) ce que nous venons de copier. Nous obtenons le code SQL suivant :

```
INSERT INTO BIBLIO (ID,TITRE,AUTEUR,GENRE,ACHAT,PRIX,DISPONIBLE) VALUES
(1,'Candide','Voltaire','Essai','18-OCT-1985',140,'o');
```

La syntaxe d'un ordre SQL **insert** est la suivante :

`syntaxe` **insert into** *table [(colonne1, colonne2, ..)]* **values** *(valeur1, valeur2, ...)*

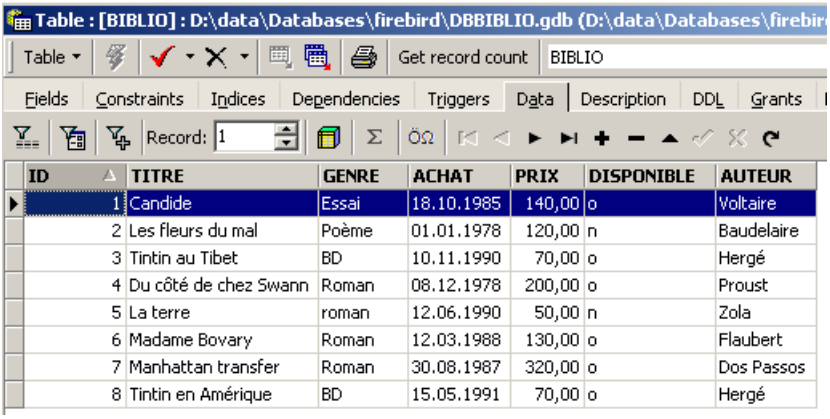
`action` ajoute une ligne *(valeur1, valeur2, ..)* à *table*. Ces valeurs sont affectées à *colonne1*, *colonne2*,... si elles sont présentes, sinon aux colonnes de la table dans l'ordre où elles ont été définies.

Pour insérer de nouvelles lignes dans la table [BIBLIO], on tapera les ordres INSERT suivants dans l'éditeur SQL. On exécutera et on validera [Commit] ces ordres un par un. On utilisera le bouton [New Query] pour passer à l'ordre INSERT suivant.

```
1.insert into biblio(id,titre,auteur,genre,achat,prix,disponible) values (2,'Les fleurs du
mal','Baudelaire','Poème','01-jan-78',120,'n');
2.insert into biblio(id,titre,auteur,genre,achat,prix,disponible) values (3,'Tintin au
Tibet','Hergé','BD','10-nov-90',70,'o');
3.insert into biblio(id,titre,auteur,genre,achat,prix,disponible) values (4,'Du côté de chez
Swann','Proust','Roman','08-dec-78',200,'o');
```

```
4.insert into biblio(id,titre,auteur,genre,achat,prix,disponible) values (5,'La
terre','Zola','roman','12-jun-90',50,'n');
5.insert into biblio(id,titre,auteur,genre,achat,prix,disponible) values (6,'Madame
Bovary','Flaubert','Roman','12-mar-88',130,'o');
6.insert into biblio(id,titre,auteur,genre,achat,prix,disponible) values (7,'Manhattan transfer','Dos
Passos','Roman','30-aug-87',320,'o');
7.insert into biblio(id,titre,auteur,genre,achat,prix,disponible) values (8,'Tintin en
Amérique','Hergé','BD','15-may-91',70,'o');
```

Après avoir validé [Commit] les différents ordres SQL, nous obtenons la table suivante :



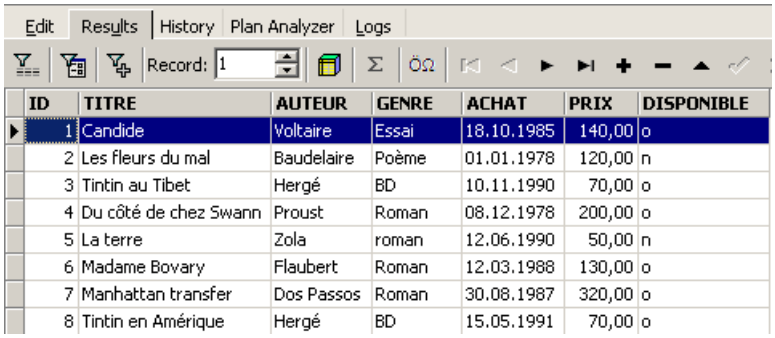
ID	TITRE	GENRE	ACHAT	PRIX	DISPONIBLE	AUTEUR
1	Candide	Essai	18.10.1985	140,00	o	Voltaire
2	Les fleurs du mal	Poème	01.01.1978	120,00	n	Baudelaire
3	Tintin au Tibet	BD	10.11.1990	70,00	o	Hergé
4	Du côté de chez Swann	Roman	08.12.1978	200,00	o	Proust
5	La terre	roman	12.06.1990	50,00	n	Zola
6	Madame Bovary	Roman	12.03.1988	130,00	o	Flaubert
7	Manhattan transfer	Roman	30.08.1987	320,00	o	Dos Passos
8	Tintin en Amérique	BD	15.05.1991	70,00	o	Hergé

3.5 Consultation d'une table

3.5.1 Introduction

Dans l'éditeur SQL, tapons la commande suivante :

et exécutons-la. Nous obtenons le résultat suivant :



ID	TITRE	AUTEUR	GENRE	ACHAT	PRIX	DISPONIBLE
1	Candide	Voltaire	Essai	18.10.1985	140,00	o
2	Les fleurs du mal	Baudelaire	Poème	01.01.1978	120,00	n
3	Tintin au Tibet	Hergé	BD	10.11.1990	70,00	o
4	Du côté de chez Swann	Proust	Roman	08.12.1978	200,00	o
5	La terre	Zola	roman	12.06.1990	50,00	n
6	Madame Bovary	Flaubert	Roman	12.03.1988	130,00	o
7	Manhattan transfer	Dos Passos	Roman	30.08.1987	320,00	o
8	Tintin en Amérique	Hergé	BD	15.05.1991	70,00	o

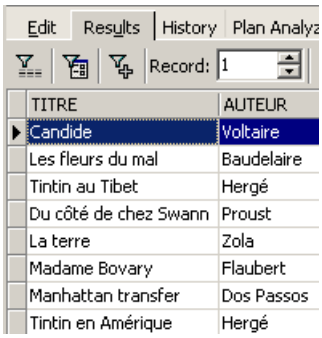
La commande **SELECT** permet de consulter le contenu de tables de la base de données. Cette commande a une syntaxe très riche. Nous ne présentons ici celle permettant d'interroger une unique table. Nous aborderons ultérieurement l'interrogation simultanée de plusieurs tables. La syntaxe de l'ordre SQL [SELECT] est la suivante :

`syntaxe` **SELECT [ALL|DISTINCT]** *[*|expression1 alias1, expression2 alias2, ...]*
FROM *table*

`action` affiche les valeurs de *expressioni* pour toutes les lignes de table. *expressioni* peut être une colonne ou une expression plus complexe. Le symbole * désigne l'ensemble des colonnes. Par défaut, toutes les lignes de table (**ALL**) sont affichées. Si **DISTINCT** est présent, les lignes identiques sélectionnées ne sont affichées qu'une fois. Les valeurs de *expressioni* sont affichées dans une colonne ayant pour titre *expressioni* ou *aliasi* si celui-ci a été utilisé.

Exemples :

```
SQL > select titre, auteur from biblio
```



TITRE	AUTEUR
Candide	Voltaire
Les fleurs du mal	Baudelaire
Tintin au Tibet	Hergé
Du côté de chez Swann	Proust
La terre	Zola
Madame Bovary	Flaubert
Manhattan transfer	Dos Passos
Tintin en Amérique	Hergé

```
SQL> select titre,prix from biblio
```

TITRE	PRIX
Candide	140,00
Les fleurs du mal	120,00
Tintin au Tibet	70,00
Du côté de chez Swann	200,00
La terre	50,00
Madame Bovary	130,00
Manhattan transfer	320,00
Tintin en Amérique	70,00

```
SQL> select titre TITRE_DU_LIVRE, prix PRIX_ACHAT from biblio
```

TITRE_DU_LIVRE	PRIX_ACHAT
Candide	140,00
Les fleurs du mal	120,00
Tintin au Tibet	70,00
Du côté de chez Swann	200,00
La terre	50,00
Madame Bovary	130,00
Manhattan transfer	320,00
Tintin en Amérique	70,00

Ci-dessus, nous avons associé des alias (TITRE_DU_LIVRE, PRIX_ACHAT) aux colonnes demandées.

3.5.2 Affichage des lignes vérifiant une condition

`syntaxe` **SELECT** ....
WHERE *condition*
`action` seules les lignes vérifiant la *condition* sont affichées

Exemples

```
SQL> select titre,prix from biblio where prix>100
```

TITRE	PRIX
Candide	140,00
Les fleurs du mal	120,00
Du côté de chez Swann	200,00
Madame Bovary	130,00
Manhattan transfer	320,00

```
SQL> select titre,prix,genre from biblio where genre='Roman'
```

TITRE	PRIX	GENRE
Du côté de chez Swann	200,00	Roman
Madame Bovary	130,00	Roman
Manhattan transfer	320,00	Roman

Un des livres a le genre 'roman' et non 'Roman'. Nous utilisons la fonction *upper* qui transforme une chaîne de caractères en majuscules pour avoir tous les romans.

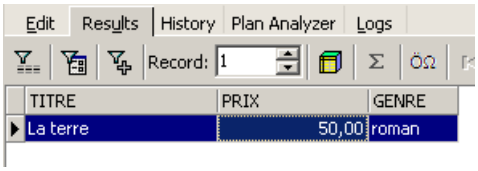
```
SQL> select titre,prix,genre from biblio where upper(genre)='ROMAN'
```

TITRE	PRIX	GENRE
Du côté de chez Swann	200,00	Roman
La terre	50,00	roman
Madame Bovary	130,00	Roman
Manhattan transfer	320,00	Roman

Nous pouvons réunir des conditions par les opérateurs logiques

`AND` ET logique
`OR` OU logique
`NOT` Négation logique

```
SQL> select titre,prix,genre from biblio where upper(genre)='ROMAN' and prix<100
```




```
SQL> select titre,genre from biblio
```

TITRE	GENRE
Candide	Essai
Les fleurs du mal	Poème
Tintin au Tibet	BD
Du côté de chez Swann	Roman
La terre	roman
Madame Bovary	Roman
Manhattan transfer	Roman
Tintin en Amérique	BD

```
SQL> select titre,genre from biblio where upper(genre)='ROMAN' or upper(genre)='BD'
```

TITRE	GENRE
Tintin au Tibet	BD
Du côté de chez Swann	Roman
La terre	roman
Madame Bovary	Roman
Manhattan transfer	Roman
Tintin en Amérique	BD

```
SQL> select titre,genre from biblio where not( upper(genre)='ROMAN' or upper(genre)='BD')
```

TITRE	GENRE
Candide	Essai
Les fleurs du mal	Poème

```
SQL> select titre,achat from biblio
```

TITRE	ACHAT
Candide	18.10.1985
Les fleurs du mal	01.01.1978
Tintin au Tibet	10.11.1990
Du côté de chez Swann	08.12.1978
La terre	12.06.1990
Madame Bovary	12.03.1988
Manhattan transfer	30.08.1987
Tintin en Amérique	15.05.1991

```
SQL>select titre,achat from biblio where achat>'31-dec-1987'
```

TITRE	ACHAT
Tintin au Tibet	10.11.1990
La terre	12.06.1990
Madame Bovary	12.03.1988
Tintin en Amérique	15.05.1991

```
SQL> select titre,prix from biblio where prix between 100 and 150
```

TITRE	PRIX
Candide	140,00
Les fleurs du mal	120,00
Madame Bovary	130,00

3.5.3 Affichage des lignes selon un ordre déterminé

Aux syntaxes précédentes, il est possible d'ajouter une clause **ORDER BY** indiquant l'ordre d'affichage désiré :

`syntaxe` **SELECT** ....
ORDER BY *expression1* [**asc**|**desc**], *expression2* [**asc**|**dec**], ...

`action` Les lignes résultat de la sélection sont affichées dans l'ordre de
1 : ordre croissant (**asc** / **ascending** qui est la valeur par défaut) ou décroissant (**desc / descending**) de *expression1*
2 : en cas d'égalité de *expression1*, l'affichage se fait selon les valeurs de *expression2*
etc ..

Exemples :

```
SQL>select titre, genre,prix,achat from biblio order by achat desc
```

TITRE	GENRE	PRIX	ACHAT
Tintin en Amérique	BD	70,00	15.05.1991
Tintin au Tibet	BD	70,00	10.11.1990
La terre	roman	50,00	12.06.1990
Madame Bovary	Roman	130,00	12.03.1988
Manhattan transfer	Roman	320,00	30.08.1987
Candide	Essai	140,00	18.10.1985
Du côté de chez Swann	Roman	200,00	08.12.1978
Les fleurs du mal	Poème	120,00	01.01.1978

```
SQL>select titre, genre,prix,achat from biblio order by prix
```

TITRE	GENRE	PRIX	ACHAT
La terre	roman	50,00	12.06.1990
Tintin au Tibet	BD	70,00	10.11.1990
Tintin en Amérique	BD	70,00	15.05.1991
Les fleurs du mal	Poème	120,00	01.01.1978
Madame Bovary	Roman	130,00	12.03.1988
Candide	Essai	140,00	18.10.1985
Du côté de chez Swann	Roman	200,00	08.12.1978
Manhattan transfer	Roman	320,00	30.08.1987

```
SQL>select titre, genre,prix,achat from biblio order by genre desc
```

TITRE	GENRE	PRIX	ACHAT
La terre	roman	50,00	12.06.1990
Madame Bovary	Roman	130,00	12.03.1988
Du côté de chez Swann	Roman	200,00	08.12.1978
Manhattan transfer	Roman	320,00	30.08.1987
Les fleurs du mal	Poème	120,00	01.01.1978
Candide	Essai	140,00	18.10.1985
Tintin au Tibet	BD	70,00	10.11.1990
Tintin en Amérique	BD	70,00	15.05.1991

```
SQL >select titre, genre,prix,achat from biblio order by genre desc, prix
```

TITRE	GENRE	PRIX	ACHAT
La terre	roman	50,00	12.06.1990
Madame Bovary	Roman	130,00	12.03.1988
Du côté de chez Swann	Roman	200,00	08.12.1978
Manhattan transfer	Roman	320,00	30.08.1987
Les fleurs du mal	Poème	120,00	01.01.1978
Candide	Essai	140,00	18.10.1985
Tintin au Tibet	BD	70,00	10.11.1990
Tintin en Amérique	BD	70,00	15.05.1991

```
SQL>select titre, genre,prix,achat from biblio order by genre desc, prix desc
```

TITRE	GENRE	PRIX	ACHAT
La terre	roman	50,00	12.06.1990
Manhattan transfer	Roman	320,00	30.08.1987
Du côté de chez Swann	Roman	200,00	08.12.1978
Madame Bovary	Roman	130,00	12.03.1988
Les fleurs du mal	Poème	120,00	01.01.1978
Candide	Essai	140,00	18.10.1985
Tintin au Tibet	BD	70,00	10.11.1990
Tintin en Amérique	BD	70,00	15.05.1991

3.6 Suppression de lignes dans une table

`syntaxe` **DELETE FROM** *table* [**WHERE** *condition*]
`action` supprime les lignes de *table* vérifiant *condition*. Si cette dernière est absente, toutes les lignes sont détruites.

Exemples :

```
SQL> select titre from biblio
```

TITRE
Candide
Les fleurs du mal
Tintin au Tibet
Du côté de chez Swann
La terre
Madame Bovary
Manhattan transfer
Tintin en Amérique

Les deux commandes ci-dessous sont émises l'une après l'autre :

```
SQL> delete from biblio where titre='Candide'
```

```
SQL> select titre from biblio
```

TITRE
Les fleurs du mal
Tintin au Tibet
Du côté de chez Swann
La terre
Madame Bovary
Manhattan transfer
Tintin en Amérique

3.7 Modification du contenu d'une table

`syntaxe` **update table set** *colonne1 = expression1, colonne2 = expression2, ...*
[**where** *condition*]

`action` Pour les lignes de *table* vérifiant *condition* (toutes les lignes s'il n'y a pas de condition), *colonnei* reçoit la valeur *expressioni*.

Exemples :

```
SQL> select genre from biblio
```

GENRE
Poème
BD
Roman
roman
Roman
Roman
BD

On met tous les genres en majuscules :

```
SQL> update biblio set genre=upper(genre)
```

On vérifie :

```
SQL> select genre from biblio
```

GENRE
POèME
BD
ROMAN
ROMAN
ROMAN
ROMAN
BD

On affiche les prix :

```
SQL> select genre,prix from biblio;
```

GENRE	PRIX
POèME	120,00
BD	70,00
ROMAN	200,00
ROMAN	50,00
ROMAN	130,00
ROMAN	320,00
BD	70,00

Le prix des romans augmente de 5% :

```
SQL> update biblio set prix=prix*1.05 where genre='ROMAN';
```

On vérifie :

```
SQL> select genre,prix from biblio
```

GENRE	PRIX
POèME	120,00
BD	70,00
ROMAN	210,00
ROMAN	52,50
ROMAN	136,50
ROMAN	336,00
BD	70,00

3.8 Mise à jour définitive d'une table

Lorsqu'on apporte des modifications à une table, Firebird les génère en fait sur une copie de la table. Elles peuvent être alors rendues définitives ou bien être annulées par les commandes **COMMIT** et **ROLLBACK**.

`syntaxe` COMMIT
`action` rend définitives les mises à jour faites sur les tables depuis le dernier COMMIT.

`syntaxe` ROLLBACK
`action` annule toutes modifications faites sur les tables depuis le dernier COMMIT.

`Remarque` Un COMMIT est fait implicitement aux moments suivants :
a) A la déconnexion de Firebird
b) Après chaque commande affectant la structure des tables : CREATE, ALTER, DROP.

Exemples

Dans l'éditeur SQL, on met la base dans un état connu en validant toutes les opérations faites depuis le dernier COMMIT ou ROLLBACK :

```
SQL> commit
```

On demande la liste des titres :

```
SQL> select titre from biblio
```

TITRE
Les fleurs du mal
Tintin au Tibet
Du côté de chez Swann
La terre
Madame Bovary
Manhattan transfer
Tintin en Amérique

Suppression d'un titre :

```
SQL> delete from biblio where titre='La terre'
```

Vérification :

```
SQL> select titre from biblio
```

TITRE
Les fleurs du mal
Tintin au Tibet
Du côté de chez Swann
Madame Bovary
Manhattan transfer
Tintin en Amérique

Le titre a bien été supprimé. Maintenant nous invalidons toutes les modifications faites depuis le dernier COMMIT / ROLLBACK :

```
SQL> rollback
```

Vérification :

```
SQL> select titre from biblio
```

TITRE
Les fleurs du mal
Tintin au Tibet
Du côté de chez Swann
La terre
Madame Bovary
Manhattan transfer
Tintin en Amérique

On retrouve le titre supprimé. Demandons maintenant la liste des prix :

```
SQL> select prix from biblio
```

PRIX
120,00
70,00
210,00
52,50
136,50
336,00
70,00

Mettons tous les prix sont mis à zéro.

```
SQL> update biblio set prix=0
```

Vérifions les prix :

```
SQL> select prix from biblio
```

PRIX
0,00
0,00
0,00
0,00
0,00
0,00
0,00

Supprimons les modifications faites sur la base :

```
SQL> rollback
```

et vérifions de nouveau les prix :

```
SQL> select prix from biblio
```

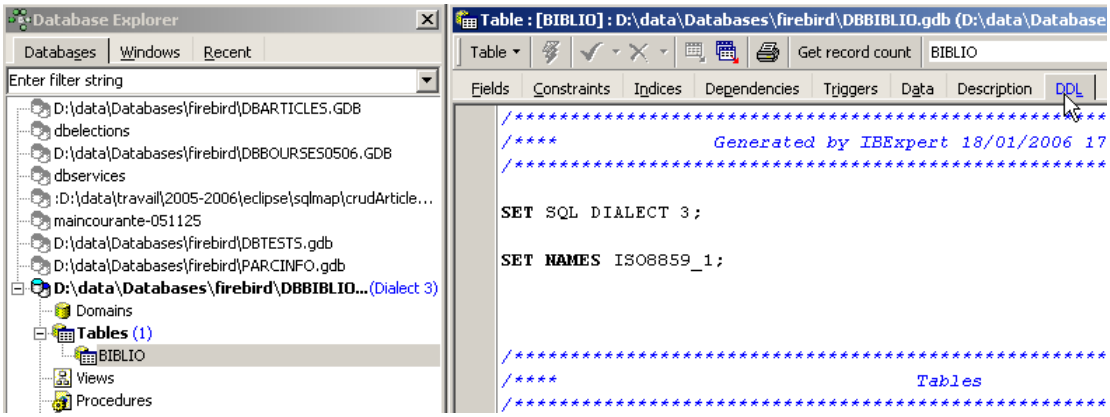
PRIX
120,00
70,00
210,00
52,50
136,50
336,00
70,00

Nous avons retrouvé les prix primitifs.

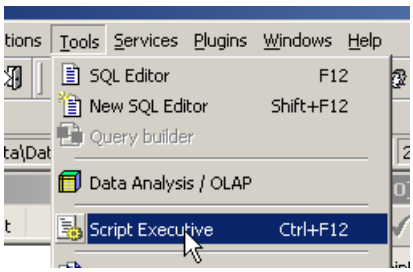
3.9 Ajout de lignes dans une table en provenance d'une autre table

Il est possible d'ajouter des lignes d'une table à une autre table lorsque leurs structures sont compatibles. Pour le montrer, commençons par créer une table [BIBLIO2] ayant la même structure que [BIBLIO].

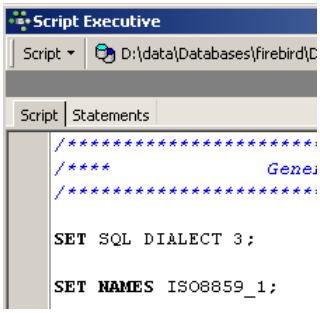
Dans l'explorateur de bases d'IBExpert, double-cliquons sur la table [BIBLIO] pour avoir accès à l'onglet [DDL] :



Dans cet onglet, on trouve la liste des ordres SQL qui permettent de générer la table [BIBLIO]. Copions la totalité de ce code dans le presse-papiers (CTRL-A, CTRL-C). Puis appelons un outil appelé [Script Executive] permettant d'exécuter une liste d'ordres SQL :



On obtient un éditeur de texte, dans lequel nous pouvons coller (CTRL-V) le texte mis précédemment dans le presse-papiers :

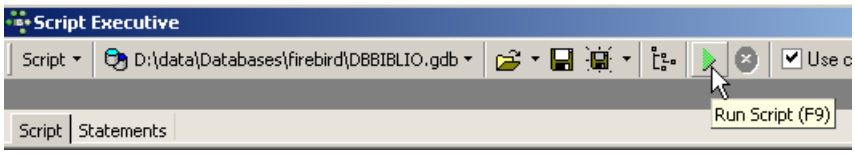


On appelle souvent **script SQL** une liste d'ordres SQL. [Script Executive] va nous permettre d'exécuter un tel script alors que l'éditeur SQL ne permettait l'exécution que d'un unique ordre à la fois. Le script SQL actuel permet de créer la table [BIBLIO]. Faisons en sorte qu'il crée une table appelée [BIBLIO2]. Il suffit pour cela de changer [BIBLIO] en [BIBLIO2] :

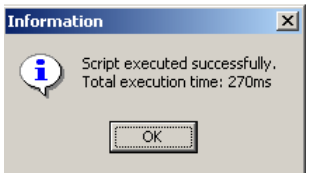
```
1.SET SQL DIALECT 3;
2.
3.SET NAMES ISO8859_1;
4.
5.CREATE TABLE BIBLIO2 (
6.    ID          INTEGER NOT NULL,
7.    TITRE       VARCHAR(30) NOT NULL,
8.    AUTEUR      VARCHAR(20) NOT NULL,
9.    GENRE       VARCHAR(20) NOT NULL,
10.    ACHAT       DATE NOT NULL,
11.    PRIX        NUMERIC(6,2) DEFAULT 10 NOT NULL,
12.    DISPONIBLE  CHAR(1) NOT NULL
13.);
```

```
14.
15.ALTER TABLE BIBLIO2 ADD CONSTRAINT UNQ1_BIBLIIO2 UNIQUE (TITRE);
16.
17.ALTER TABLE BIBLIO2 ADD CONSTRAINT PK_BIBLIIO2 PRIMARY KEY (ID);
```

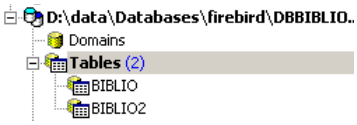
Exécutons ce script avec le bouton [Run Script] ci-dessous :



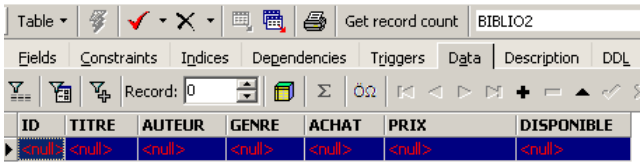
Le script est exécuté :



et on peut voir la nouvelle table dans l'explorateur de bases :



Si on double-clique sur [BIBLIO2] pour vérifier son contenu, on découvre qu'elle est vide, ce qui est normal :

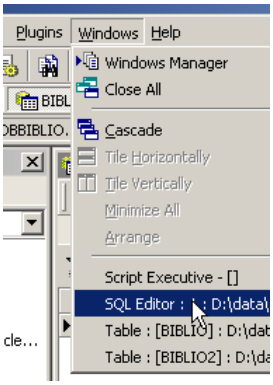


Une variante de l'ordre SQL INSERT permet d'insérer dans une table, des lignes provenant d'une autre table :

`syntaxe` **INSERT INTO** *table1* [(*colonne1, colonne2*, ...)]
SELECT *colonnea, colonneb*, ... **FROM** *table2* **WHERE** *condition*

`action` Les lignes de *table2* vérifiant *condition* sont ajoutées à *table1*. Les colonnes *colonnea, colonneb*, ... de *table2* sont affectées dans l'ordre à *colonne1, colonne2*, ... de *table1* et doivent donc être de type compatible.

Revenons dans l'éditeur SQL :



et émettons l'ordre SQL suivant :

```
SQL> insert into BIBLIO2 select * from BIBLIO where upper(genre)='ROMAN'
```

qui insère dans [BIBLIO2] toutes les lignes de [BIBLIO] correspondant à un roman. Après exécution de l'ordre SQL, validons-le par un [Commit] :

```
SQL> commit
```

Ceci fait, consultons les données de la table [BIBLIO2] :

```
SQL> select * from BIBLIO2
```

ID	TITRE	AUTEUR	GENRE	ACHAT	PRIX	DISPONIBLE
4	Du côté de chez Swann	Proust	ROMAN	08.12.1978	210,00	o
5	La terre	Zola	ROMAN	12.06.1990	52,50	n
6	Madame Bovary	Flaubert	ROMAN	12.03.1988	136,50	o
7	Manhattan transfer	Dos Passos	ROMAN	30.08.1987	336,00	o

3.10 Suppression d'une table

`syntaxe` **DROP TABLE** *table*
`action` supprime *table*

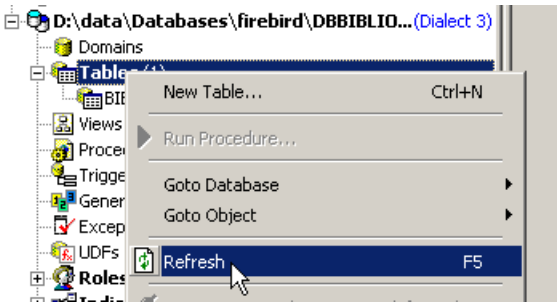
Exemple : on supprime la table BIBLIO2

```
SQL> drop table BIBLIO2
```

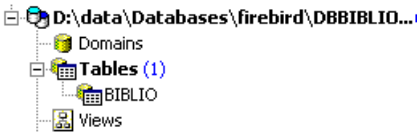
On valide le changement :

```
SQL> commit
```

Dans l'explorateur de bases, on rafraîchit l'affichage des tables :



On découvre que la table [BIBLIO2] a été supprimée :



3.11 Modification de la structure d'une table

`syntaxe` **ALTER TABLE** *table*
[**ADD** *nom_colonne1 type_colonne1 contrainte_colonne1*]
[**ALTER** *nom_colonne2* ***TYPE*** *type_colonne2*]
[**DROP** *nom_colonne3*]
[**ADD** *contrainte*]
[**DROP CONSTRAINT** *nom_contrainte*]
`action` permet d'ajouter (**ADD**) de modifier (**ALTER**) et de supprimer (**DROP)** des colonnes de table. La syntaxe *nom_colonnei type_colonnei contrainte_colonnei* est celle du CREATE TABLE. On peut également ajouter / supprimer des contraintes de table.

Exemple : Exécutons successivement les deux commandes SQL suivantes dans l'éditeur SQL

```
SQL > alter table biblio add nb_pages numeric(4), alter genre type varchar(30)
```

```
SQL> commit
```

Dans l'explorateur de bases, vérifions la structure de la table [BIBLIO] :

ID INTEGER NOT NULL

#	FK	PK	Field Name	Field Type	Domain	Size	Scale
1		🔑1	ID	INTEGER			
2			TITRE	VARCHAR		30	
3			AUTEUR	VARCHAR		20	
4			GENRE	VARCHAR		30	
5			ACHAT	DATE			
6			PRIX	NUMERIC		6	2
7			DISPONIBLE	CHAR		1	
8			NB_PAGES	NUMERIC		4	0

Les modifications ont été prises en compte. Voyons comment a évolué le contenu de la table :

```
SQL> select * from biblio
```

ID	TITRE	AUTEUR	GENRE	ACHAT	PRIX	DISPONIBLE	NB_PAGES
2	Les fleurs du mal	Baudelaire	POèME	01.01.1978	120,00	n	<null>
3	Tintin au Tibet	Hergé	BD	10.11.1990	70,00	o	<null>
4	Du côté de chez Swann	Proust	ROMAN	08.12.1978	210,00	o	<null>
5	La terre	Zola	ROMAN	12.06.1990	52,50	n	<null>
6	Madame Bovary	Flaubert	ROMAN	12.03.1988	136,50	o	<null>
7	Manhattan transfer	Dos Passos	ROMAN	30.08.1987	336,00	o	<null>
8	Tintin en Amérique	Hergé	BD	15.05.1991	70,00	o	<null>

La nouvelle colonne [NB_PAGES] a été créée mais n'a aucune valeur. Supprimons cette colonne :

```
SQL> alter table biblio drop nb_pages
```

```
SQL> commit
```

Vérifions la nouvelle structure de la table [BIBLIO] :

ID INTEGER NOT NULL

#	FK	PK	Field Name	Field Type	Domain	Size	Scale
1		🔑1	ID	INTEGER			
2			TITRE	VARCHAR		30	
3			AUTEUR	VARCHAR		20	
4			GENRE	VARCHAR		30	
5			ACHAT	DATE			
6			PRIX	NUMERIC		6	2
7			DISPONIBLE	CHAR		1	

La colonne [NB_PAGES] a bien disparu.

3.12 Les vues

Il est possible d'avoir une vue partielle d'une table ou de plusieurs tables. Une vue se comporte comme une table mais ne contient pas de données. Ses données sont extraites d'autres tables ou vues. Une vue comporte plusieurs avantages :

1. Un utilisateur peut n'être intéressé que par certaines colonnes et certaines lignes d'une table donnée. La vue lui permet de ne voir que ces lignes et ces colonnes.
2. Le propriétaire d'une table peut désirer n'en autoriser qu'un accès limité, à d'autres utilisateurs. La vue lui permet de le faire. Les utilisateurs qu'il aura autorisés n'auront accès qu'à la vue qu'il aura définie.

3.12.1 Création d'une vue

`syntaxe`	**CREATE VIEW** *nom_vue* **AS SELECT** *colonne1, colonne2, ...* **FROM** *table* **WHERE** *condition* [**WITH CHECK OPTION**]
`action`	crée la vue *nom_vue*. Celle-ci est une table ayant pour structure *colonne1, colonne2, ...* de *table* et pour lignes, les lignes de *table* vérifiant *condition* (toutes les lignes s'il n'y a pas de condition)
`WITH CHECK OPTION`	Cette clause optionnelle indique que les insertions et les mises à jour sur la vue, ne doivent pas créer de lignes que la vue ne pourrait sélectionner.

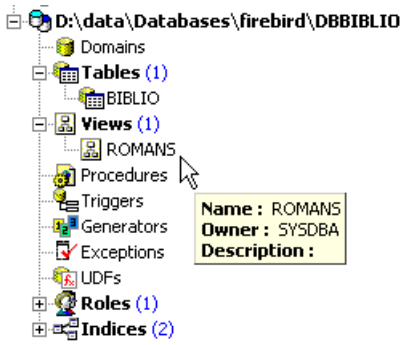
Remarque La syntaxe de CREATE VIEW est en fait plus complexe que celle présentée ci-dessus et permet notamment de créer une vue à partir de plusieurs tables. Il suffit pour cela que la requête SELECT porte sur plusieurs tables (cf chapitre suivant).

Exemples

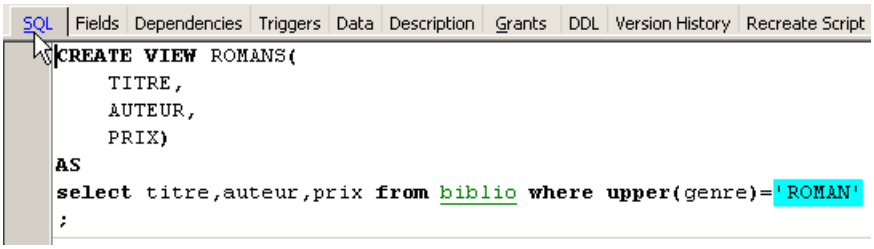
On crée à partir de la table **biblio**, une vue ne comportant que les romans (sélection de lignes) et que les colonnes **titre, auteur, prix** (sélection de colonnes) :

```
SQL> create view romans as select titre,auteur,prix from biblio where upper(genre)='ROMAN';

SQL> commit
```

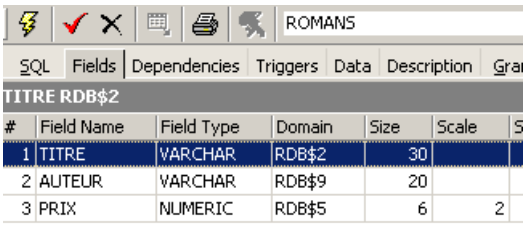
Dans l'explorateur de bases, rafraîchissons la vue (F5). On voit apparaître une vue :



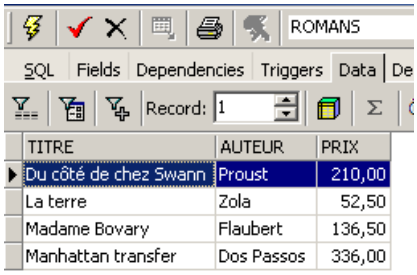
On peut connaître l'ordre SQL associé à la vue. Pour cela, double-cliquons sur la vue [ROMANS] :



Une vue est comme une table. Elle a une structure :



et un contenu :



TITRE	AUTEUR	PRIX
Du côté de chez Swann	Proust	210,00
La terre	Zola	52,50
Madame Bovary	Flaubert	136,50
Manhattan transfer	Dos Passos	336,00

Une vue s'utilise comme une table. On peut émettre des requêtes SQL dessus. Voici quelques exemples à jouer dans l'éditeur SQL :

```
SQL> select * from romans
```

TITRE	AUTEUR	PRIX
Du côté de chez Swann	Proust	210,00
La terre	Zola	52,50
Madame Bovary	Flaubert	136,50
Manhattan transfer	Dos Passos	336,00

```
SQL> insert into biblio values (10,'Le père Goriot','Balzac','Roman','01-sep-91',200,'o')
```

Le nouveau roman est-il visible dans la vue [ROMANS] ?

```
SQL> select * from romans
```

TITRE	AUTEUR	PRIX
Du côté de chez Swann	Proust	210,00
La terre	Zola	52,50
Madame Bovary	Flaubert	136,50
Manhattan transfer	Dos Passos	336,00
Le père Goriot	Balzac	200,00

Ajoutons autre chose qu'un roman à la table [BIBLIO] :

```
SQL> insert into biblio(id,titre,auteur,genre,achat,prix,disponible) values (11,'Poèmes
saturniens','Verlaine','Poème','02-sep-92',200,'o');
```

Vérifions la table [BIBLIO] :

```
SQL> select titre, auteur from BIBLIO
```

TITRE	AUTEUR
Les fleurs du mal	Baudelaire
Tintin au Tibet	Hergé
Du côté de chez Swann	Proust
La terre	Zola
Madame Bovary	Flaubert
Manhattan transfer	Dos Passos
Tintin en Amérique	Hergé
Le père Goriot	Balzac
Poèmes saturniens	Verlaine

Vérifions la vue [ROMANS] :

```
SQL> select titre, auteur from ROMANS
```

TITRE	AUTEUR
Du côté de chez Swann	Proust
La terre	Zola
Madame Bovary	Flaubert
Manhattan transfer	Dos Passos
Le père Goriot	Balzac

Le livre ajouté n'est pas dans la vue [ROMANS] parce qu'il n'avait pas upper(genre)='ROMAN'.

3.12.2 Mise à jour d'une vue

Il est possible de mettre à jour une vue comme on le fait pour une table. Toutes les tables d'où sont extraites les données de la vue sont affectées par cette mise à jour. Voici quelques exemples :

```
SQL> insert into biblio(id,titre,auteur,genre,achat,prix,disponible) values (13,'Le Rouge et le
Noir','Stendhal','Roman','03-oct-92',110,'o')
```

```
SQL> select * from romans
```

TITRE	AUTEUR	PRIX
Du côté de chez Swann	Proust	210,00
La terre	Zola	52,50
Madame Bovary	Flaubert	136,50
Manhattan transfer	Dos Passos	336,00
Le père Goriot	Balzac	200,00
Le Rouge et le Noir	Stendhal	110,00

```
SQL> select titre, auteur from biblio
```

TITRE	AUTEUR
Les fleurs du mal	Baudelaire
Tintin au Tibet	Hergé
Du côté de chez Swann	Proust
La terre	Zola
Madame Bovary	Flaubert
Manhattan transfer	Dos Passos
Tintin en Amérique	Hergé
Le père Goriot	Balzac
Poèmes saturniens	Verlaine
Le Rouge et le Noir	Stendhal

On supprime une ligne de la vue [ROMANS] :

```
SQL> delete from ROMANS where titre='Le Rouge et le Noir'
```

```
SQL> select * from romans
```

TITRE	AUTEUR	PRIX
Du côté de chez Swann	Proust	210,00
La terre	Zola	52,50
Madame Bovary	Flaubert	136,50
Manhattan transfer	Dos Passos	336,00
Le père Goriot	Balzac	200,00

```
SQL> select auteur, titre from BIBLIO
```

AUTEUR	TITRE
Baudelaire	Les fleurs du mal
Hergé	Tintin au Tibet
Proust	Du côté de chez Swann
Zola	La terre
Flaubert	Madame Bovary
Dos Passos	Manhattan transfer
Hergé	Tintin en Amérique
Balzac	Le père Goriot
Verlaine	Poèmes saturniens

La ligne supprimée de la vue [ROMANS] a été également supprimée dans la table [BIBLIO]. On augmente maintenant le prix des livres de la vue [ROMANS] :

```
SQL> update romans set prix=prix*1.05
```

On vérifie dans [ROMANS] :

```
SQL> select * from romans
```

TITRE	AUTEUR	PRIX
Du côté de chez Swann	Proust	220,50
La terre	Zola	55,13
Madame Bovary	Flaubert	143,33
Manhattan transfer	Dos Passos	352,80
Le père Goriot	Balzac	210,00

Quel a été l'impact sur la table [BIBLIO] ?

```
SQL> select titre, auteur, prix from biblio
```

TITRE	AUTEUR	PRIX
Les fleurs du mal	Baudelaire	120,00
Tintin au Tibet	Hergé	70,00
Du côté de chez Swann	Proust	220,50
La terre	Zola	55,13
Madame Bovary	Flaubert	143,33
Manhattan transfer	Dos Passos	352,80
Tintin en Amérique	Hergé	70,00
Le père Goriot	Balzac	210,00
Poèmes saturniens	Verlaine	200,00

Les romans ont bien été augmentés de 5% dans [BIBLIO] également.

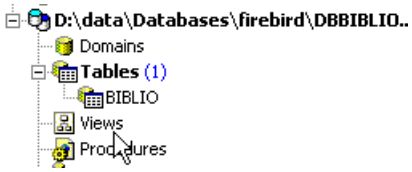
3.12.3 Supprimer une vue

`syntaxe` **DROP VIEW** nom_vue
`action` supprime la vue nommée

Exemple

```
SQL> drop view romans

SQL> commit
```

Dans l'explorateur de bases, on peut rafraîchir la vue (F5) pour constater que la vue [ROMANS] a disparu :



3.13 Utilisation de fonctions de groupes

Il existe des fonctions qui, au lieu de travailler sur chaque ligne d'une table, travaillent sur des groupes de lignes. Ce sont essentiellement des fonctions statistiques nous permettant d'avoir la moyenne, l'écart-type, etc ... des données d'une colonne.

`syntaxe1` **SELECT** *f1, f2, .., fn* **FROM** *table*
[**WHERE** *condition*]
`action` calcule les fonctions statistiques *fi* sur l'ensemble des lignes de table vérifiant l'éventuelle *condition.*

`syntaxe2` **SELECT** *f1, f2, .., fn* **FROM** *table*
[**WHERE** *condition*]
[**GROUP BY** *expr1*, *expr2*, ..]
`action` Le mot clé **GROUP BY** a pour effet de diviser les lignes de table en groupes. Chaque groupe contient les lignes pour lesquelles les expressions *expr1, expr2*, ... ont la même valeur.
Exemple : **GROUP BY genre** met dans un même groupe, les livres ayant le même genre. La clause **GROUP BY auteur,genre** mettrait dans le même groupe les livres ayant même auteur et même genre. Le **WHERE condition** élimine d'abord de la table les lignes ne vérifiant pas **condition**. Ensuite les groupes sont formés par la clause GROUP BY. Les fonctions *fi* sont ensuite calculées pour chaque groupe de lignes.

`syntaxe3` **SELECT** *f1, f2, .., fn* **FROM** *table*
[**WHERE** *condition*]
[**GROUP BY** *expression*]
[**HAVING** *condition_de*_groupe]

`action` La clause **HAVING** filtre les groupes formés par la clause **GROUP BY**. Elle est donc toujours liée à la présence de cette clause GROUP BY. Exemple : **GROUP BY** genre **HAVING** genre!='ROMAN'

Les fonctions statistiques *fi* disponibles sont les suivantes :

`AVG(expression)`	moyenne de expression
`COUNT(expression)`	nombre de lignes pour lesquelles expression a une valeur
`COUNT(*)`	nombre total de lignes dans la table
`MAX(expression)`	max de expression
`MIN(expression)`	min de expression
`SUM(expression)`	somme de expression

Exemples

```
SQL> select prix from biblio
```

PRIX
120,00
70,00
220,50
55,13
143,33
352,80
70,00
210,00
200,00

Prix moyen ? Prix maximal ? Prix minimal ?

```
SQL> select avg(prix), max(prix), min (prix) from biblio
```

AVG	MAX	MIN
160,19	352,80	55,13

```
SQL> select titre, prix,genre from biblio
```

TITRE	PRIX	GENRE
Les fleurs du mal	120,00	POèME
Tintin au Tibet	70,00	BD
Du côté de chez Swann	220,50	ROMAN
La terre	55,13	ROMAN
Madame Bovary	143,33	ROMAN
Manhattan transfer	352,80	ROMAN
Tintin en Amérique	70,00	BD
Le père Goriot	210,00	Roman
Poèmes saturniens	200,00	Poème

Prix moyen d'un roman ? Prix maximal ?

```
SQL> select avg(prix) moyenne, max(prix) prix_maxi from biblio where upper(genre)='ROMAN'
```

MOYENNE	PRIX_MAXI
196,35	352,80

Combien de BD ?

```
SQL> select count(*) from biblio where upper(genre)='BD'
```

COUNT
2

Combien de romans à moins de 100 F ?

```
SQL> select count(*) from biblio where upper(genre)='ROMAN' and prix<100
```

COUNT
1

```
SQL> select genre, prix from biblio
```

UPPER	PRIX
POèME	120,00
BD	70,00
ROMAN	220,50
ROMAN	55,13
ROMAN	143,33
ROMAN	352,80
BD	70,00
ROMAN	210,00
POèME	200,00

Nombre de livres et prix moyen du livre pour les livres d'un même genre ?

```
SQL> select upper(genre) GENRE,avg(prix) PRIX_MOYEN,count(*) NOMBRE from biblio group by upper(genre)
```

GENRE	PRIX_MOYEN	NOMBRE
BD	70,00	2
POèME	160,00	2
ROMAN	196,35	5

Même question mais seulement pour les livres qui ne sont pas des romans :

```
SQL>
select upper(genre) GENRE,avg(prix) PRIX_MOYEN,count(*) NOMBRE
from biblio
group by upper(genre)
having upper(GENRE)!='ROMAN'
```

GENRE	PRIX_MOYEN	NOMBRE
BD	70,00	2
POèME	160,00	2

Même question mais seulement pour les livres à moins de 150 F :

```
SQL>
select upper(genre) GENRE,avg(prix) PRIX_MOYEN,count(*) NOMBRE
from biblio
where prix<150
group by upper(genre)
having upper(GENRE)!='ROMAN'
```

GENRE	PRIX_MOYEN	NOMBRE
BD	70,00	2
POèME	120,00	1

Même question mais on ne garde que les groupes ayant un prix moyen de livre >100 F

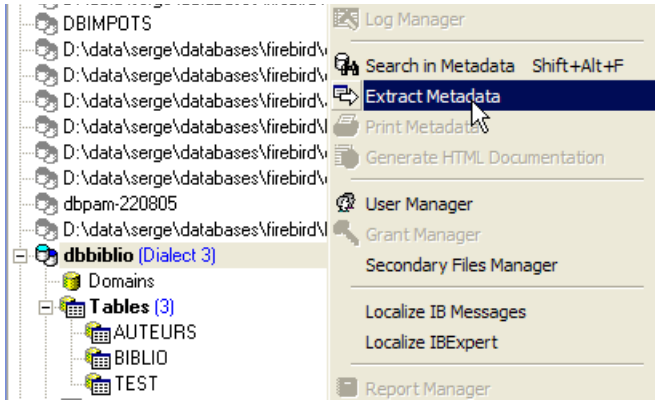
```
SQL>
select upper(genre) GENRE, avg(prix) PRIX_MOYEN,count(*) NOMBRE
from biblio
group by upper(genre)
having avg(prix)>100
```

GENRE	PRIX_MOYEN	NOMBRE
POèME	160,00	2
ROMAN	196,35	5

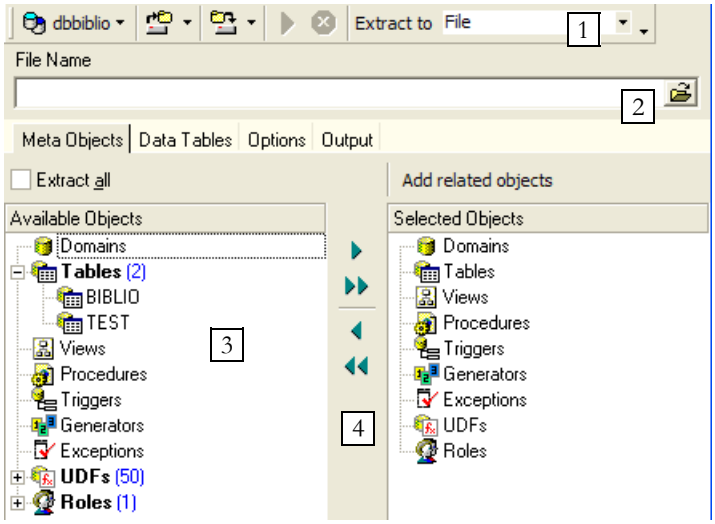
3.14 Créer le script SQL d'une table

Le langage SQL est un langage standard utilisable avec de nombreux SGBD. Afin de pouvoir passer d'un SGBD à un autre, il est intéressant d'exporter une base ou simplement certains éléments de celle-ci sous la forme d'un script SQL qui, rejoué dans un autre SGBD, sera capable de recréer les éléments exportés dans le script.

Nous allons ici exporter la table [BIBLIO]. Prenons l'option [Extract Metadata] :



On remarquera ci-dessus, qu'il faut être positionné sur la base dont on veut exporter des éléments. L'option démarre un assistant :



1 où générer le script SQL :

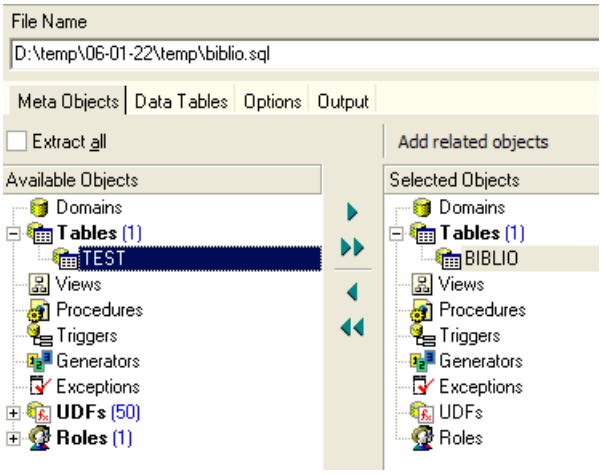
- dans un fichier (File)
- dans le Presse-Papiers (Clipboard)
- dans l'outil Script Executive

2 nom du fichier si l'option [File] est choisie

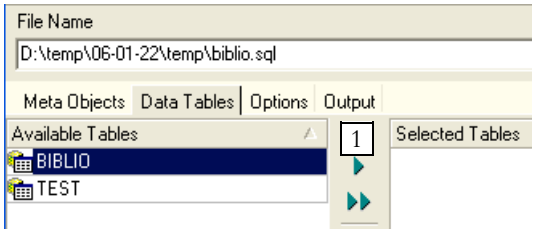
3 quoi exporter

4 boutons pour sélectionner (->) ou désélectionner (<-) les objets à exporter

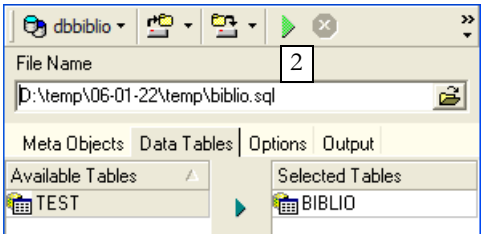
Si nous voulions exporter la totalité de la base, nous cocherions l'option [Extract All] ci-dessus. Nous voulons simplement exporter la table BIBLIO. Pour ce faire, avec [4], nous sélectionnons la table [BIBLIO] et avec [2] nous désignons un fichier :



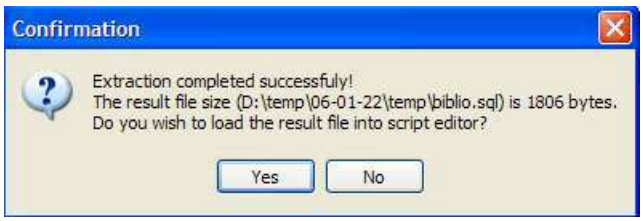
Si nous nous arrêtons là, seule la structure de la table [BIBLIO] sera exportée. Pour exporter son contenu, il nous faut utiliser l'onglet [Data Tables] :



Utillisons [1] pour sélectionner la table [BIBLIO] :



Utilisons [2] pour générer le script SQL :



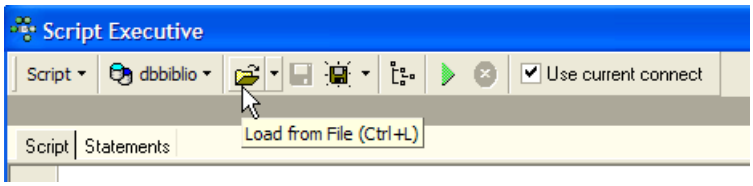
Acceptons l'offre. Ceci nous permet de voir le script qui a été généré dans le fichier [biblio.sql] :

```
1./******************************************************************************/
2./****         Generated by IBExpert 2004.06.17 22/01/2006 15:06:13         ****/
3./******************************************************************************/
4.
5.SET SQL DIALECT 3;
6.
7.SET NAMES ISO8859_1;
8.
9.CREATE DATABASE 'D:\data\serge\travail\2005-2006\polys\sql\DBBIBLIO.GDB'
10.USER 'SYSDBA' PASSWORD 'masterkey'
11.PAGE_SIZE 16384
12.DEFAULT CHARACTER SET ISO8859_1;
13.
14.
15.
```

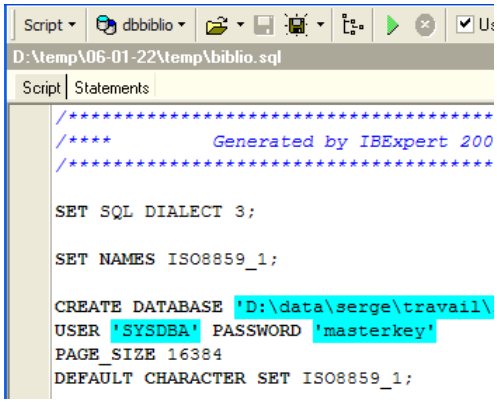
```
16./******************************************************************************/
17./****                                 Tables                                   ****/
18./******************************************************************************/
19.
20.
21.
22.CREATE TABLE BIBLIO (
23.    ID          INTEGER NOT NULL,
24.    TITRE       VARCHAR(30) NOT NULL,
25.    AUTEUR      VARCHAR(20) NOT NULL,
26.    GENRE       VARCHAR(30) NOT NULL,
27.    ACHAT       DATE NOT NULL,
28.    PRIX        NUMERIC(6,2) DEFAULT 10 NOT NULL,
29.    DISPONIBLE  CHAR(1) NOT NULL
30.);
31.
32.INSERT INTO BIBLIO (ID, TITRE, AUTEUR, GENRE, ACHAT, PRIX, DISPONIBLE) VALUES (2, 'Les fleurs du
mal', 'Baudelaire', 'POèME', '1978-01-01', 120, 'n');
33.INSERT INTO BIBLIO (ID, TITRE, AUTEUR, GENRE, ACHAT, PRIX, DISPONIBLE) VALUES (3, 'Tintin au
Tibet', 'Hergé', 'BD', '1990-11-10', 70, 'o');
34.INSERT INTO BIBLIO (ID, TITRE, AUTEUR, GENRE, ACHAT, PRIX, DISPONIBLE) VALUES (4, 'Du côté de chez
Swann', 'Proust', 'ROMAN', '1978-12-08', 220.5, 'o');
35.INSERT INTO BIBLIO (ID, TITRE, AUTEUR, GENRE, ACHAT, PRIX, DISPONIBLE) VALUES (5, 'La terre',
'Zola', 'ROMAN', '1990-06-12', 55.13, 'n');
36.INSERT INTO BIBLIO (ID, TITRE, AUTEUR, GENRE, ACHAT, PRIX, DISPONIBLE) VALUES (6, 'Madame Bovary',
'Flaubert', 'ROMAN', '1988-03-12', 143.33, 'o');
37.INSERT INTO BIBLIO (ID, TITRE, AUTEUR, GENRE, ACHAT, PRIX, DISPONIBLE) VALUES (7, 'Manhattan
transfer', 'Dos Passos', 'ROMAN', '1987-08-30', 352.8, 'o');
38.INSERT INTO BIBLIO (ID, TITRE, AUTEUR, GENRE, ACHAT, PRIX, DISPONIBLE) VALUES (8, 'Tintin en
Amérique', 'Hergé', 'BD', '1991-05-15', 70, 'o');
39.INSERT INTO BIBLIO (ID, TITRE, AUTEUR, GENRE, ACHAT, PRIX, DISPONIBLE) VALUES (10, 'Le père
Goriot', 'Balzac', 'Roman', '1991-09-01', 210, 'o');
40.INSERT INTO BIBLIO (ID, TITRE, AUTEUR, GENRE, ACHAT, PRIX, DISPONIBLE) VALUES (11, 'Poèmes
saturniens', 'Verlaine', 'Poème', '1992-09-02', 200, 'o');
41.
42.COMMIT WORK;
43.
44.
45.
46./******************************************************************************/
47./****                           Unique Constraints                             ****/
48./******************************************************************************/
49.
50.ALTER TABLE BIBLIO ADD CONSTRAINT UNQ1_BIBLIO UNIQUE (TITRE);
51.
52.
53./******************************************************************************/
54./****                             Primary Keys                                 ****/
55./******************************************************************************/
56.
57.ALTER TABLE BIBLIO ADD CONSTRAINT PK_BIBLIO PRIMARY KEY (ID);
```

- les lignes 1 à 3 sont des commentaires
- les lignes 5 à 12 sont du SQL propriétaire de Firebird
- les autres lignes sont du SQL standard qui devraient pouvoir être rejouées dans un SGBD qui aurait les types de données déclarés dans la table BIBLIO.

Rejouons ce script à l'intérieur de Firebird pour créer une table BIBLIO2 qui sera un clône de la table BIBLIO. Utillisons pour cela [Script Executive] (Ctrl-F12) :



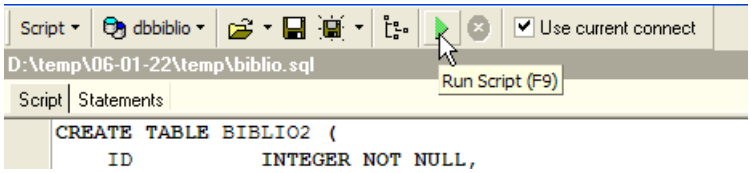
Chargeons le script [biblio.sql] que nous venons de générer :



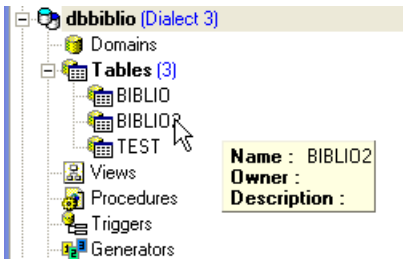
Modifions-le pour ne garder que la partie création de la table et insertion de lignes. La table est renommée [BIBLIO2] :

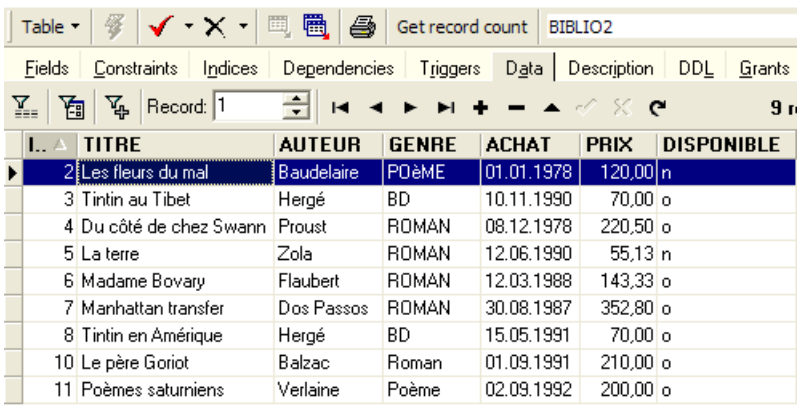
```
CREATE TABLE BIBLIO2 (
    ID          INTEGER NOT NULL,
    TITRE       VARCHAR(30) NOT NULL,
    AUTEUR      VARCHAR(20) NOT NULL,
    GENRE       VARCHAR(30) NOT NULL,
    ACHAT       DATE NOT NULL,
    PRIX        NUMERIC(6,2) DEFAULT 10 NOT NULL,
    DISPONIBLE  CHAR(1) NOT NULL
);

INSERT INTO BIBLIO2 (ID, TITRE, AUTEUR, GENRE, ACHAT, PRIX, DISPONIBLE) VALUES (2, 'Les fleurs du mal',
'Baudelaire', 'POèME', '1978-01-01', 120, 'n');
...

COMMIT WORK;
```

Exécutons ce script :



Nous pouvons vérifier dans l'explorateur de bases que la table [BIBLIO2] a bien été créée et qu'elle a bien la structure et le contenu attendus :





I..	TITRE	AUTEUR	GENRE	ACHAT	PRIX	DISPONIBLE
2	Les fleurs du mal	Baudelaire	POèME	01.01.1978	120,00	n
3	Tintin au Tibet	Hergé	BD	10.11.1990	70,00	o
4	Du côté de chez Swann	Proust	ROMAN	08.12.1978	220,50	o
5	La terre	Zola	ROMAN	12.06.1990	55,13	n
6	Madame Bovary	Flaubert	ROMAN	12.03.1988	143,33	o
7	Manhattan transfer	Dos Passos	ROMAN	30.08.1987	352,80	o
8	Tintin en Amérique	Hergé	BD	15.05.1991	70,00	o
10	Le père Goriot	Balzac	Roman	01.09.1991	210,00	o
11	Poèmes saturniens	Verlaine	Poème	02.09.1992	200,00	o

4 Les expressions du langage SQL

4.1 Introduction

Dans la plupart des commandes SQL, il est possible d'utiliser une expression. Prenons par exemple la commande SELECT :

`syntaxe` **SELECT** *expr1, expr2, ...* **from** *table*
WHERE *expression*

SELECT sélectionne les lignes pour lesquelles *expression* est vraie et affiche pour chacune d'elles les valeurs de *expri*.

Exemples

```
SQL> select prix*1.186 from biblio
SQL> select titre from biblio where prix between 100 and 150
```

Nous nous proposons dans ce paragraphe d'expliciter la notion d'expression. Une expression élémentaire est du type :

opérande1 *opérateur* **opérande2**
ou
fonction(*paramètres*)

Exemple

Dans l'expression *GENRE = 'ROMAN'*

- GENRE est l'opérande1
- 'ROMAN' est l'opérande2
- = est l'opérateur

Dans l'expression
upper(*genre*)

- **upper** est une fonction
- genre est un paramètre de cette fonction.

Nous traitons tout d'abord des expressions avec opérateurs, puis nous présenterons les fonctions disponibles sous Firebird.

4.2 Expressions avec opérateur

Nous classifierons les expressions avec opérateur suivant le type de leurs opérandes :

. numérique
. chaîne de caractères
. date
. booléen ou logique

4.2.1 Les expressions à opérandes de type numérique

4.2.1.1 Liste des opérateurs

Soient *nombre1*, *nombre2*, *nombre3* des nombres. Les opérateurs utilisables sont les suivants :

Opérateurs relationnels

`nombre1 > nombre2`	: nombre1 plus grand que nombre2
`nombre1 >= nombre2`	: nombre1 plus grand ou égal à nombre2
`nombre1 < nombre2`	: nombre1 plus petit que nombre2
`nombre1 <= nombre2`	: nombre1 plus petit ou égal à nombre2
`nombre1 = nombre2`	: nombre1 égal à nombre2

`nombre1 != nombre2` : nombre1 différent de nombre2
`nombre1 <> nombre2` : idem
`nombre1 BETWEEN nombre2 AND nombre3` : nombre1 dans l'intervalle [nombre2,nombre3]
`nombre1 IN (liste de nombres)` : nombre1 appartient à liste de nombres
`nombre1 IS NULL` : nombre1 n'a pas de valeur
`nombre1 IS NOT NULL` : nombre1 a une valeur

Opérateurs arithmétiques

`nombre1 + nombre2` : addition
`nombre1 - nombre2` : soustraction
`nombre1 * nombre2` : multiplication
`nombre1 / nombre2` : division

4.2.1.2 Opérateurs relationnels

Une expression relationnelle exprime une relation qui est vraie ou fausse. Le résultat d'une telle expression est donc un booléen ou valeur logique.

Exemples :

```
SQL> select titre,prix from biblio where prix between 100 and 150
```

TITRE	PRIX
Les fleurs du mal	120,00
Madame Bovary	143,33

```
SQL> select titre,prix from biblio where prix not between 100 and 150
```

TITRE	PRIX
Tintin au Tibet	70,00
Du côté de chez Swann	220,50
La terre	55,13
Manhattan transfer	352,80
Tintin en Amérique	70,00
Le père Goriot	210,00
Poèmes saturniens	200,00

```
SQL> select titre,prix from biblio where prix in (200,210)
```

TITRE	PRIX
Le père Goriot	210,00
Poèmes saturniens	200,00

4.2.1.3 Opérateurs arithmétiques

L'expression arithmétique nous est familière. Elle exprime un calcul à faire entre des données numériques. Nous avons déjà rencontré de telles expressions : on suppose que le prix mémorisé dans les fiches du fichier BIBLIO soit un prix hors taxes. On veut visualiser chaque titre avec son prix TTC pour un taux de TVA de 18.6% :

```
SELECT TITRE, PRIX*1.186 FROM BIBLIO
```

Si les prix doivent augmenter de 3%, la commande sera

```
UPDATE BIBLIO SET PRIX = PRIX*1.03
```

On peut trouver plusieurs opérateurs arithmétiques dans une expression avec de plus des fonctions et des parenthèses. Ces éléments sont traités selon des priorités différentes :

1 `fonctions` <---- plus prioritaire
2 `()`
3 `* et /`

4 `+ et -` <---- moins prioritaire

Lorsque deux opérateurs de même priorité sont présents dans l'expression, c'est celui qui est le plus à gauche dans l'expression qui est évalué tout d'abord.

Exemples

L'expression PRIX*TAUX+TAXES sera évaluée comme (PRIX*TAUX)+TAXES. C'est en effet l'opérateur de multiplication qui sera utilisé en premier. L'expresssion PRIX*TAUX/100 sera évaluée comme (PRIX*TAUX)/100.

4.2.2 Les expressions à opérandes de type caractères

4.2.2.1 Liste des opérateurs

Les opérateurs utilisables sont les suivants :

Soient *chaine1, chaine2, chaine3, modèle* des chaînes de caractères

`chaine1 > chaine2`	: chaine1 plus grande que chaine2
`chaine1 >= chaine2`	: chaine1 plus grande ou égale à chaine2
`chaine1 < chaine2`	: chaine1 plus petite que chaine2
`chaine1 <= chaine2`	: chaine1 plus petite ou égale à chaine2
`chaine1 = chaine2`	: chaine1 égale à chaine2
`chaine1 != chaine2`	: chaine1 différente de chaine2
`chaine1 <> chaine2`	: idem
`chaine1 BETWEEN chaine2 AND chaine3`	: chaine1 dans l'intervalle [chaine2,chaine3]
`chaine1 IN liste de chaines`	: chaine1 appartient à liste de chaines
`chaine1 IS NULL`	: chaine1 n'a pas de valeur
`chaine1 IS NOT NULL`	: chaine1 a une valeur
`chaine1 LIKE modèle`	: chaine1 correspond à modèle

Opérateur de concaténation

chaine1 || *chaine2* : *chaine2* concaténée à *chaine1*

4.2.2.2 Opérateurs relationnels

Que signifie comparer des chaînes avec des opérateurs tels que <, <=, etc ... ?

Tout caractère est codé par un nombre entier. Lorsqu'on compare deux caractères, ce sont leurs codes entiers qui sont comparés. Le codage adopté respecte l'ordre naturel du dictionnaire :

```
blanc<..< 0 < 1 < ...< 9 < ...< A < B <... < Z < ... < a < b < ... < z
```

Les chiffres viennent avant les lettres, et les majuscules avant les minuscules.

4.2.2.3 Comparaison de deux chaînes

Soit la relation 'CHAT' < 'CHIEN'. Est-elle vraie ou fausse ? Pour effectuer cette comparaison, le SGBD compare les deux chaînes caractère par caractère sur la base de leurs codes entiers. Dès que deux caractères sont trouvés différents, la chaîne à qui appartient le plus petit des deux est dite plus petite que l'autre chaîne. Dans notre exemple 'CHAT' est comparée à 'CHIEN'. On a les résultats successifs suivants :

```
  'CHAT' 'CHIEN'
----------------------------
  'C' = 'C'
  'H' = 'H'
  'A' < 'I'
```

Après cette dernière comparaison, la chaîne 'CHAT' est déclarée plus petite que la chaîne 'CHIEN'. La relation *'CHAT' < 'CHIEN'* est donc vraie.

Soit à comparer maintenant 'CHAT' et 'chat'.

```
  'CHAT' 'chat'
-------------------------
  'C' < 'c'
```

Après cette comparaison, la relation 'CHAT' < 'chat' est déclarée vraie.

Exemples

```
SQL> select titre from biblio
```

TITRE
Les fleurs du mal
Tintin au Tibet
Du côté de chez Swann
La terre
Madame Bovary
Manhattan transfer
Tintin en Amérique
Le père Goriot
Poèmes saturniens

```
SQL> select titre from biblio where upper(titre) between 'L' and 'M'
```

TITRE
Les fleurs du mal
La terre
Le père Goriot

4.2.2.4 L'opérateur LIKE

L'opérateur LIKE s'utilise comme suit : *chaîne* **LIKE** *modèle*

La relation est vraie si *chaîne* correspond au *modèle*. Celui-ci est une chaîne de caractères pouvant comporter deux caractères génériques :

- `%` qui désigne toute suite de caractères
- `_` qui désigne 1 caractère quelconque

Exemples

```
SQL> select titre from biblio
```

TITRE
Les fleurs du mal
Tintin au Tibet
Du côté de chez Swann
La terre
Madame Bovary
Manhattan transfer
Tintin en Amérique
Le père Goriot
Poèmes saturniens

```
SQL> select titre from biblio where titre like 'M%';
```

TITRE
Madame Bovary
Manhattan transfer

```
SQL> select titre from biblio where titre like 'L_ %';
```

TITRE
La terre
Le père Goriot

4.2.2.5 L'opérateur de concaténation

```
SQL > select '[' || titre || ']' from biblio where upper(titre) LIKE 'L_ %'
```

CONCATENATION
[La terre]
[Le père Goriot]

4.2.3 Les expressions à opérandes de type date

Soient *date1, date2, date3* des dates. Les opérateurs utilisables sont les suivants :

Opérateurs relationnels

`date1 < date2`	est vraie si date1 est antérieure à date2
`date1 <= date2`	est vraie si date1 est antérieure ou égale à date2
`date1 > date2`	est vraie si date1 est postérieure à date2
`date1 >= date2`	est vraie si date1 est postérieure ou égale à date2
`date1 = date2`	est vraie si date1 et date2 sont identiques
`date1 <> date2`	est vraie si date1 et date2 sont différentes.
`date1 != date2`	idem
`date1 BETWEEN date2 AND date3`	est vraie si date1 est situé entre date2 et date3
`date1 IN (liste de dates)`	est vraie si date1 se trouve dans la liste de dates
`date1 IS NULL`	est vraie si date1 n'a pas de valeur
`date1 IS NOT NULL`	est vraie si date1 a une valeur
`date1 LIKE modèle`	est vraie si date1 correspond au modèle
`ALL,ANY,EXISTS`	

Opérateurs arithmétiques

`date1 - date2`	: nombre de jours séparant date1 de date2
`date1 - nombre`	: date2 telle que date1-date2=nombre
`date1 + nombre`	: date2 telle que date2-date1=nombre

Exemples

```
SQL> select achat from biblio
```

ACHAT
01.01.1978
10.11.1990
08.12.1978
12.06.1990
12.03.1988
30.08.1987
15.05.1991
01.09.1991
02.09.1992

```
SQL>select achat from biblio
   where achat between '01.01.1988' and '31.12.1988';
```

ACHAT
12.03.1988

```
SQL> select titre, achat from biblio
where cast(achat as char(10)) like '1988'
```

TITRE	ACHAT
Madame Bovary	12.03.1988

Age des livres de la bibliothèque ?

```
SQL> select titre, cast('now' as date)-achat "age(jours)" from biblio
```

TITRE	age(jours)
Les fleurs du mal	10 246
Tintin au Tibet	5 550
Du côté de chez Swann	9 905
La terre	5 701
Madame Bovary	6 523
Manhattan transfer	6 718
Tintin en Amérique	5 364
Le père Goriot	5 255
Poèmes saturniens	4 888

4.2.4 Expressions à opérandes booléens

Rappelons qu'un booléen ou valeur logique a deux valeurs possibles : vrai ou faux. L'opérande logique est souvent le résultat d'une expression relationnelle.

Soient *booléen1* et *booléen2* deux booléens. Il y a trois opérateurs possibles qui sont par ordre de priorité :

booléen1 **AND** *booléen2*	est vraie si *booléen1* et *booléen2* sont vrais tous les deux.
booléen1 **OR** *booléen2*	est vraie si *booléen1* ou *booléen2* est vrai.
NOT *booléen1*	a pour valeur l'inverse de la valeur de *booléen1*.

Exemples

```
SQL> select titre,genre,prix from biblio order by prix desc
```

TITRE	GENRE	PRIX
Manhattan transfer	ROMAN	352,80
Du côté de chez Swann	ROMAN	220,50
Le père Goriot	Roman	210,00
Poèmes saturniens	Poème	200,00
Madame Bovary	ROMAN	143,33
Les fleurs du mal	POèME	120,00
Tintin au Tibet	BD	70,00
Tintin en Amérique	BD	70,00
La terre	ROMAN	55,13

On recherche les livres entre deux prix :

```
SQL> select titre,genre,prix from biblio
     where prix>=130 and prix <=170
```

TITRE	GENRE	PRIX
Madame Bovary	ROMAN	143,33

Recherche inverse :

```
SQL> select titre,genre,prix from biblio
     where prix<130 or prix >170
     order by prix asc
```

TITRE	GENRE	PRIX
La terre	ROMAN	55,13
Tintin au Tibet	BD	70,00
Tintin en Amérique	BD	70,00
Les fleurs du mal	POèME	120,00
Poèmes saturniens	Poème	200,00
Le père Goriot	Roman	210,00
Du côté de chez Swann	ROMAN	220,50
Manhattan transfer	ROMAN	352,80

Attention à la priorité des opérateurs !

```
SQL> select titre,genre,prix from biblio
  where genre='ROMAN' and prix>200 or prix<100
  order by prix asc
```

TITRE	GENRE	PRIX
La terre	ROMAN	55,13
Tintin au Tibet	BD	70,00
Tintin en Amérique	BD	70,00
Du côté de chez Swann	ROMAN	220,50
Manhattan transfer	ROMAN	352,80

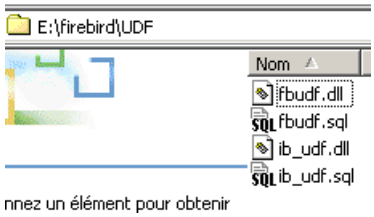
On met des parenthèses pour contrôler la priorité des opérateurs :

```
SQL> select titre,genre,prix from biblio
  where genre='ROMAN' and (prix>200 or prix<100)
  order by prix asc
```

TITRE	GENRE	PRIX
La terre	ROMAN	55,13
Du côté de chez Swann	ROMAN	220,50
Manhattan transfer	ROMAN	352,80

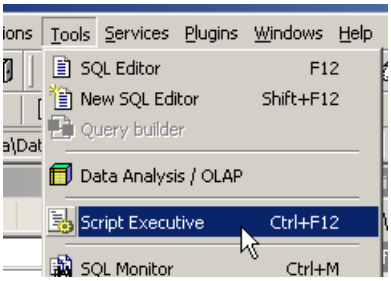
4.3 Les fonctions prédéfinies de Firebird

Firebird dispose de fonctions prédéfinies. Elles ne sont pas immédiatement utilisables dans les ordres SQL. Il faut tout d'abord exécuter le script SQL **<firebird>\UDF\ib_udf.sql** où <firebird> désigne le répertoire d'installation du SGBD Firebird :



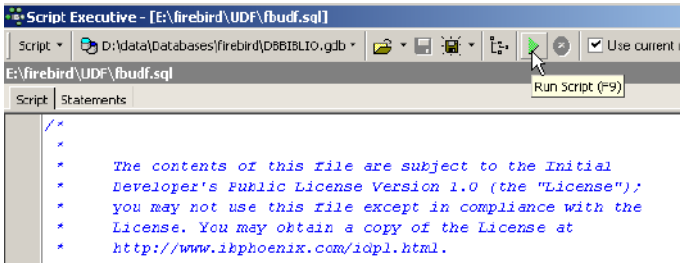
Avec IBExpert, nous procédons de la façon suivante :

- nous utilisons l'outil [Script Excecutive] obtenu par l'option [Tools/ Script Executive] :

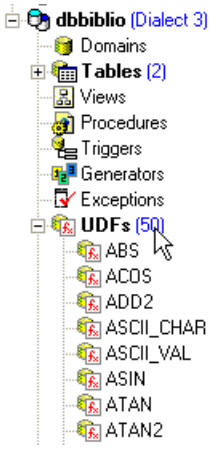


- une fois l'outil présent, nous chargeons le script <firebird>\UDF\ib_udf.sql :

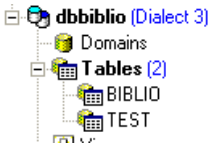
- puis nous exécutons le script :



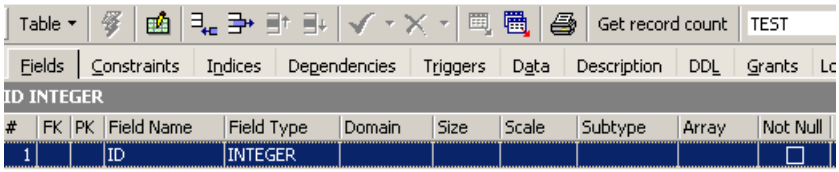
Ceci fait, les fonctions prédéfinies de Firebird sont disponibles pour la base à laquelle on était connectée lorsque le script a été exécuté. Pour le voir, il suffit d'aller dans l'explorateur de bases, et de cliquer sur le noeud [UDF] de la base dans laquelle ont été importées les fonctions :



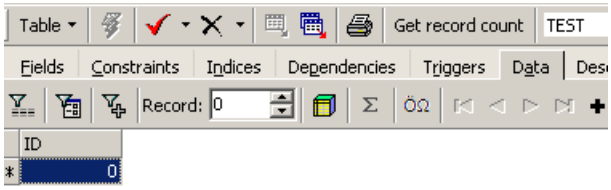
On a ci-dessus, les fonctions disponibles à la base. Pour les tester, il est pratique d'avoir une table à une ligne. Appelons-la TEST :



et définissons-la comme suit (clic droit sur Tables / New Table) :



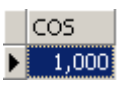
Mettons une unique ligne dans cette table :



Passons maintenant dans l'éditeur SQL (F12) et émettons l'ordre SQL suivant :

```
SQL> select cos(0) from test
```

qui utilise la fonction prédéfinie cos (cosinus). La commande ci-dessus évalue *cos(0)* pour chaque ligne de la table TEST, donc en fait pour une seule ligne. Il y a donc simplement affichage de la valeur de *cos(0) :*



Les fonctions UDF (User Defined Fonction) sont des fonctions que l'utilisateur peut créer et on peut ainsi trouver des bibliothèques de fonctions UDF sur le web. Nous ne décrivons ici que certaines de celles disponibles avec la version téléchargeable de Firebird (2005). Nous les classifions selon le type prédominant de leurs paramètres ou selon leur rôle :

. fonctions à paramètres de type numérique
. fonctions à paramètres de type chaîne de caractères

4.3.1 Fonctions à paramètres de type numérique

abs*(nombre)*	valeur absolue de nombre abs(-15)=15
ceil*(nombre)*	plus petit entier plus grand ou égal à *nombre* ceil(15.7)=16
floor*(nombre)*	plus grand entier inférieur ou égal à *nombre* floor(14.3)=14
div*(nombre1,nombre2)*	quotient de la division entière (le quotient est entier) de *nombre1* par *nombre2* div(7,3)=2
mod*(nombre1,nombre2)*	reste de la division entière (le quotient est entier) de *nombre1* par *nombre2* mod(7,3)=1
sign*(nombre)*	-1 si *nombre*<0 0 si *nombre*=0 +1 si *nombre*>0 sign(-6)=-1
sqrt*(nombre)*	racine carrée de *nombre* si *nombre*>=0 -1 si *nombre*<0 sqrt(16)=4

4.3.2 Fonctions à paramètres de type chaîne de caractères

ascii_char*(nombre)*	caractère de code ASCII *nombre* ascii_char(65)='A'
lower*(chaine)*	met *chaine* en minuscules lower('INFO')='info'
ltrim*(chaine)*	Left Trim - Les espaces précédant le texte de *chaine* sont supprimés : ltrim(' chaton')='chaton'
replace*(chaine1,chaine2,chaine3)*	remplace *chaine2* par *chaine3* dans *chaine1*. replace('chat et chien','ch','**')='**at et **ien'
rtrim*(chaine1,chaine2)*	Right Trim - idem ltrim mais à droite rtrim('chat ')='chat'
substr*(chaine,p,q)*	sous-chaîne de *chaine* commençant en position *p* et se terminant en position *q*. substr('chaton',3,5)='ato'
ascii_val*(caractère)*	code ASCII de *caractère* ascii_val('A')=65
strlen(*chaine)*	nombre de caractères de *chaine* strlen('chaton')=6

5 Relations entre tables

5.1 Les clés étrangères

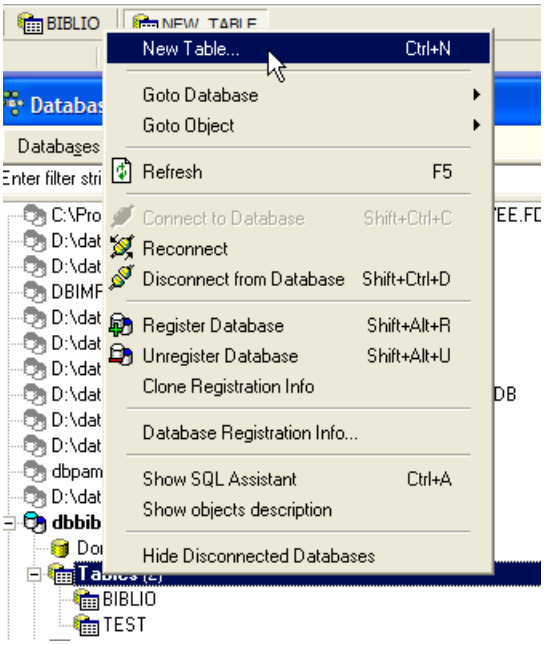
Une base de données relationnelle est un ensemble de tables liées entre-elles par des relations. Prenons un exemple inspiré de la table [BIBLIO] précédente dont la structure était la suivante :

#	FK	PK	Field Name	Field Type	Domain	Size
1		1	ID	INTEGER		
2			TITRE	VARCHAR		30
3			AUTEUR	VARCHAR		20
4			GENRE	VARCHAR		30
5			ACHAT	DATE		
6			PRIX	NUMERIC		6
7			DISPONIBLE	CHAR		1

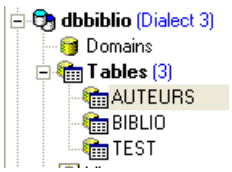
Un exemple de contenu était le suivant :

ID	TITRE	AUTEUR	GENRE	ACHAT	PRIX	DISPONIBLE
2	Les fleurs du mal	Baudelaire	POèME	01.01.1978	120,00	n
3	Tintin au Tibet	Hergé	BD	10.11.1990	70,00	o
4	Du côté de chez Swann	Proust	ROMAN	08.12.1978	220,50	o
5	La terre	Zola	ROMAN	12.06.1990	55,13	n
6	Madame Bovary	Flaubert	ROMAN	12.03.1988	143,33	o
7	Manhattan transfer	Dos Passos	ROMAN	30.08.1987	352,80	o
8	Tintin en Amérique	Hergé	BD	15.05.1991	70,00	o
10	Le père Goriot	Balzac	Roman	01.09.1991	210,00	o
11	Poèmes saturniens	Verlaine	Poème	02.09.1992	200,00	o

On pourrait vouloir des informations sur les différents auteurs de ces ouvrages par exemple ses *nom* et *prénom*, sa *date de naissance*, sa *nationalité*. Créons une telle table. Cliquons droit sur [DBBIBLIO / Tables] puis prenons l'option [New Table] :



Construisons maintenant la table [AUTEURS] suivante :



FK	PK	Field Name	Field Type	Domain	Size
	1	ID	INTEGER		
		NOM	VARCHAR		30
		PRENOM	VARCHAR		30
		DATE_NAISSANCE	DATE		
		NATIONALITE	VARCHAR		20

`id`	clé primaire de la table - sert à identifier une ligne de façon unique
`nom`	nom de l'auteur
`prénom`	prénom de l'auteur s'il en a un
`date_naissance`	sa date de naissance
`nationalite`	son pays d'origine

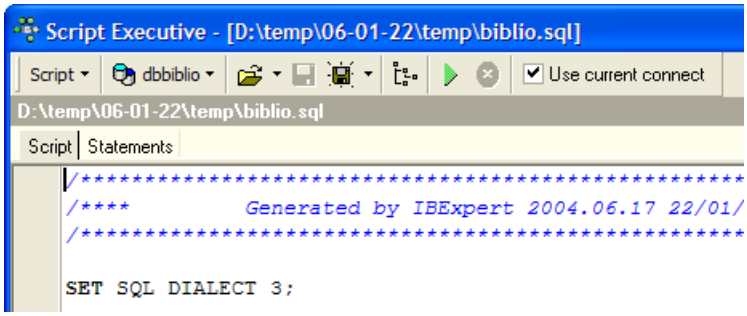
Le contenu de la table [AUTEURS] pourrait être le suivant :

ID	NOM	PRENOM	DATE_NAISSANCE	NATIONALITE
1	Flaubert	Gustave	12.12.1821	FRANCE
2	Verlaine	Paul	30.03.1884	FRANCE
3	Proust	Marcel	10.07.1871	FRANCE
4	Zola	Emile	02.04.1840	FRANCE
5	Balzac	Honoré de	20.05.1799	FRANCE
6	Hergé	<null>	22.05.1907	BELGIQUE
7	Passos	John Dos	14.01.1886	ETATS-UNIS
8	Baudelaire	Charles	09.04.1821	FRANCE

Revenons à la table [BIBLIO] et à son contenu :

ID	TITRE	AUTEUR	GENRE	ACHAT	PRIX	DISPONIBLE
2	Les fleurs du mal	Baudelaire	POèME	01.01.1978	120,00	n
3	Tintin au Tibet	Hergé	BD	10.11.1990	70,00	o
4	Du côté de chez Swann	Proust	ROMAN	08.12.1978	220,50	o
5	La terre	Zola	ROMAN	12.06.1990	55,13	n
6	Madame Bovary	Flaubert	ROMAN	12.03.1988	143,33	o
7	Manhattan transfer	Dos Passos	ROMAN	30.08.1987	352,80	o
8	Tintin en Amérique	Hergé	BD	15.05.1991	70,00	o
10	Le père Goriot	Balzac	Roman	01.09.1991	210,00	o
11	Poèmes saturniens	Verlaine	Poème	02.09.1992	200,00	o

Dans la rubrique [AUTEUR] de la table, il devient inutile de mettre le nom de l'auteur. Il est plutôt préférable de mettre le n° (id) qu'il a dans la table [AUTEURS]. Créons donc une nouvelle table appelée [LIVRES]. Pour la créer, nous allons utiliser le script [biblio.sql] créé au paragraphe 3.14, page 43. Nous chargeons ce script avec l'outil [Script Executive, Ctrl-F12] :



Nous modifions le script de création de la table BIBLIO pour l'adapter à celui de la table LIVRES :

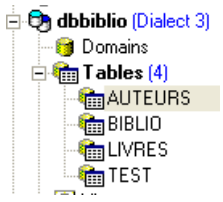
```
1.CREATE TABLE LIVRES (
2.    ID           INTEGER NOT NULL,
3.    TITRE        VARCHAR(30) NOT NULL,
4.    AUTEUR       INTEGER,
5.    GENRE        VARCHAR(30) NOT NULL,
```

```
6.    ACHAT       DATE NOT NULL,
7.    PRIX        NUMERIC(6,2) DEFAULT 10 NOT NULL,
8.    DISPONIBLE  CHAR(1) NOT NULL
9.);
10.
11.INSERT INTO LIVRES (ID, TITRE, AUTEUR, GENRE, ACHAT, PRIX, DISPONIBLE) VALUES (2, 'Les fleurs
du mal', 8, 'POèME', '1978-01-01', 120, 'n');
12.INSERT INTO LIVRES (ID, TITRE, AUTEUR, GENRE, ACHAT, PRIX, DISPONIBLE) VALUES (3, 'Tintin au
Tibet', 6, 'BD', '1990-11-10', 70, 'o');
13.INSERT INTO LIVRES (ID, TITRE, AUTEUR, GENRE, ACHAT, PRIX, DISPONIBLE) VALUES (4, 'Du côté de
chez Swann', 3, 'ROMAN', '1978-12-08', 220.5, 'o');
14.INSERT INTO LIVRES (ID, TITRE, AUTEUR, GENRE, ACHAT, PRIX, DISPONIBLE) VALUES (5, 'La terre',
4, 'ROMAN', '1990-06-12', 55.13, 'n');
15.INSERT INTO LIVRES (ID, TITRE, AUTEUR, GENRE, ACHAT, PRIX, DISPONIBLE) VALUES (6, 'Madame
Bovary', 1, 'ROMAN', '1988-03-12', 143.33, 'o');
16.INSERT INTO LIVRES (ID, TITRE, AUTEUR, GENRE, ACHAT, PRIX, DISPONIBLE) VALUES (7, 'Manhattan
transfer', 7, 'ROMAN', '1987-08-30', 352.8, 'o');
17.INSERT INTO LIVRES (ID, TITRE, AUTEUR, GENRE, ACHAT, PRIX, DISPONIBLE) VALUES (8, 'Tintin en
Amérique', 6, 'BD', '1991-05-15', 70, 'o');
18.INSERT INTO LIVRES (ID, TITRE, AUTEUR, GENRE, ACHAT, PRIX, DISPONIBLE) VALUES (10, 'Le père
Goriot', 5, 'Roman', '1991-09-01', 210, 'o');
19.INSERT INTO LIVRES (ID, TITRE, AUTEUR, GENRE, ACHAT, PRIX, DISPONIBLE) VALUES (11, 'Poèmes
saturniens', 2, 'Poème', '1992-09-02', 200, 'o');
20.
21.COMMIT WORK;
22.
23.
24.
25./******************************************************************************/
26./****                          Unique Constraints                          ****/
27./******************************************************************************/
28.
29.ALTER TABLE LIVRES ADD CONSTRAINT UNQ1_LIVRES UNIQUE (TITRE);
30.
31.
32./******************************************************************************/
33./****                            Primary Keys                              ****/
34./******************************************************************************/
35.
36.ALTER TABLE LIVRES ADD CONSTRAINT PK_LIVRES PRIMARY KEY (ID);
```

Nous ne commentons que les changements :

- ligne 4 : la rubrique [AUTEUR] de la table devient un numéro entier. Ce numéro référence l'un des auteurs de la table [AUTEURS] construite précédemment.
- lignes 11-19 : les noms des auteurs ont été remplacés par leurs numéros d'auteur.
- ligne 29 : le nom de la contrainte a été changée. Elle s'appelait auparavant [UNQ1_BIBLIO]. Elle s'appelle désormais [UNQ1_LIVRES]. Ce nom peut être quelconque. Il est préférable cependant qu'il ait un sens. Ici cet effort n'a pas été fait. Les contraintes sur les différents champs, les différentes tables d'une base doivent être différenciées par des noms différents. Rappelons que la contrainte de la ligne 29 demande qu'un titre soit unique dans la table.
- ligne 36 : changement du nom de la contrainte sur la clé primaire ID.

Exécutons ce script. S'il réussit, nous obtenons la nouvelle table [LIVRES] suivante :



ID	TITRE	AUTEUR	GENRE	ACHAT	PRIX	DISPONIBLE
2	Les fleurs du mal	8	POèME	01.01.1978	120,00	n
3	Tintin au Tibet	6	BD	10.11.1990	70,00	o
4	Du côté de chez Swann	3	ROMAN	08.12.1978	220,50	o
5	La terre	4	ROMAN	12.06.1990	55,13	n
6	Madame Bovary	1	ROMAN	12.03.1988	143,33	o
7	Manhattan transfer	7	ROMAN	30.08.1987	352,80	o
8	Tintin en Amérique	6	BD	15.05.1991	70,00	o
10	Le père Goriot	5	Roman	01.09.1991	210,00	o
11	Poèmes saturniens	2	Poème	02.09.1992	200,00	o

On peut se demander si finalement nous avons gagné au change. En effet, la table [LIVRES] présente des n°s d'auteurs au lieu de leurs noms. Comme il y a des milliers d'auteurs, le lien entre un livre et son auteur semble difficile à faire. Heureusement le langage SQL est là pour nous aider. Il nous permet d'interroger plusieurs tables en même temps. Pour

l'exemple, nous présentons la requête SQL qui nous permet d'obtenir les titres des livres de la bibliothèque, associés aux informations de leurs auteurs. Utilisons l'éditeur SQL (F12) pour émettre l'ordre SQL suivant :

```
SQL> select LIVRES.titre, AUTEURS.nom, AUTEURS.prenom,AUTEURS.date_naissance
FROM LIVRES inner join AUTEURS on LIVRES.AUTEUR=AUTEURS.ID
ORDER BY AUTEURS.nom asc
```

Il est trop tôt pour expliquer cet ordre SQL. Nous reviendrons dessus prochainement. Le résultat de cette requête est le suivant :

TITRE	NOM	PRENOM	DATE_NAISSANCE
Le père Goriot	Balzac	Honoré de	20.05.1799
Les fleurs du mal	Baudelaire	Charles	09.04.1821
Madame Bovary	Flaubert	Gustave	12.12.1821
Tintin au Tibet	Hergé	<null>	22.05.1907
Tintin en Amérique	Hergé	<null>	22.05.1907
Manhattan transfer	Passos	John Dos	14.01.1886
Du côté de chez Swann	Proust	Marcel	10.07.1871
Poèmes saturniens	Verlaine	Paul	30.03.1884
La terre	Zola	Emile	02.04.1840

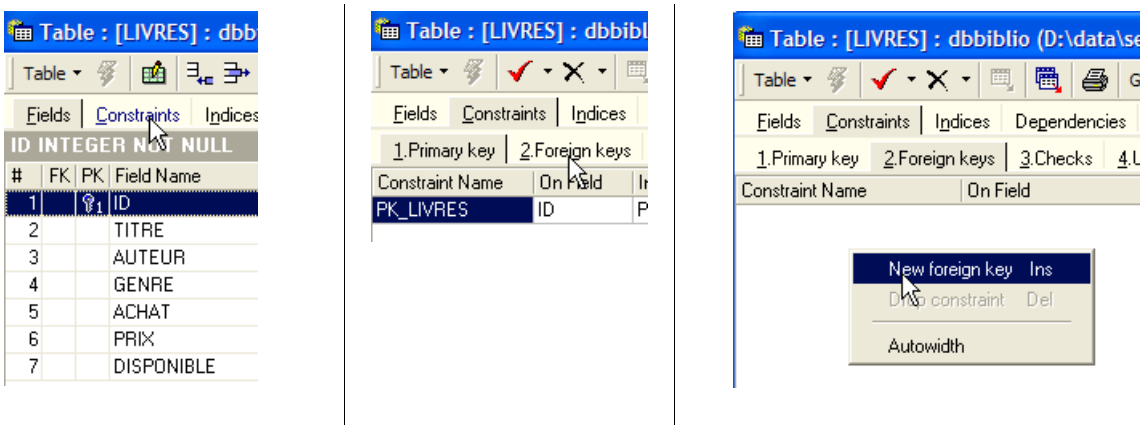
Chaque livre a été associé correctement à son auteur et aux informations qui lui sont liées.

Résumons ce que nous venons de faire :

- nous avons deux tables rassemblant des informations de nature différente :
  - la table AUTEURS rassemble des informations sur les auteurs
  - la table LIVRES rassemble des informations sur les livres achetés par la bibliothèque
- ces tables sont liées entre-elles. Un livre a forcément un auteur. Il peut même en avoir plusieurs. Ce cas n'a pas été pris en compte ici. La rubrique [AUTEUR] de la table [LIVRES] référence une ligne de la table [AUTEURS]. On appelle cela une **relation**.

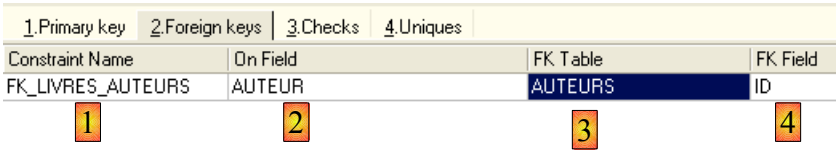
La relation qui lie la table [LIVRES] à la table [AUTEURS] est en fait une forme de contrainte : une ligne de la table [LIVRES] doit toujours avoir un n° d'auteur qui existe dans la table [AUTEURS]. Si une ligne de [LIVRES] avait un n° d'auteur qui n'existe pas dans la table [AUTEURS], on serait dans une situation anormale où on ne serait pas capable de retrouver l'auteur d'un livre.

Le SGBD est capable de vérifier que cette contrainte est toujours vérifiée. Pour cela, nous allons ajouter une contrainte à la table [LIVRES] :



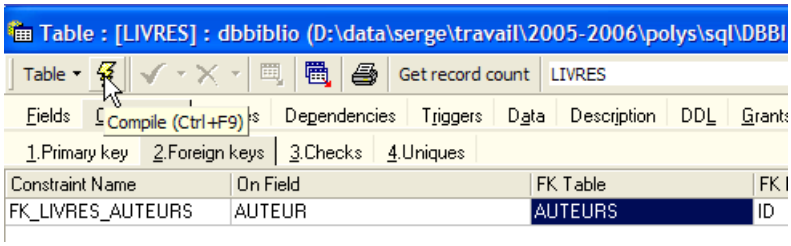
Le lien qui unit la rubrique [AUTEUR] de la table [LIVRES] au champ [ID] de la table [AUTEURS] s'appelle un lien de **clé étangère**. La rubrique [AUTEUR] de la table [LIVRES] est appelée "clé étrangère" ou "foreign key" dans l'assistant ci-dessus. Définir une clé étrangère, c'est dire que la valeur d'une colonne [c1] d'une table [T1] doit exister dans la colonne [c2] de la table [T2]. La colonne [c1] est dite "clé étrangère" de la table T1 sur la colonne [c2] de la table [T2]. La colonne [c2] est souvent clé primaire de la table [T2], mais ce n'est pas obligatoire.

Nous définissons la clé étrangère [AUTEUR] de la table [LIVRES] sur le champ [ID] de la table [AUTEURS] de la façon suivante :

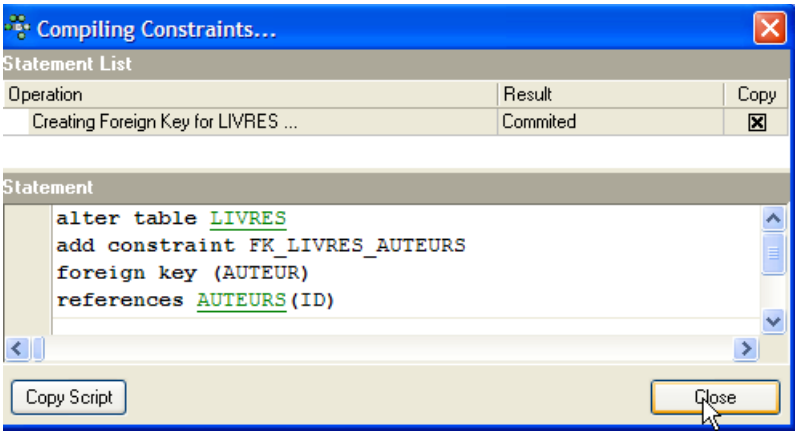


1. nom de la contrainte : libre
2. colonne "clé étrangère", ici la colonne [AUTEUR] de la table [LIVRES]
3. table référencée par la clé étangère. Ici la colonne [AUTEUR] de la table [LIVRES] doit avoir une valeur dans la colonne [ID] de la table [AUTEURS]. C'est donc la table [AUTEURS] qui est référencée.
4. colonne référencée par la clé étangère. Ici la colonne [ID] de la table [AUTEURS].

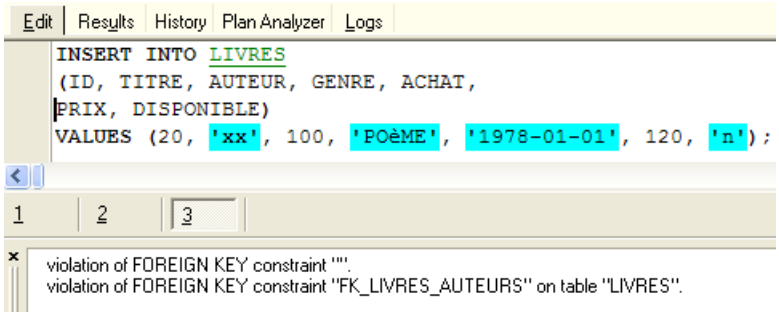
Nous validons cette contrainte :



Si tout va bien, elle est acceptée :



Quelle est la conséquence de cette nouvelle contrainte de clé étrangère ? Avec l'éditeur SQL (F12), essayons d'insérer une ligne dans la table LIVRES avec un n° d'auteur inexistant :



L'opération [INSERT] ci-dessus a essayé d'insérer un livre avec un n° d'auteur (100) inexistant. L'exécution de la requête a échoué. Le message d'erreur associé indique qu'il y a eu violation de la contrainte de clé étrangère "FK_LIVRES_AUTEURS". C'est celle que nous venons de définir.

5.2 Opérations de jointures entre deux tables

Toujours dans la base [DBBIBLIO] (ou une autre base peu importe), créons deux tables de test appelées TA et TB et définies comme suit :

Table TA

PK	Field Name	Field Type	Domain	Size
1	ID	INTEGER		
	DATA	VARCHAR		10

ID	DATA
1	data1
2	data2
3	data3

- ID : clé primaire de la table TA

- DATA : une donnée quelconque

Table TB

FK	PK	Field Name	Field Type	Domain	Size
	1	ID	INTEGER		
		VALEUR	VARCHAR		10
F		IDTA	INTEGER		

ID	VALEUR	IDTA
1	valeur1	2
2	valeur2	1
3	valeur3	2

- ID : clé primaire de la table TB

- IDTA : clé étrangère de la table TB qui référence la colonne ID de la table TA. Ainsi une valeur de la colonne IDTA de la table TA doit exister dans la colonne ID de la table TA

- VALEUR : une donnée quelconque

Dans l'éditeur SQL (F12), nous allons émettre des ordres SQL exploitant simultanément les deux tables TA et TB.

```
SQL>select * from TA,  TB
```

ID	DATA	ID1	VALEUR	IDTA
1	data1	1	valeur1	2
2	data2	1	valeur1	2
3	data3	1	valeur1	2
1	data1	2	valeur2	1
2	data2	2	valeur2	1
3	data3	2	valeur2	1
1	data1	3	valeur3	2
2	data2	3	valeur3	2
3	data3	3	valeur3	2

L'odre SQL fait intervenir derrière le mot clé FROM, les deux tables TA et TB. L'opération FROM TA, TB va provoquer la création temporaire d'une nouvelle table dans laquelle chaque ligne de la table TA sera associée à chacune des lignes de la table TB. Ainsi si la table TA a NA lignes et la table TB a NB lignes, la table résultante aura NA x NB lignes. C'est ce que montre la copie d'écran ci-dessus. Par ailleurs, chaque ligne a les colonnes des deux tables. Les colonnes *coli* précisées dans l'ordre [SELECT *col1*, *col2*, ... FROM ...] indiquent celles qu'il faut retenir. Ici le mot clé * indique que toutes les colonnes de la table résultante sont demandées. On dit parfois que la table résultante de l'ordre SQL précédent est le **produit cartésien** des tables TA et TB.

Ci-dessus, chaque ligne de la table TA a été associée à chaque ligne de la table TB. En général on veut associer à une ligne de TA, les lignes de TB qui ont **une relation** avec elle. Cette relation prend souvent la forme d'une contrainte de clé étrangère. C'est le cas ici. A une ligne de la table TA, on peut associer les lignes de la table TB qui vérifient la relation TB.IDTA=TA.ID. Il y a plusieurs façons de demander cela :

```
SQL>select TA.ID, TA.data, TB.valeur, TB.IDTA FROM TA, TB where TA.ID=TB.IDTA
```

L'ordre SQL précédent est analogue au précédent avec cependant deux différences :

- les lignes résultat du produit cartésien TA x TB sont filtrées par une clause WHERE qui associe à une ligne de la table TA, les seules lignes de la table TB qui vérifient la relation TB.IDTA=TA.ID

•on ne demande que certaines colonnes avec la syntaxe [T.col] où T est le nom d'une table et col le nom d'une colonne de cette table. Cette syntaxe permet de lever l'ambiguïté qui pourrait surgir si deux tables avaient des colonnes de même nom. Lorsque cette ambiguïté n'existe pas, on peut utiliser la syntaxe [col] sans préciser la table de cette colonne.

Le résultat obtenu est le suivant :

ID	DATA	VALEUR	IDTA
2	data2	valeur1	2
1	data1	valeur2	1
2	data2	valeur3	2

Le même résultat peut être obtenu avec l'ordre SQL suivant :

```
SQL>select TA.ID, TA.data, TB.valeur, TB.IDTA FROM TA inner join TB on TA.ID=TB.idta
```

Du terme [inner join] vient le nom de "jointure interne" donné à ce type d'opérations entre deux tables. On verra qu'il existe une "jointure externe". Dans une jointure interne, l'ordre des tables dans la requête n'a pas de conséquences sur le résultat : **FROM TA inner join TB** est équivalent à **FROM TB inner join TA**.

L'ordre SQL précédent ne met dans la table résultante que les lignes de la table TA référencées par au moins une ligne de la table TB. Ainsi la ligne de TA [3, data3] n'apparaît pas dans le résultat car elle n'est pas référencée par une ligne de TB. On peut vouloir toutes les lignes de TA, qu'elles soient ou non référencées par une ligne de TB. On utilise alors une **jointure externe** entre les deux tables :

```
SQL> select TA.ID, TA.data, TB.valeur, TB.IDTA FROM TA left outer join TB on TA.ID= TB.IDTA
```

ID	DATA	VALEUR	IDTA
1	data1	valeur2	1
2	data2	valeur1	2
2	data2	valeur3	2
3	data3	<null>	<null>

On a ici une jointure externe gauche "**left outer join**". Pour comprendre le terme "FROM TA left outer join TB", il faut imaginer une jointure avec la table TA à gauche et la table TB à droite. Toutes les lignes de la table de gauche se retrouvent dans le résultat d'une jointure externe gauche même celles pour qui la relation de jointure n'est pas vérifiée. Cette relation de jointure n'est pas forcément une contrainte de clé étrangère même si c'est néanmoins le cas le plus courant.

Dans l'ordre suivant :

```
SQL> select TA.ID, TA.data, TB.valeur, TB.IDTA FROM TB left outer join TA on TA.ID= TB.IDTA
```

c'est la table TB qui est à "gauche" dans la jointure externe. On retrouvera donc toutes les lignes de TB dans le résultat :

ID	DATA	VALEUR	IDTA
2	data2	valeur1	2
1	data1	valeur2	1
2	data2	valeur3	2

Contrairement à la jointure interne, l'ordre des tables a donc une importance. Il existe également des jointures externes droites :

- FROM TA left outer join TB est équivalent à FROM TB right outer join TA : la table TA est à gauche
- FROM TB left outer join TA est équivalent à FROM TA right outer join TB : la table TB est à gauche

Les bases de l'exploitation simultanée de plusieurs tables étant maintenant connues, nous pouvons aborder des opérations de consultation plus complexes sur les bases de données.

6 Approfondissement du langage SQL

6.1 Introduction

Dans ce chapitre nous présentons

- d'autres syntaxes de la commande SELECT qui en font une commande de consultation très puissante notamment pour consulter plusieurs tables à la fois.
- des syntaxes élargies de commandes déjà étudiées

Pour illustrer les diverses commandes, nous travaillerons avec les tables suivantes utilisées pour la gestion des commandes dans une PME de diffusion de livres :

6.1.1 la table CLIENTS

Elle mémorise des informations sur les clients de la PME :

PK	Field Name	Field Type	D..	Size
1	ID	INTEGER		
	NOM	VARCHAR		30
	STATUT	CHAR		1
	PRENOM	VARCHAR		20
	CONTACT	VARCHAR		30
	RUE	VARCHAR		25
	VILLE	VARCHAR		20
	CPOSTAL	CHAR		5
	TELEPH	VARCHAR		20
	DEPUIS	DATE		
	DEBITEUR	CHAR		1

ID	NOM	STATUT	PRENOM	CONTACT	RUE	VILLE	CPOSTAL	TELEPH	DEPUIS	DEBITEUR
1	Librairie La Comète	E	<null>	<null>	<null>	Angers	49000	<null>	01.03.1988 00:00	N
2	Librairie du Marché	E	<null>	<null>	<null>	Angers	49100	<null>	01.04.1989 00:00	O
3	Trésor Public	A	<null>	<null>	<null>	Angers	49000	<null>	01.01.1987 00:00	N
4	Mairie de Saumur	A	<null>	<null>	<null>	Saumur	49700	<null>	01.02.1978 00:00	O
5	Netlogos	E	<null>	<null>	<null>	Segré	49500	<null>	01.10.1977 00:00	N
6	Barnard	I	<null>	<null>	<null>	Cholet	49800	<null>	01.09.1977 00:00	N
7	Préfecture du M-L	A	<null>	<null>	<null>	Angers	49000	<null>	10.12.1966 00:00	O
8	Artimon	E	<null>	<null>	<null>	Avrillé	49350	<null>	14.06.1988 00:00	N
9	Gazprom	E	<null>	<null>	<null>	Saumur	49550	<null>	23.07.1990 00:00	N
10	Pluchot	I	<null>	<null>	<null>	Segré	49100	<null>	21.02.1989 00:00	O

`ID`	n° identifiant le client de façon unique - clé primaire
`NOM`	nom du client
`STATUT`	I=Individu, E=Entreprise, A=Administration
`PRENOM`	prénom dans le cas d'un individu
`CONTACT`	Nom de la personne à contacter chez le client (dans le cas d'une entreprise ou d'une administration)
`RUE`	Adresse du client - rue
`VILLE`	ville
`CPOSTAL`	code postal
`TELEPH`	Téléphone
`DEPUIS`	Client depuis quelle date ?
`DEBITEUR`	O (Oui) si le client doit de l'argent à l'entreprise et N (Non) sinon.

6.1.2 la table ARTICLES

Elle mémorise des informations sur les produits vendus, ici des livres. Sa structure est la suivante :

PK	Field Name	Field Type	Domain	Size
1	ISBN	CHAR		13
	TITRE	VARCHAR		30
	CODEDITEUR	CHAR		7
	AUTEUR	VARCHAR		30
	RESUME	VARCHAR		400
	QTEANCOUR	INTEGER		
	QTEANPREC	INTEGER		
	DERNVENTE	DATE		
	QTERECUE	INTEGER		
	DERNLIV	DATE		
	PRIXVENTE	NUMERIC		6
	COUT	NUMERIC		6
	MINCDE	INTEGER		
	MINSTOCK	INTEGER		
	QTESTOCK	INTEGER		

ISBN	n° identifiant un livre de façon unique (ISBN= International Standard Book Number) - clé primaire
TITRE	Titre du livre
CODEDITEUR	Code identifiant un éditeur de façon unique
AUTEUR	Nom de l'auteur
RESUME	Résumé du livre
QTEANCOUR	Quantité vendue dans l'année
QTEANPREC	Quantité vendue l'année précédente
DERNVENTE	date de la dernière vente
QTERECUE	Quantité de la dernière livraison
DERNLIV	Date de la dernière livraison
PRIXVENTE	Prix de vente
COUT	Coût d'achat
MINCDE	Quantité minimale à commander
MINSTOCK	Seuil minimal du stock
QTESTOCK	Quantité en stock

Son contenu pourrait être le suivant :

TITRE	ISBN	C..	A..	R..	QTEANCOUR	QTEANPREC	DERNVENTE	QTERE...	D..	PRIXVENTE	COUT	MINCDE	MINSTOCK	QTESTOCK
Using 1-2-3 release 3	0-07-881309-3	<nu	<nu	<nu	1 000	5 000	01.09.1991 00:00	100	<nu	200,00	180,00	100	30	430
DOS - The complete reference	0-07-881497-9	<nu	<nu	<nu	670	8 000	14.09.1991 00:00	500	<nu	340,00	270,00	100	100	780
Using Quick Pascal	0-07-881520-7	<nu	<nu	<nu	150	600	14.09.1991 00:00	80	<nu	280,00	230,00	40	40	200
Using SQL	0-07-881524-X	<nu	<nu	<nu	800	1 000	06.09.1991 00:00	40	<nu	320,00	250,00	50	30	210
Using Windows 3	0-07-881537-1	<nu	<nu	<nu	45	0	18.09.1991 00:00	50	<nu	450,00	400,00	30	15	67
Dvorak's guide to telecoms	0-07-881551-7	<nu	<nu	<nu	100	500	14.02.1990 00:00	50	<nu	250,00	200,00	50	20	100

6.1.3 la table COMMANDES

Elle enregistre les informations sur les commandes faites par les clients. Sa structure est la suivante :

FK	PK	Field Name	Field Type	Domain	Size
	1	NOCMD	INTEGER		
F		IDCLI	INTEGER		
		DATECMD	DATE		
		ANNULE	CHAR		1

NOCMD	N° identifiant une commande de façon unique - clé primaire
IDCLI	N° du client faisant cette commande - clé étangère - référence CLIENTS(ID)
DATE_CMD	Date de saisie de cette commande
ANNULE	O (Oui) si la commande a été annulée et N (Non) sinon.

NOCMD	IDCLI	DATECMD	ANNULE
1	3	21.09.1991 00:00	N
2	2	21.09.1991 00:00	N
3	5	23.09.1991 00:00	N
4	6	24.09.1991 00:00	N
5	2	25.09.1991 00:00	N
7	4	25.09.1991 00:00	N
9	5	26.09.1991 00:00	N
10	3	26.09.1991 00:00	N

6.1.4 la table DETAILS

Elle contient les détails d'une commande, c'est à dire les références et quantités des livres commandés. Sa structure est la suivante :

FK	PK	Field Name	Field Type	Domain	Size
	1	ID	INTEGER		
F		NOCMD	INTEGER		
F		ISBN	CHAR		13
		QTE	INTEGER		

`NOCMD` N° de la commande - clé étrangère référençant la colonne NOCMD de la table COMMANDES
`ISBN` N° du livre commandé - clé étrangère référençant la colonne ISBN de la table LIVRES
`QTE` Quantité commandée

Son contenu pourrait être le suivant :

ID	NOCMD	ISBN	QTE
1	1	0-07-881551-7	2
2	1	0-07-881524-X	1
3	2	0-07-881537-1	4
4	3	0-07-881520-7	1
5	3	0-07-881551-7	1
6	3	0-07-881497-9	4
7	4	0-07-881551-7	2
8	4	0-07-881309-3	1
9	5	0-07-881309-3	1
10	7	0-07-881551-7	1
11	9	0-07-881524-X	3
12	10	0-07-881497-9	12
13	10	0-07-881520-7	2

Ci-dessus, on voit que la commande n° 3 (NOCMD) concerne trois livres. Cela veut dire que le client a commandé trois livres en même temps. Les références de ce client peuvent être trouvées dans la table [COMMANDES] où l'on voit que la commande n° 3 a été effectuée par le client n° 5. La table [CLIENTS] nous apprend que le client n° 5 est la société NetLogos de Segré.

6.2 La commande SELECT

Nous nous proposons ici d'approfondir notre connaissance de la commande SELECT en présentant de nouvelles syntaxes de celle-ci.

6.2.1 Syntaxe d'une requête multi-tables

`syntaxe`
SELECT *colonne1, colonne2, ...*
FROM *table1, table2, ..., tablep*
WHERE *condition*
ORDER BY ...

`action` La nouveauté ici vient du fait que les colonnes *colonne1, colonne2, ...* proviennent de plusieurs tables *table1, table2, ...* Si deux tables ont des colonnes de même nom, on lève l'ambigüité par la notation *tablei.colonnej*. La *condition* peut porter sur les colonnes des différentes tables.

Fonctionnement

1. La table produit cartésien de *table1, table2, ..., tablep* est réalisée. Si *ni* est le nombre de lignes de *tablei*, la table construite a donc *n1*n2*...*np* lignes comportant l'ensemble des colonnes des différentes tables.
2. La *condition* du **WHERE** est appliquée à cette table. Une nouvelle table est ainsi produite

3 Celle-ci est ordonnée selon le mode indiqué dans **ORDER**.
4 Les colonnes demandées derrière **SELECT** sont affichées.

Exemples

On utilise les tables présentées précédemment. On veut connaître le détail des commandes passées après le 25 septembre :

```
SQL>select details.nocmd,isbn,qte from commandes,details
  where commandes.datecmd>'25-sep-91'
  and details.nocmd=commandes.nocmd
```

NOCMD	ISBN	QTE
9	0-07-881524-X	3
10	0-07-881497-9	12
10	0-07-881520-7	2

On remarquera que derrière FROM, on met le nom de toutes les tables dont on référence les colonnes. Dans l'exemple précédent, les colonnes sélectionnées appartiennent toutes à la table DETAILS. Cependant la condition fait référence à la table COMMANDES. D'où la nécessité de nommer cette dernière derrière le FROM. L'opération qui teste l'égalité de colonnes de deux tables différentes est souvent appelée une équi-jointure.

La requête SELECT aurait pu être également écrite de la façon suivante :

```
SQL> select details.nocmd,isbn,qte from commandes
    inner join details on details.nocmd=commandes.nocmd
    where commandes.datecmd>'25-sep-91'
```

Continuons nos exemples. On désire le même résultat que précédemment mais avec le titre du livre commandé, plutôt que son n° ISBN :

```
SQL>select commandes.nocmd, articles.titre, details.qte
  from commandes,articles,details
  where commandes.datecmd>'25-sep-91'
  and details.nocmd=commandes.nocmd
  and details.isbn=articles.isbn
```

NOCMD	TITRE	QTE
9	Using SQL	3
10	DOS - The complete reference	12
10	Using Quick Pascal	2

Le même résultat est obtenu avec la requête SQL suivante, moins lisible :

```
SQL> select details.nocmd,articles.titre,details.qte from details
    inner join commandes on details.nocmd=commandes.nocmd
    inner join articles on  details.isbn=articles.isbn
    where commandes.datecmd>'25-sep-91'
```

Ci-dessus, deux jointures internes sont faites avec la table [DETAILS] :

- l'une avec la table [COMMANDES] pour avoir accès à la date de commande d'un livre
- l'une avec la table [ARTICLES] pour avoir accès au titre du livre commandé

On veut de plus le nom du client qui fait la commande :

```
SQL>select commandes.nocmd, articles.titre, qte ,clients.nom
  from commandes,details,articles,clients
  where commandes.datecmd>'25-sep-91'
  and details.nocmd=commandes.nocmd
  and details.isbn=articles.isbn
  and commandes.idcli=clients.id
```

NOCMD	TITRE	QTE	NOM
9	Using SQL	3	Netlogos
10	DOS - The complete reference	12	Trésor Public
10	Using Quick Pascal	2	Trésor Public

On veut de plus les dates de commande et un affichage par ordre décroissant de ces dates :

```
SQL>select commandes.nocmd, commandes.datecmd, articles.titre, qte ,clients.nom
```

```
    from commandes,details,articles,clients
    where commandes.datecmd>'25-sep-91'
    and details.nocmd=commandes.nocmd
    and details.isbn=articles.isbn
    and commandes.idcli=clients.id
    order by commandes.datecmd descending
```

NOCMD	DATECMD	TITRE	QTE	NOM
9	26.09.1991 00:00	Using SQL	3	Netlogos
10	26.09.1991 00:00	DOS - The complete reference	12	Trésor Public
10	26.09.1991 00:00	Using Quick Pascal	2	Trésor Public

Voici quelques règles à observer dans les jointures :

1. Derrière SELECT, on met les colonnes que l'on désire obtenir à l'affichage. Si la colonne existe dans diverses tables, on la fait précéder du nom de la table.
2. Derrière FROM, on met toutes les tables qui seront explorées par le SELECT, c'est à dire les tables propriétaires des colonnes qui se trouvent derrière SELECT et WHERE.

6.2.2 L'auto-jointure

On veut connaître les livres qui ont un prix de vente supérieur à celui du livre 'Using SQL' :

```
SQL>select a.titre from articles a, articles b
  where b.titre='Using SQL'
  and a.prixvente>b.prixvente
```

TITRE
DOS - The complete reference
Using Windows 3

Les deux tables de la jointure sont ici identiques : la table *articles*. Pour les différencier, on leur donne un alias : *from articles a, articles b.* L'alias de la première table s'appelle *a* et celui de la seconde, *b.* Cette syntaxe peut s'utiliser même si les tables sont différentes. Lors de l'utilisation d'un alias, celui-ci doit être utilisé partout dans la commande SELECT en lieu et place de la table qu'il désigne.

6.2.3 Jointure externe

On veut connaître les clients qui ont acheté quelque chose en septembre avec la date de la commande. Les autres clients sont affichés sans cette date :

```
SQL>select clients.nom,commandes.datecmd from clients
    left outer join commandes  on clients.id=commandes.idcli
    where datecmd between '01-sep-91' and '30-sep-91'
```

NOM	DATECMD
Librairie du Marché	21.09.1991 00:00
Librairie du Marché	25.09.1991 00:00
Trésor Public	21.09.1991 00:00
Trésor Public	26.09.1991 00:00
Mairie de Saumur	25.09.1991 00:00
Netlogos	23.09.1991 00:00
Netlogos	26.09.1991 00:00
Barnard	24.09.1991 00:00

On est étonné ici de ne pas avoir le bon résultat. On devrait avoir tous les clients présents dans la table [CLIENTS], ce qui n'est pas le cas. Lorsqu'on réfléchit au fonctionnement de la jointure externe, on réalise que les clients n'ayant pas acheté ont été associées à une ligne vide de la table COMMANDES et donc avec une date vide (valeur NULL dans la terminologie SQL). Cette date ne vérifie pas alors la condition fixée sur la date et le client correspondant n'est pas affiché. Essayons autre chose :

```
SQL>select clients.nom,commandes.datecmd from clients
    left outer join commandes  on clients.id=commandes.idcli
    where (commandes.datecmd between '01-sep-91' and '30-sep-91')
        or (commandes.datecmd is null)
```

NOM	DATECMD
Librairie La Comète	<null>
Librairie du Marché	21.09.1991 00:00
Librairie du Marché	25.09.1991 00:00
Trésor Public	21.09.1991 00:00
Trésor Public	26.09.1991 00:00
Mairie de Saumur	25.09.1991 00:00
Netlogos	23.09.1991 00:00
Netlogos	26.09.1991 00:00
Barnard	24.09.1991 00:00
Préfecture du M-L	<null>
Artimon	<null>
Mecafluid	<null>
Pluchot	<null>

On obtient cette fois-ci la réponse correcte à notre question.

6.2.4 Requêtes imbriquées

`syntaxe`

SELECT *colonne[s]* FROM *table[s]*
WHERE *expression opérateur requête*
ORDER BY ...

`fonctionnement`

requête est une commande SELECT qui délivre un groupe de 0, 1 ou plusieurs valeurs. On a alors une condition **WHERE** du type

expression opérateur (val1, val2, ..., vali)

expression et *vali* doivent être de même type. Si la requête délivre une seule valeur, on est ramené à une condition du type

expression opérateur valeur

que nous connaissons bien. Si la requête délivre une liste de valeurs, on pourra employer les opérateurs suivants :

`IN`

expression **IN** *(val1, val2, ..., vali)* **:** **vraie** si *expression* a pour valeur l'un des éléments de la liste *vali.*

`NOT IN`

inverse de **IN**

`ANY`

doit être précédé de =,!=,>,>=,<,<=
expression >= **ANY** *(val1, val2, .., valn)* **:** vraie si *expression* est >= à l'une des valeurs *vali* de la liste

`ALL`

doit être précédé de =,!=,>,>=,<,<=
expression >= **ALL** *(val1, val2, .., valn)* : **vraie** si expression est >= à toutes les valeurs vali de la liste

`EXISTS`

requête : vraie si la *requête* rend au moins une ligne.

Exemples

On reprend la question déjà résolue par une équi-jointure : afficher les titres ayant un prix de vente supérieur à celui du livre 'Using SQL'.

```
SQL>select titre from ARTICLES
    where prixvente > (select prixvente from ARTICLES where titre='Using SQL')
```

TITRE
DOS - The complete reference
Using Windows 3

Cette solution semble plus intuitive que celle de l'équi-jointure. On fait un premier filtrage avec un SELECT, puis un second sur le résultat obtenu. On peut opérer ainsi plusieurs filtrages en série.

On veut connaître les titres ayant un prix de vente supérieur au prix moyen de vente :

```
SQL> select titre from ARTICLES
    where prixvente > (select avg(prixvente) from ARTICLES)
```

TITRE
Using SQL
DOS - The complete reference
Using Windows 3

Quels sont les clients ayant commandé les titres résultat de la requête précédente ?

```
SQL>select distinct idcli from COMMANDES,DETAILS
    where DETAILS.isbn in
    (select isbn from ARTICLES where prixvente
        > (select avg(prixvente) from ARTICLES))
    and COMMANDES.nocmd=DETAILS.nocmd
```

IDCLI
2
3
5

Explications

a) on sélectionne dans la table DETAILS les codes ISBN se trouvant parmi les livres ayant un prix supérieur au prix moyen des livres.
b) dans les lignes sélectionnées de l'étape précédente, il n'y a pas le code client IDCLI. Il se trouve dans la table COMMANDES. Le lien entre les deux tables se fait par le n° de commande NOCMD, d'où l'équi-jointure COMMANDES.nocmd=DETAILS.nocmd.
c) Un même client peut avoir acheté plusieurs fois l'un des livres concernés, auquel cas on aura son code IDCLI plusieurs fois. Pour éviter cela, on met le mot clé DISTINCT derrière SELECT. DISTINCT de façon générale élimine les doublons dans les lignes résultat d'un SELECT.
d) pour avoir le nom du client, il nous faudrait faire une équi-jointure supplémentaire entre les tables COMMANDES et CLIENTS comme le montre la requête qui suit.

```
SQL> select distinct CLIENTS.nom from COMMANDES,DETAILS,CLIENTS
    where DETAILS.isbn in
    (select isbn from ARTICLES where prixvente
        > (select avg(prixvente) from ARTICLES))
    and COMMANDES.nocmd=DETAILS.nocmd
    and COMMANDES.IDCLI=CLIENTS.ID
```

NOM
Librairie du Marché
Netlogos
Trésor Public

Trouver les clients qui n'ont pas fait de commande depuis le 24 septembre :

```
SQL>select nom from CLIENTS
    where clients.id not in
    (select distinct commandes.idcli from commandes where datecmd>='24-sep-91')
```

NOM
Librairie La Comète
Préfecture du M-L
Artimon
Mecafluid
Pluchot

Nous avons vu qu'on pouvait filtrer des lignes autrement qu'avec la clause WHERE : en utilisant la clause HAVING en conjonction avec la clause GROUP BY. La clause HAVING **filtre des groupes de lignes**.

De la même façon que pour la clause WHERE, la syntaxe
HAVING *expression opérateur requête*
est possible, avec la contrainte déjà présentée que *expression* doit être l'une des expressions *expri* de la clause
GROUP BY *expr1, expr2, ...*

Exemples

Quelles sont les quantités vendues pour les livres de plus de 200F ?

Affichons tout d'abord les quantités vendues par titre :

```
SQL>select ARTICLES.titre,sum(qte) QTE from ARTICLES, DETAILS
    where DETAILS.isbn=ARTICLES.isbn
    group by titre
```

TITRE	QTE
DOS - The complete reference	16
Dvorak's guide to telecoms	6
Using 1-2-3 release 3	2
Using Quick Pascal	3
Using SQL	4
Using Windows 3	4

Maintenant, filtrons les titres :

```
SQL> select ARTICLES.titre,sum(qte) QTE from ARTICLES, DETAILS
    where DETAILS.isbn=ARTICLES.isbn
    group by titre
    having titre in (select titre from ARTICLES where prixvente>200)
```

TITRE	QTE
DOS - The complete reference	16
Dvorak's guide to telecoms	6
Using Quick Pascal	3
Using SQL	4
Using Windows 3	4

De façon peut-être plus évidente on aurait pu écrire :

```
SQL>select ARTICLES.titre,sum(qte) QTE from ARTICLES, DETAILS
    where DETAILS.isbn=ARTICLES.isbn
    and ARTICLES.prixvente>200
    group by titre
```

TITRE	QTE
DOS - The complete reference	16
Dvorak's guide to telecoms	6
Using Quick Pascal	3
Using SQL	4
Using Windows 3	4

6.2.5 Requêtes corrélées

Dans le cas des requêtes imbriquées, on a une requête parent (la requête la plus externe) et une requête fille (la requête la plus interne). La requête mère n'est évaluée que lorsque la requête fille l'a été complètement.

Les requêtes corrélées ont la même syntaxe au détail près suivant : la requête fille fait une jointure sur la table de la requête mère. Dans ce cas, l'ensemble requête mère-requête fille est évalué de façon répétée pour chaque ligne de la table mère.

Exemple

Nous reprenons l'exemple où nous désirons les noms des clients n'ayant pas fait de commande depuis le 24 septembre :

```
SQL>
select nom from clients
    where not exists
        (select idcli from commandes
            where datecmd>='24-sep-91'
                and commandes.idcli=clients.id)
```

NOM
Librairie La Comète
Préfecture du M-L
Artimon
Mecafluid
Pluchot

La requête mère s'exerce sur la table *clients*. La requête fille fait une jointure entre les tables *clients* et *commandes*. On a donc une requête corrélée. Pour chaque ligne de la table *clients*, la requête fille s'exécute : elle cherche le code *id* du client dans les commandes faites après le 24 septembre. Si elle n'en trouve pas (*not exists*), le nom du client est affiché. Ensuite, on passe à la ligne suivante de la table *clients*.

6.2.6 Critères de choix pour l'écriture du SELECT

Nous avons vu, à plusieurs reprises, qu'il était possible d'obtenir un même résultat par différentes écritures du SELECT. Prenons un exemple : Afficher les clients ayant commandé quelque chose :

Jointure

```
SQL>
select distinct nom from clients,commandes
    where clients.id=commandes.idcli
```

NOM
Barnard
Librairie du Marché
Mairie de Saumur
Netlogos
Trésor Public

Requêtes imbriquées

```
SQL>
select nom from clients
    where id in (select idcli from commandes)
```

donne le même résultat.

Requêtes corrélées

```
SQL>
select nom from clients
    where exists (select * from commandes where commandes.idcli=clients.id)
```

donne le même résultat.

Les auteurs Christian MAREE et Guy LEDANT, dans leur livre ' SQL, Initiation, Programmation et Maîtrise' proposent quelques critères de choix :

Performances

L'utilisateur ne sait pas comment le SGBD "se débrouille" pour trouver les résultats qu'il demande. Ce n'est donc que par expérience, qu'il découvrira que telle écriture est plus performante qu'une autre. MAREE et LEDANT affirment par expérience que les requêtes corrélées semblent généralement plus lentes que les requêtes imbriquées ou les jointures.

Formulation

La formulation par requêtes imbriquées est souvent plus lisible et plus intuitive que la jointure. Elle n'est cependant pas toujours utilisable. Deux points sont notamment à observer :

- Les tables propriétaires des colonnes arguments du SELECT (SELECT col1, col2, ...) doivent être nommées derrière le mot clé FROM. Le produit cartésien de ces tables est alors effectué, ce qu'on appelle une jointure.
- Lorsque la requête affiche des résultats provenant d'**une seule table**, et que le filtrage des lignes de cette dernière impose la consultation d'une autre table, les requêtes imbriquées peuvent être utilisées.

6.3 Extensions de syntaxe

Pour des questions de commodité, nous avons le plus souvent présenté des syntaxes réduites des différentes commandes. Dans cette section, nous en présentons des syntaxes élargies. Elles se comprennent d'elles-mêmes car elles sont analogues à celles de la commande SELECT largement étudiée.

INSERT

`syntaxe1` **INSERT INTO** *table (col1, col2, ..)* **VALUES** *(val1, val2, ...)*
`syntaxe2` **INSERT INTO** *table (col1, col2, ..) (requête)*
`explication` Ces deux syntaxes ont été présentées

DELETE

`syntaxe1` **DELETE FROM** *table* **WHERE** *condition*
`explication` Cette syntaxe est connue. Ajoutons que la condition peut contenir une requête avec la syntaxe *WHERE expression opérateur (requête)*

UPDATE

`syntaxe1` **UPDATE** *table*
SET *col1=expr1, col2=expr2, ...*
WHERE *condition*

`explication` Cette syntaxe a déjà été présentée. Ajoutons que la condition peut contenir une requête avec la syntaxe **WHERE** *expression opérateur (requête)*

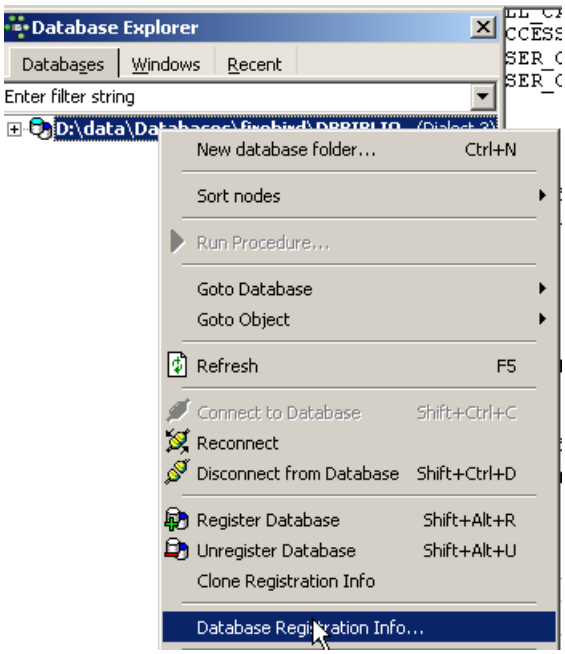
`syntaxe2` **UPDATE** *table*
SET *(col1, col2, ..)= requête1*, *(cola, colb, ..)= requête2*, ...
WHERE *condition*
`explication` Les valeurs affectées aux différentes colonnes peuvent provenir d'une requête.

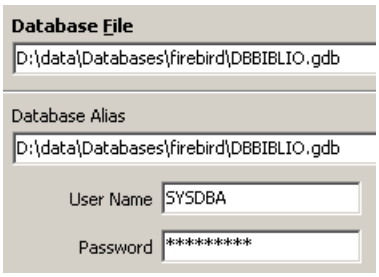
7 Gestion de l'accès concurrent aux données

Nous avons jusqu'à maintenant utilisé des tables dont nous étions les seuls utilisateurs. Dans la pratique, sur une machine multi-utilisateurs, les données sont le plus souvent partagées entre différents utilisateurs. Se pose alors la question : Qui peut utiliser telle ou telle table et sous quelle forme (consultation, insertion, suppression, ajout, ...) ?

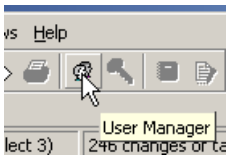
7.1 Création d'utilisateurs Firebird

Lorsque nous avons travaillé avec IB-Expert, nous nous sommes connectés en tant qu'utilisateur SYSDBA. On peut retrouver cette information dans les propriétés de la connexion ouverte au SGBD :

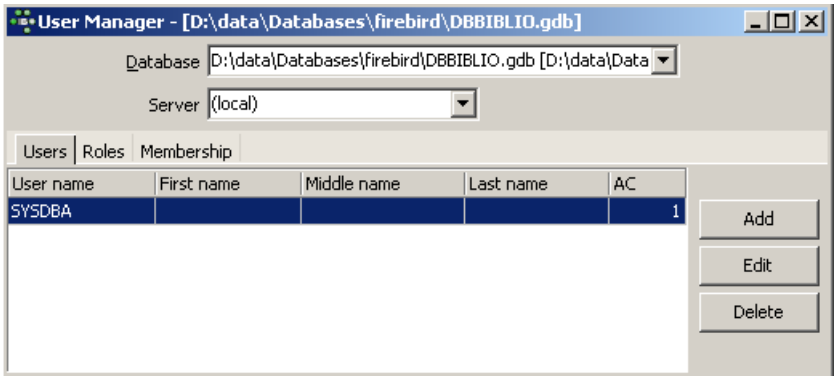




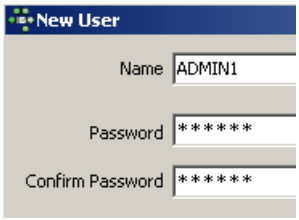
A droite, on voit que l'utilisateur connecté est [SYSDBA]. Ce qu'on ne voit pas c'est son mot de passe [masterkey]. [SYSDBA] est un utilisateur particulier de Firebird : il a tous les droits sur tous les objets gérés par le SGBD. On peut créer de nouveaux utilisateurs avec IBExpert avec l'option [Tools / User Manager] ou l'icône suivante :



Nous obtenons la fenêtre de de gestion des utilisateurs :

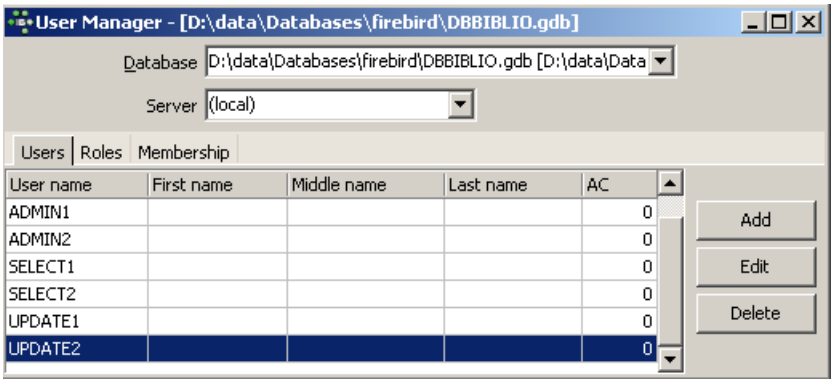


Le bouton [Add] permet de créer de nouveaux utilisateurs :



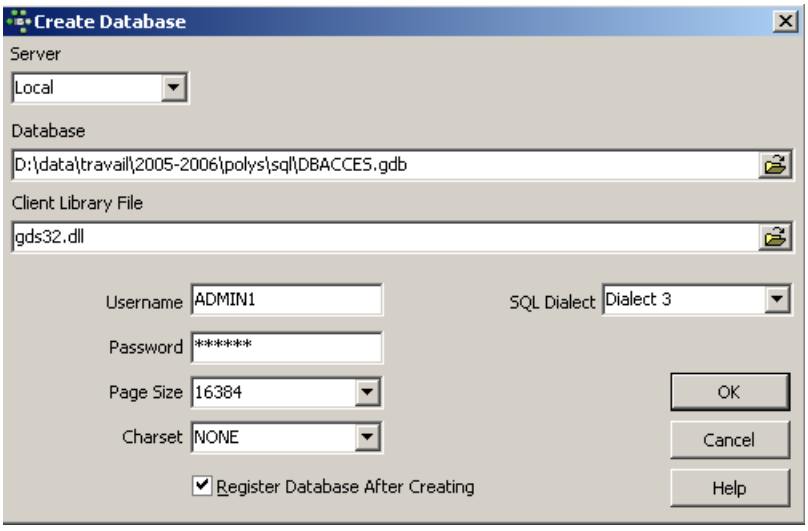
Créons ainsi les utilisateurs suivants :

nom	mot de passe
ADMIN1	admin1
ADMIN2	admin2
SELECT1	select1
SELECT2	select2
UPDATE1	update1
UPDATE2	update2

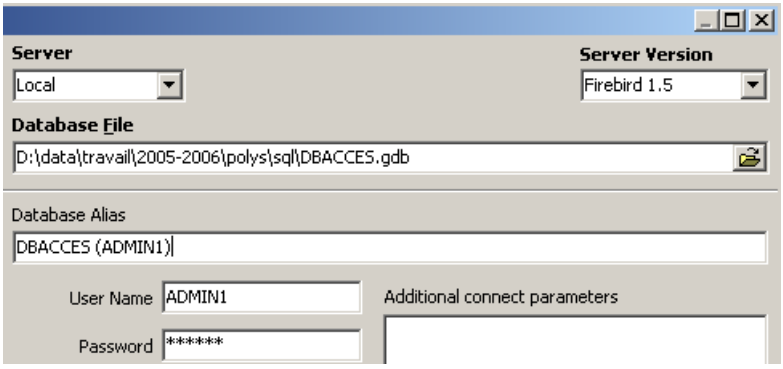


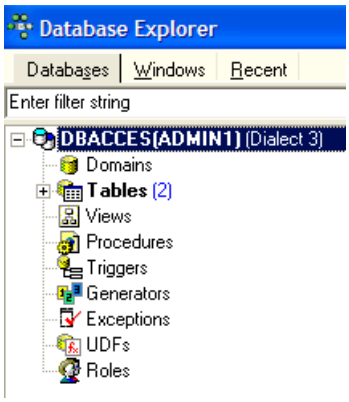
7.2 Accorder des droits d'accès aux utilisateurs

Une base appartient à celui qui l'a créée. Les bases que nous avons créées jusqu'à maintenant appartenaient à l'utilisateur [SYSDBA]. Pour illustrer la notion de droits, créons (Database / Create Database) une nouvelle base de données sous l'identité [ADMIN1, admin1] :



et enregistrons-la avec l'alias DBACCES (ADMIN1). L'utilisation d'alias permet d'ouvrir des connexions sur une même base en leur donnant des identifiants différents, ce qui permet de mieux les repérer dans l'explorateur de bases d'IBExpert. :





Créons maintenant les deux tables TA et TB suivantes :

Table TA

Field Name	Field Type	Domain	Size
ID	INTEGER		
DATA	VARCHAR		10

ID	DATA
1	data1
2	data2
3	data3

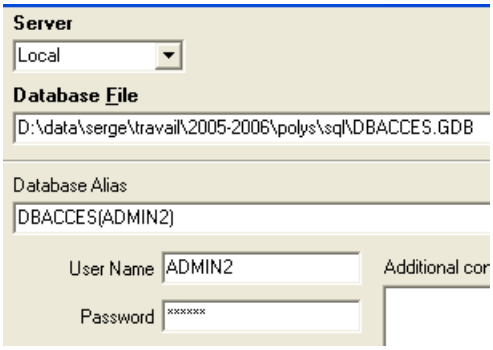
Table TB

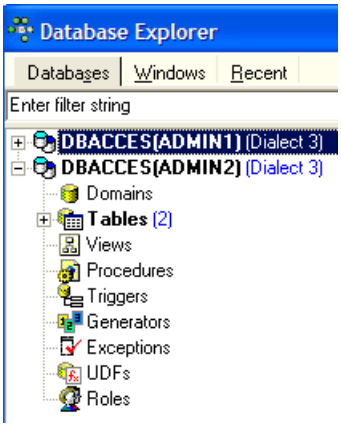
Field Name	Field Type	Domain	Size
ID	INTEGER		
VALEUR	VARCHAR		10

ID	VALEUR
1	valeur1
2	valeur2
3	valeur3

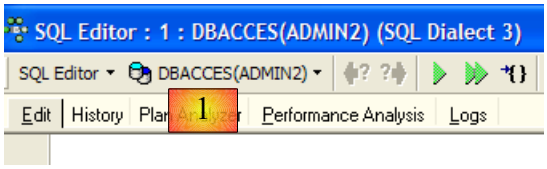
Ces tables n'ont pas de lien entre elles.

Avec IB-Expert, créons une seconde connexion à la base [DBACCES], cette fois-ci sous le nom [ADMIN2 / admin2]. Nous utilisons pour cela l'option [Database / Register Database] :



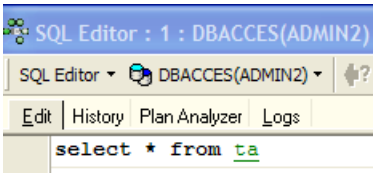


Positionnons-nous sur DBACCES(ADMIN2) et ouvrons un éditeur SQL (Shift + F12) :

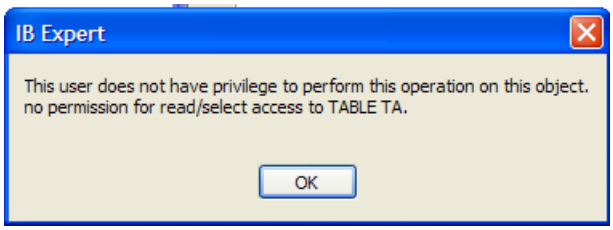


Nous aurons l'occasion d'utiliser diverses connexions sur la même base [DBACCES]. Pour chacune d'entre-elles, nous aurons un éditeur SQL. En [1], l'éditeur SQL indique l'alias de la base connectée. Utilisez cette indication pour savoir dans quel éditeur SQL vous êtes. Cela va avoir son importance car nous allons créer des connexions qui n'auront pas les mêmes droits d'accès sur les objets de la base.

Demandons le contenu de la table TA :



Nous obtenons le message d'erreur suivant :



Que signifie-t-il ? La base [DBACCESS] a été créée par l'utilisateur [ADMIN1] et est donc sa propriété. Seul lui a accès aux différents objets de cette base. Il peut accorder des droits d'accès à d'autres utilisateurs avec la commande SQL GRANT. Celle-ci a diverses syntaxes. L'une d'elles est la suivante :

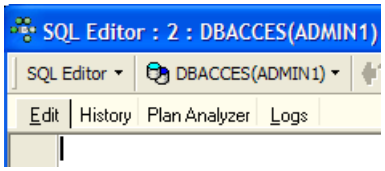
`syntaxe`
GRANT *privilège1, privilège2, ...* | **ALL PRIVILEGES**
ON *table/vue*
TO *utilisateur1, utilisateur2, ...* | **PUBLIC**
[WITH GRANT OPTION]

`action` accorde des privilèges d'accès *privilègei* ou tous les privilèges (**ALL PRIVILEGES**) sur la *table* ou *vue* aux utilisateurs *utilisateuri* ou à tous les utilisateurs (**PUBLIC**). La clause **WITH GRANT OPTION** permet aux utilisateurs ayant reçu les privilèges de les transmettre à leur tour à d'autres utilisateurs.

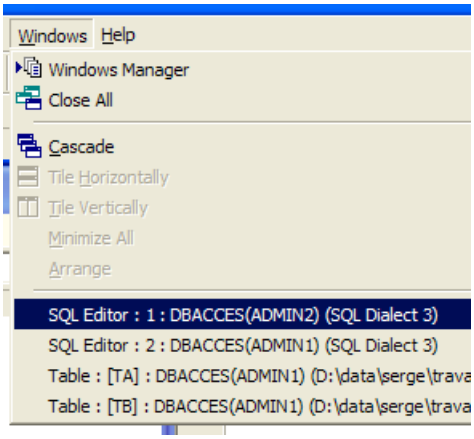
Parmi les privilèges *privilègei* qui peuvent être accordés se trouvent les suivants :

`DELETE` droit d'utiliser la commande DELETE sur la table ou vue.
`INSERT` droit d'utiliser la commande INSERT sur la table ou vue
`SELECT` droit d'utiliser la commande SELECT sur la table ou vue
`UPDATE` droit d'utiliser la commande UPDATE sur la table ou vue. Ce droit peut être restreint à certaines colonnes par la syntaxe : GRANT update **(col1, col2, ...) ON table/vue TO utilisateur1, utilisateur2, ...| PUBLIC [WITH GRANT OPTION]**

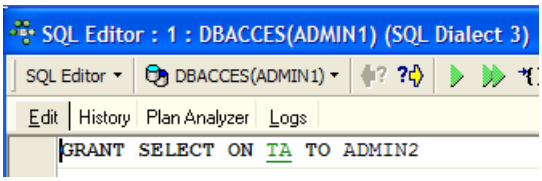
Donnons le droit à l'utilisateur [ADMIN2] le droit SELECT sur la table TA. Seul le propriétaire de la table peut donner ce droit, c.a.d. ici [ADMIN1]. Positionnons-nous sur la connexion DBACCES(ADMIN1) et ouvrons un nouvel éditeur SQL (Shift+F12) :



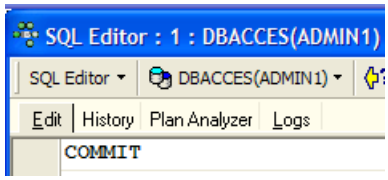
Par la suite, nous allons passer d'un éditeur SQL à l'autre. Pour s'y retrouver, on peut utiliser l'option [Windows] du menu :



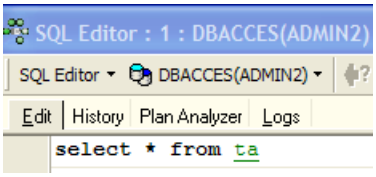
Ci-dessus, on voit les deux éditeurs SQL, chacun associé à un utilisateur particulier. Revenons sur l'éditeur SQL(ADMIN1) et émettons la commande suivante :



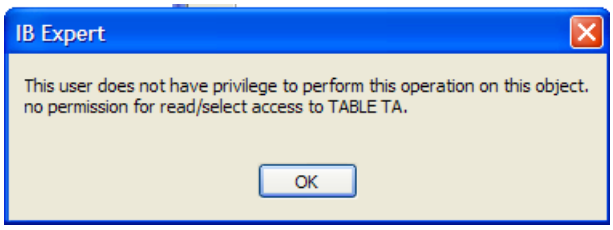
Puis validons-la par un COMMIT :



Ceci fait, passons dans l'éditeur de l'utilisateur ADMIN2 pour refaire le SELECT qui avait échoué :

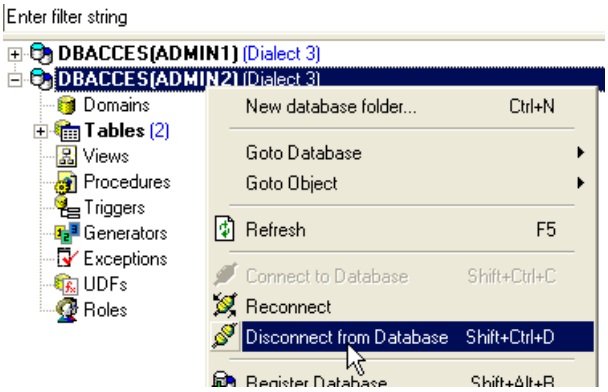


Nous obtenons le message d'erreur suivant :

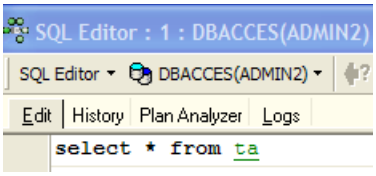


L'utilisateur [ADMIN2] n'a toujours pas le droit de consulter la table [TA]. En fait, il semble que les droits d'un utilisateur soient chargés au moment où il se connecte. [ADMIN2] aurait alors toujours les mêmes droits qu'au début de sa connexion, c'est à dire aucun. Vérifions-le. Déconnectons l'utilisateur [ADMIN2] :

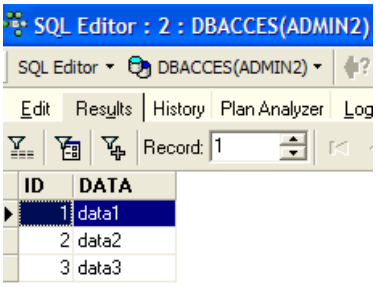
- se placer sur sa connexion
- demander la déconnexion en cliquant droit sur la connexion et en prenant l'option [Deconnect from database] ou (Shift + Ctrl + D)



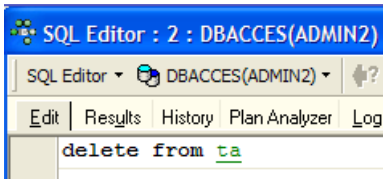
Si un panneau demande un [COMMIT], faites le [COMMIT]. Puis reconnectons l'utilisateur [ADMIN2] en prenant l'option [Reconnect] ci-dessus. Ceci fait, revenons sur l'éditeur SQL (ADMIN2) et rejouons la requête SELECT qui a échoué :

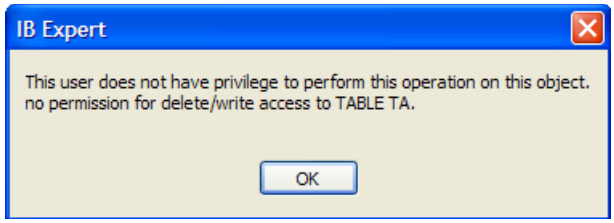


On obtient alors le résultat suivant :



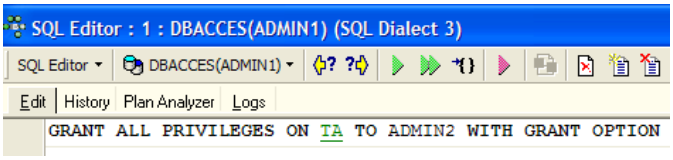
Cette fois-ci ADMIN2 peut consulter la table TA grâce au droit SELECT que lui a donné son propriétaire ADMIN1. Normalement c'est le seul droit qu'il a. Vérifions-le. Toujours dans l'éditeur SQL (ADMIN2) :

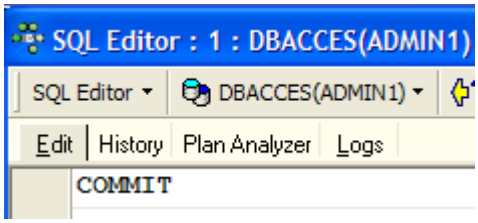




L'écran de droite montre que ADMIN2 n'a pas le droit DELETE sur la table TA.

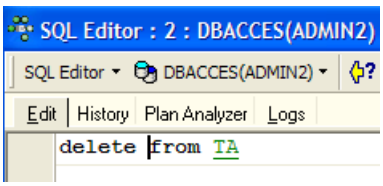
Revenons dans l'éditeur SQL(ADMIN1) pour donner davantage de droits à l'utilisateur ADMIN2. Nous émettons successivement les deux commandes suivantes :

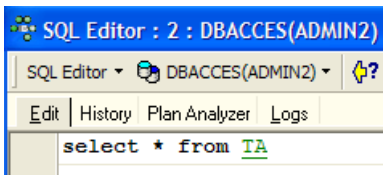


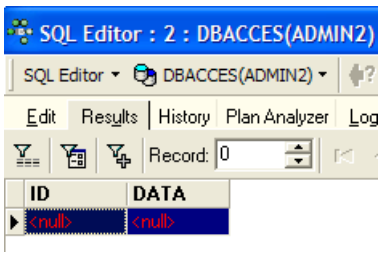


- la première commande donne à l'utilisateur ADMIN2 tous les droits d'accès à la table [TA] avec de plus la possibilité d'accorder lui aussi des droits (WITH GRANT OPTION)
- la deuxième commande valide la précédente

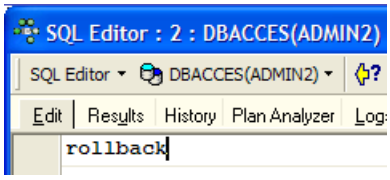
Ceci fait, comme fait précédemment renouvelons la connexion de l'utilisateur [ADMIN2] (Deconnect / Reconnect) puis dans l'éditeur SQL(ADMIN2) tapons les commandes suivantes :

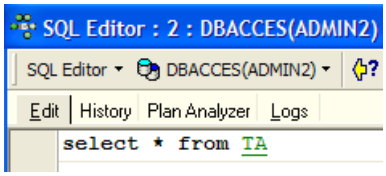


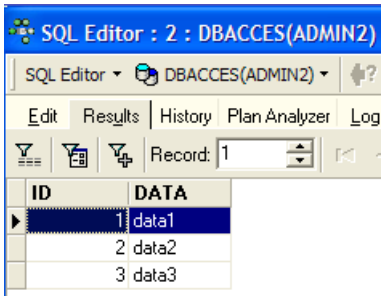




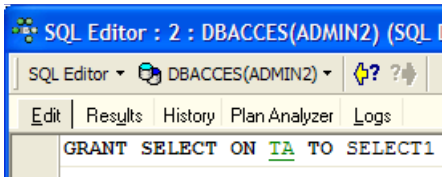
ADMIN2 a pu supprimer toutes les lignes de la table TA. Annulons cette suppression avec un ROLLBACK :

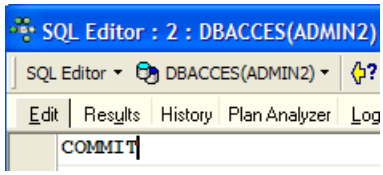




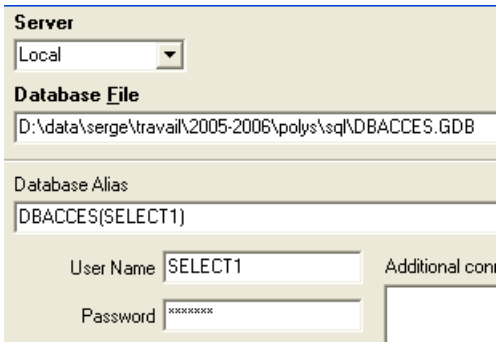


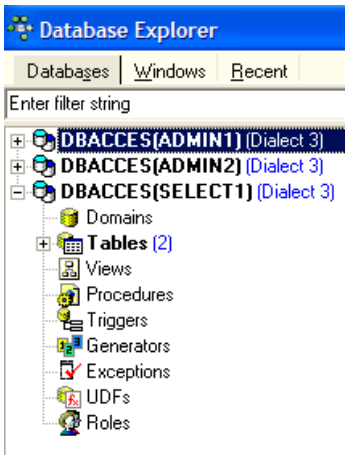
Vérifions que ADMIN2 peut à son tour donner des droits sur la table TA.



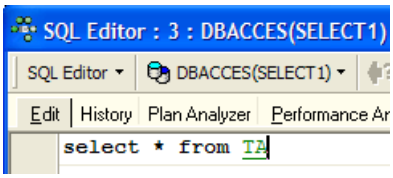


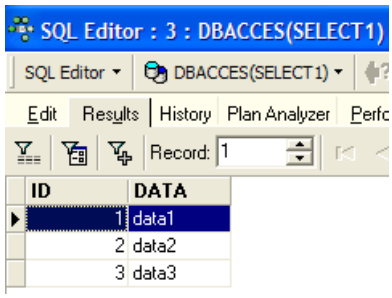
Ouvrons maintenant une connexion sur la base [DBACCES] (Database / Register database) sous le nom [SELECT1 / select1], l'un des utilisateurs créés précédemment puis double-cliquons sur le lien ainsi créé dans [Database Explorer] :



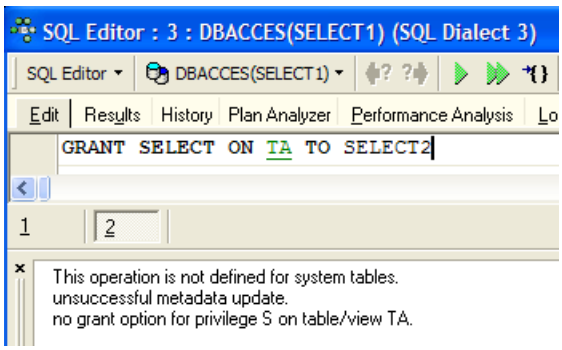


Positionnons-nous sur cette nouvelle connexion et ouvrons un nouvel éditeur SQL (Shift + F12) pour y taper les commandes suivantes :





L'utilisateur SELECT1 a bien le droit SELECT sur la table TA. A-t-il la possibilité de transmettre ce droit à l'utilisateur SELECT2 ?



L'opération a échoué parce que l'utilisateur SELECT1 n'a pas reçu le droit de transmettre le droit SELECT qu'il a reçu de l'utilisateur ADMIN2. Il aurait fallu pour cela que l'utilisateur ADMIN2 utilise la clause WITH GRANT OPTION dans son ordre SQL GRANT. Les règles de transmission sont simples :

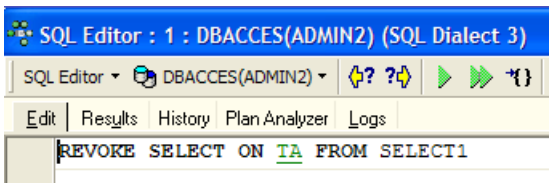
- un utilisateur ne peut transmettre que les droits qu'il a reçus et pas plus
- il ne peut les transmettre que s'il les a reçus avec le privilège [WITH GRANT OPTION]

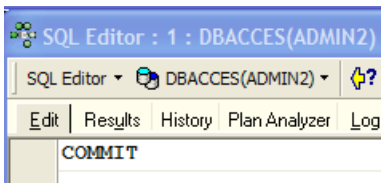
Un droit accordé peut être retiré avec l'ordre REVOKE :

`syntaxe` **REVOKE** *privilège1, privilège2, ...* | **ALL PRIVILEGES**
ON *table/vue*
FROM *utilisateur1, utilisateur2, ...* | **PUBLIC**

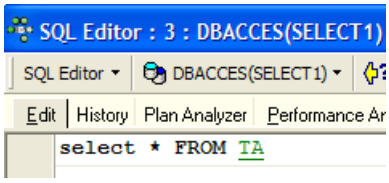
`action` supprime des privilèges d'accès *privilègei* ou tous les privilèges (**ALL PRIVILEGES**) sur la *table* ou *vue* aux utilisateurs *utilisateuri* ou à tous les utilisateurs (**PUBLIC**).

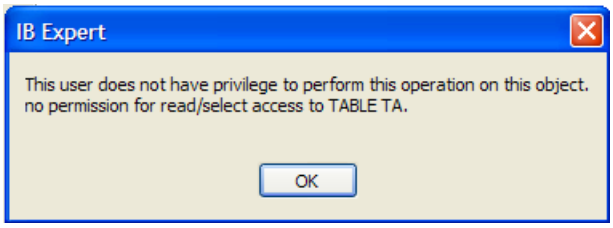
Essayons. Revenons dans l'éditeur SQL d'ADMIN2 pour enlever le droit SELECT que nous avons donné à l'utilisateur SELECT1 :





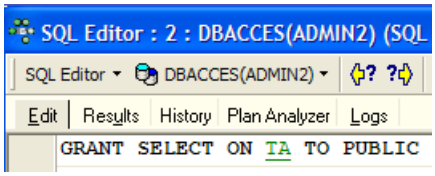
Déconnectons puis reconnectons la connexion de l'utilisateur SELECT1. Puis dans l'éditeur SQL (SELECT1) demandons le contenu de la table TA :

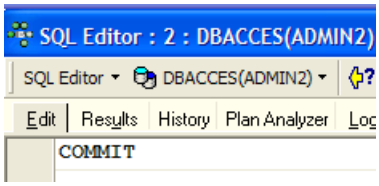




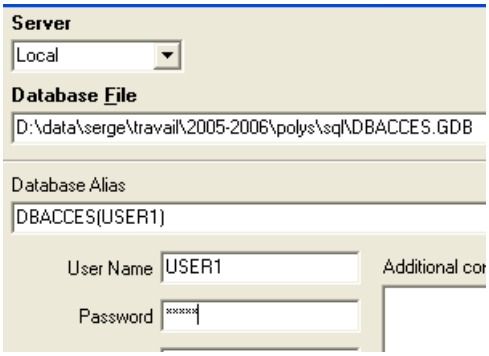
L'utilisateur SELECT1 a bien perdu son droit de lecture de la table TA. On notera que c'est ADMIN2 qui lui avait donné ce droit et c'est ADMIN2 qui le lui a retiré. Si ADMIN1 essaie de le lui retirer, aucune erreur n'est signalée mais on peut constater ensuite que SELECT1 a gardé son droit SELECT.

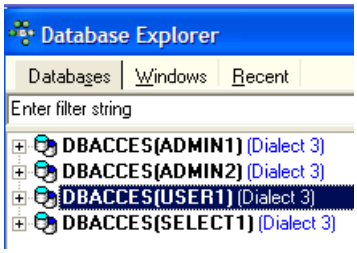
Un droit peut être donné à tous avec la syntaxe : GRANT droit(s) ON table / vue TO **PUBLIC**. Donnons ainsi le droit SELECT sur la table TA à tous. On peut utiliser ADMIN1 ou ADMIN2 pour le faire. Nous utilisons ADMIN2 :



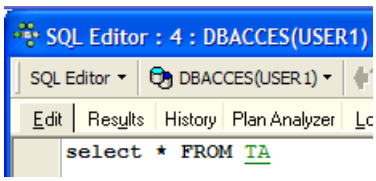


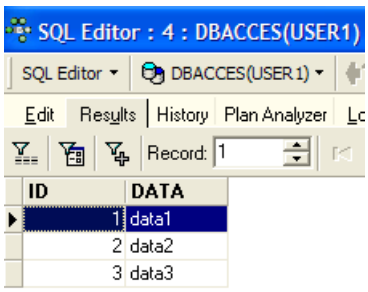
Créons une connexion sur la base avec l'utilisateur USER1 / user1 :





Avec la connexion DBACCES(USER1), ouvrons un nouvel éditeur SQL (Shift + F12) et tapons les commandes suivantes :





L'utilisateur USER1 a bien le droit SELECT sur la table TA.

7.3 Les transactions

7.3.1 Niveaux d'étanchéité

Nous quittons maintenant le problème des droits d'accès aux objets d'une base de données pour aborder celui des accès concurrents à ces objets. Deux utilisateurs ayant les droits d'accès suffisants à un objet de la base, une table par exemple, veulent l'utiliser en même temps. Que se passe-t-il ?

Chaque utilisateur travaille au sein d'une transaction. Une transaction est une suite d'ordres SQL qui est exécutée de façon "atomique" :

- soit toutes les opérations réussissent
- soit l'une d'elles échoue et alors toutes celles qui ont précédé sont annulées

Au final, les opérations d'une transaction ont soit toutes été appliquées avec succès soit aucune n'a été appliquée. Lorsque l'utilisateur a lui-même la maîtrise de la transaction (le cas dans tout ce document), il valide une transaction par un ordre COMMIT ou l'annule par un ordre ROLLBACK.

Chaque utilisateur travaille dans une transaction qui lui appartient. On distingue habituellement quatre niveaux d'étanchéité entre les différents utilisateurs :

- **Uncommitted Read**
- **Committed Read**
- **Repeatable Read**
- **Serializable**

Uncommitted Read

Ce mode d'isolation est également appelé "Dirty Read". Voici un exemple de ce qui se peut se passer dans ce mode :

1. un utilisateur U1 commence une transaction sur une table T
2. un utilisateur U2 commence une transaction sur cette même table T
3. l'utilisateur U1 modifie des lignes de la table T mais ne les valide pas encore
4. l'utilisateur U2 "voit" ces modifications et prend des décisions à partir de ce qu'il voit
5. l'utilisateur annule sa transaction par un ROLLBACK

On voit qu'en 4, l'utilisateur U2 a pris une décision à partir de données qui s'avèreront fausses ultérieurement.

Committed Read

Ce mode d'isolation évite l'écueil précédent. Dans ce mode, l'utilisateur U2 à l'étape 4 ne "verra" pas les modifications apportées par l'utilisateur U1 à la table T. Il ne les verra qu'après que U1 ait fait un COMMIT de sa transaction.

Dans ce mode, également appelé "Unrepeatable Read", on peut rencontrer les situations suivantes :

1. un utilisateur U1 commence une transaction sur une table T
2. un utilisateur U2 commence une transaction sur cette même table T
3. l'utilisateur U2 fait un SELECT pour obtenir la moyenne d'une colonne C des lignes de T vérifiant une certaine condition
4. l'utilisateur U1 modifie (UPDATE) certaines valeurs de la colonne C de T et les valide (COMMIT)
5. l'utilisateur U2 refait le même SELECT qu'en 3. Il découvrira que la moyenne de la colonne C a changé à cause des modifications faites par U1.

Maintenant l'utilisateur U2 ne voit que les modifications "validées" par U1. Mais alors qu'il reste dans la même transaction, deux opérations identiques 3 et 5 donnent des résultats différents. Le terme "Unrepeatable Read" désigne cette situation. C'est une situation ennuyeuse pour quelqu'un qui désire avoir une image stable de la table T.

Repeatable Read

Dans ce mode d'isolation, un utilisateur est assuré d'avoir les mêmes résultats pour ses lectures de la base tant qu'il reste dans la même transaction. Il travaille sur une photo sur laquelle ne sont jamais répercutées les modifications apportées par les autres transactions, mêmes validées. Il ne verra celles-ci que lorsque lui-même terminera sa transaction par un COMMIT ou ROLLBACK.

Ce mode d'isolation n'est cependant pas encore parfait. Après l'opération 3 ci-dessus, les lignes consultées par l'utilisateur U2 sont verrouillées. Lors de l'opération 4, l'utilisateur U1 ne pourra pas modifier (UPDATE) les valeurs de la colonne C de ces lignes. Il

peut cependant rajouter des lignes (INSERT). Si certaines des lignes ajoutées vérifient la condition testée en 3, l'opération 5 donnera une moyenne différente de celle trouvée en 3 à cause des lignes rajoutées.

Pour résoudre ce nouveau problème, il faut passer en isolation "Serializable".

Serializable

Dans ce mode d'isolation, les transactions sont complètement étanches les unes des autres. Il assure que le résultat de deux transactions menées simultanément donneront le même résultat que si elles étaient faites l'une après l'autre. Pour arriver à ce résultat, lors de l'opération 4 où l'utilisateur U1 veut ajouter des lignes qui changeraient le résultat du SELECT de l'utilisateur U1, il en sera empêché. Un message d'erreur lui indiquera que l'insertion n'est pas possible. Elle le deviendra lorsque l'utilisateur U2 aura validé sa transaction.

Les quatres niveaux SQL d'isolation des transactions ne sont pas disponibles dans tous les SGBD. Firebird fournit les niveaux d'isolation suivants :

- **snapshot** : mode d'isolation par défaut. Correspond au mode "Repeatable Read" du standard SQL.
- **committed read** : correspond au mode **"committed read"** du standard SQL

Ce niveau d'isolation est fixé par la commande SET TRANSACTION :

`syntaxe`

SET TRANSACTION

[READ WRITE | READ ONLY]

[WAIT|NOWAIT]

ISOLATION LEVEL [SNAPSHOT | READ COMMITTED]

`fonctionnement`

les mots clés soulignés sont les valeurs par défaut

READ WRITE : la transaction peut lire et écrire

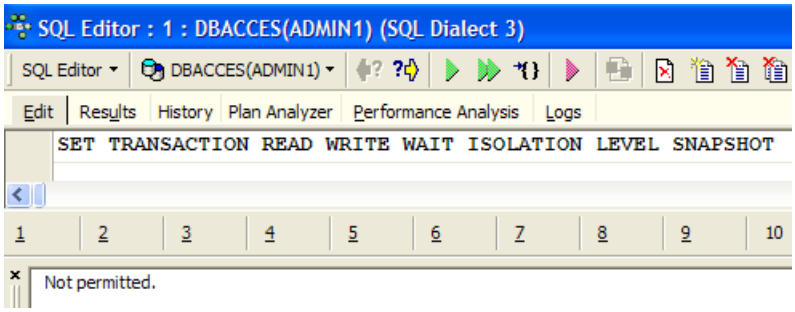
READ ONLY : la transaction ne peut que lire

WAIT : en cas de conflit entre deux transactions, celle qui n'a pu faire son opération attend que l'autre transaction soit validée. Elle ne peut plus émettre d'ordres SQL.

NOWAIT : la transaction qui n'a pu faire son opération n'est pas bloquée. Elle reçoit un message d'erreur et elle peut continuer à travailler.

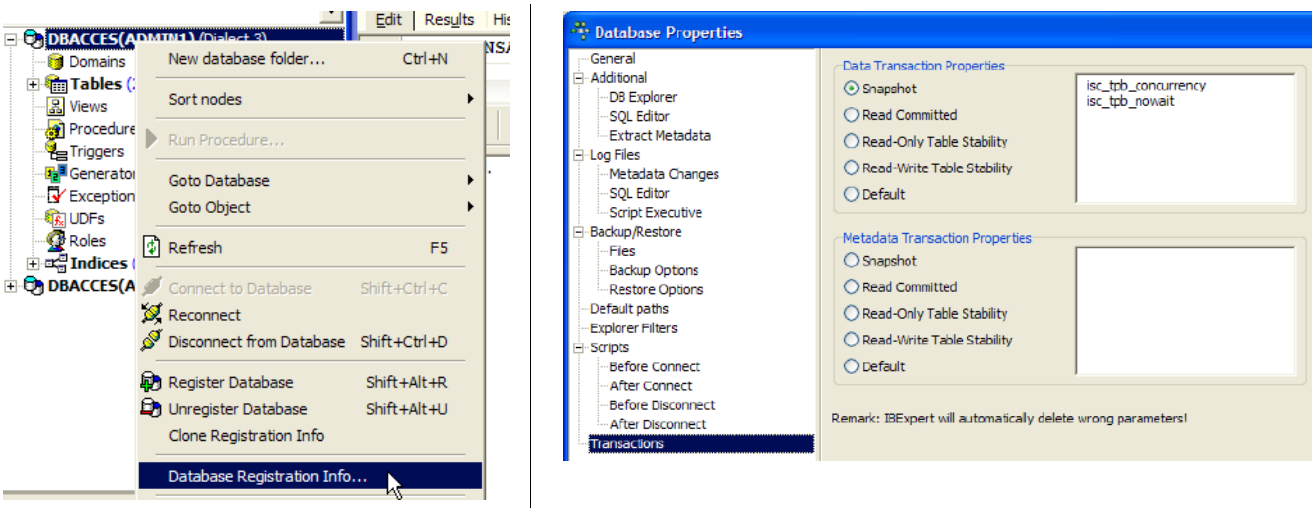
ISOLATION LEVEL [SNAPSHOT | READ COMMITTED] : niveau d'isolation

Essayons. Dans l'éditeur SQL(ADMIN1) on tape la commande SQL suivante :



On voit qu'elle n'a pas été autorisée. On ne sait pourquoi...

IB-Expert permet de fixer d'une autre façon le mode d'isolation. Cliquons droit sur la connexion DBACCES(ADMIN1) pour prendre l'option [Database Registration Info] :



L'écran de droite montre la présence d'une option [Transactions]. Elle va nous permettre de fixer le niveau d'isolation des transactions. Nous le fixons ici à [snapshot]. Nous faisons de même avec la connexion DBACCES(ADMIN2).

7.3.2 Le mode snapshot

Examinons le niveau d'isolation *snapshot* qui est le mode d'isolation par défaut de Firebird. Lorsque l'utilisateur commence une transaction, une photo est prise de la base. L'utilisateur va alors travailler sur cette photo. Chaque utilisateur travaille ainsi sur une photo de la base qui lui est propre. S'il apporte des modifications à celle-ci, les autres utilisateurs ne les voient pas. Ils ne les verront que lorsque l'utilisateur les ayant faites les aura validées par un COMMIT.

On peut considérer deux cas :

- un utilisateur lit la table (select) pendant qu'un autre est en train de la modifier (insert, update, delete)
- les deux utilisateurs veulent modifier la table en même temps

7.3.2.1 Principe de la lecture cohérente

Soient deux utilisateurs *U1* et *U2* travaillant sur la même table *TAB* :

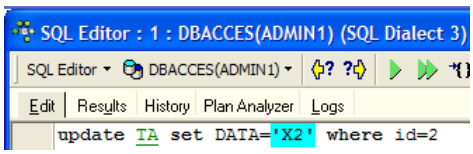
```
--------+----------+--------+-------+----------------------
         T1a        T2a       T1b     T2b
```

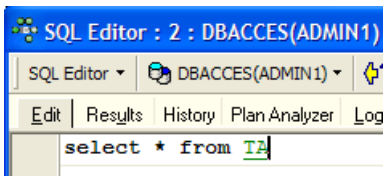
La transaction de l'utilisateur *U1* commence au temps *T1a* et finit au temps *T1b*.
La transaction de l'utilisateur *U2* commence au temps *T2a* et finit au temps *T2b*.

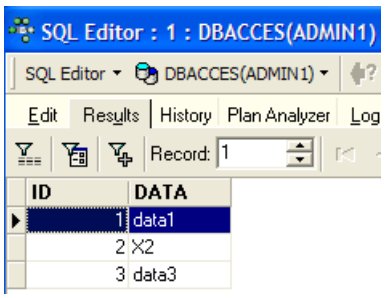
U1 travaille sur une photo de *TAB* prise au temps *T1a*. Entre *T1a* et *T1b*, il modifie *TAB*. Les autres utilisateurs n'auront accès à ces modifications qu'au temps *T1b*, lorsque *U1* fera un *COMMIT*.

U2 travaille sur une photo de *TAB* prise au temps *T2a*, donc la même photo qu'utilisée par *U1* (si d'autres utilisateurs n'ont pas modifié l'original entre-temps). Il ne "voit" pas les modifications qu'a pu apporter l'utilisateur *U1* sur *TAB*. Il ne pourra les voir qu'au temps *T1b*.

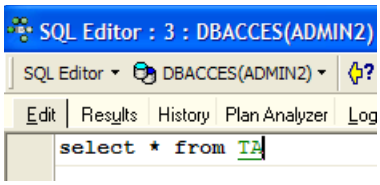
Illustrons ce point sur notre base [DBACCES]. Nous allons faire travailler simultanément les deux utilisateurs [ADMIN1] et [ADMIN2]. Plaçons-nous sur la connexion DBACCES(ADMIN1) et dans l'éditeur SQL d'ADMIN1, faisons les opérations suivantes :

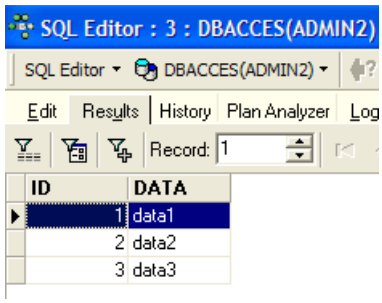




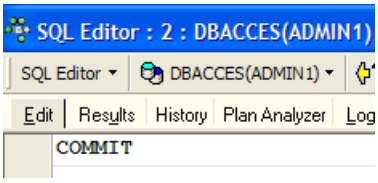


ADMIN1 a modifié la ligne n° 2 de la table TA mais n'a pas encore validé (COMMIT) son opération. L'utilisateur ADMIN2 fait alors un SELECT sur la table TA (on passe dans l'éditeur SQL d'ADMIN2). Nous sommes avant le temps T2a de l'exemple.

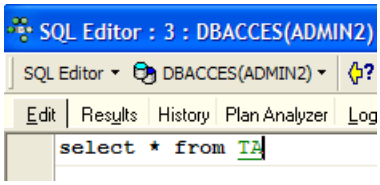


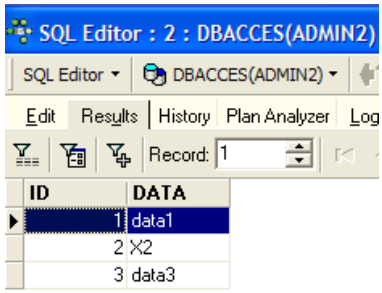


Retour dans l'éditeur SQL d'ADMIN1 qui valide son ajout :



Retour dans l'éditeur SQL d'ADMIN2 pour refaire le SELECT :





ADMIN2 voit les modifications apportées par ADMIN1. Dans le mode snapshot, une transaction ne voit pas les modifications apportées par les autres transactions tant que ces dernières ne sont pas terminées.

7.3.2.2 Modification simultanée par deux transactions d'un même objet de la base

Prenons un exemple en comptabilité : *U1* et *U2* travaillent sur des comptes. *U1* débite *comptex* d'une somme S et crédite *comptey* de la même somme. Il va le faire en plusieurs étapes :

```
--------+----------+--------+-------+----------------------
        T1a        T1b      T1c       T1d
```

U1 démarre une transaction au temps T1a, débite *comptex* au temps T1b, crédite *comptey* au temps T1c et valide les deux opérations au temps T1d. Supposons par ailleurs que *U2* veuille faire la même chose, commence sa transaction au temps *T2a* et la termine au temps T2d selon le schéma suivant :

```
--------+----------+----+----+-------+------+-----+-------+---------
        T1a        T1b  T2a    T1c      T2b    T1d    T2c     T2d
```

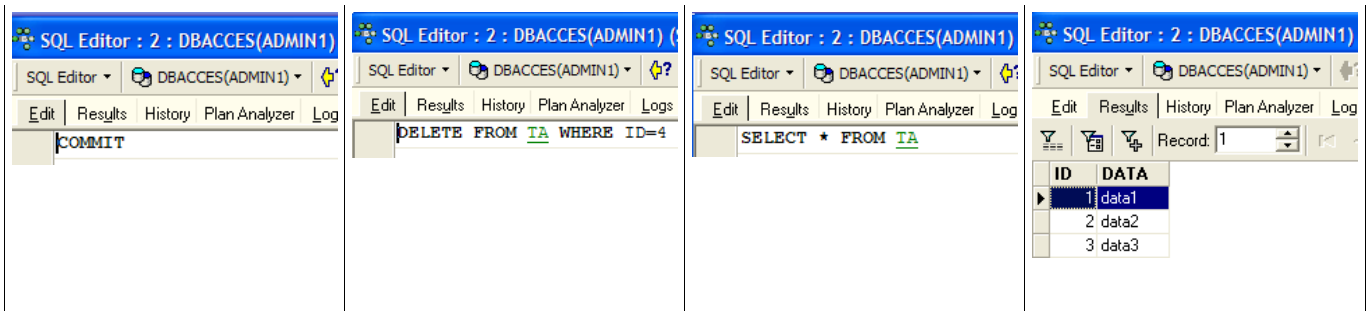
Au temps *T2*, une photo de la table des comptes est prise pour *U2*. Elle est cohérente d'après le principe du *snapshot*. U2 voit l'état initial des comptes *comptex* et *comptey* car U1 n'a pas encore validé ses opérations.

Supposons que *comptex* ait un solde initial de 1000 € et que chacun des utilisateurs U1 et U2 veuillent le débiter de 100 €.

- au temps T1b, U1 décrémente *comptex* de 100 € et le passe donc à 90 €. Cette opération ne sera validée qu'au temps T1d.
- au temps T2b, U2 voit *comptex* avec 1000 € (principe de lecture cohérente) et le décrémente de 100 € et le passe donc à 90 €.
- au final, au temps T2d lorsque tout aura été validé, *comptex* aura un solde de 90 € au lieu des 80 € attendus.

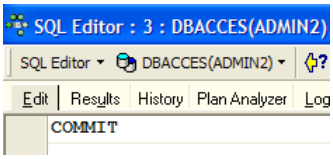
La solution à ce problème est d'empêcher U2 de modifier *comptex* tant que U1 n'a pas terminé sa transaction. U2 sera ainsi bloqué jusqu'au temps T1d. Le mode *snapshot* fournit ce mécanisme.

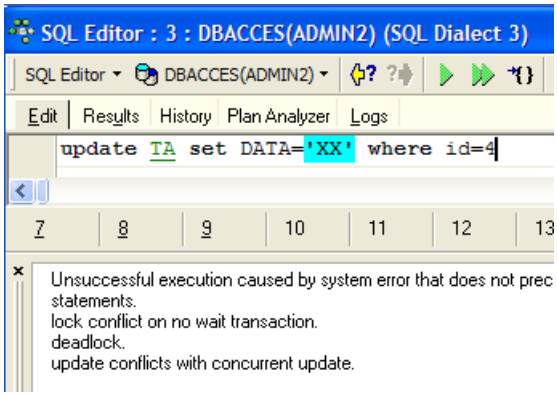
Illustrons-le avec la base DBACCES. ADMIN1 commence une transaction dans son éditeur SQL(ADMIN1) :



Nous avons commencé par faire un COMMIT pour être sûrs de démarrer une nouvelle transaction. Puis nous avons supprimé la ligne n° 4. La transaction n'a pas encore été validée.

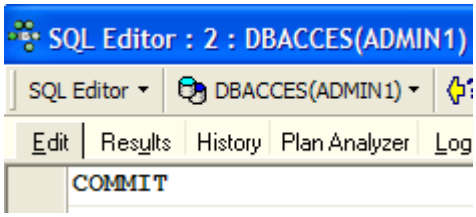
ADMIN2 commence à son tour une transaction dans son éditeur SQL(ADMIN2) :



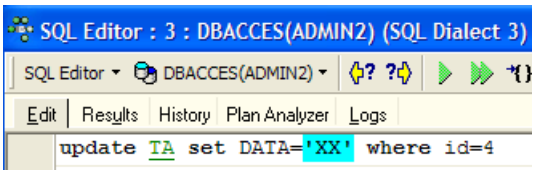


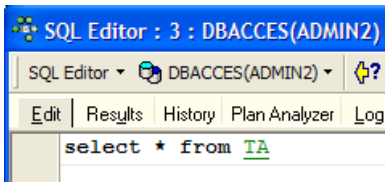
L'écran de droite montre que ADMIN2 a voulu modifier la ligne n° 4. On lui a répondu que ce n'était pas possible parce que quelqu'un d'autre l'avait déjà modifiée mais pas encore validé cette modification.

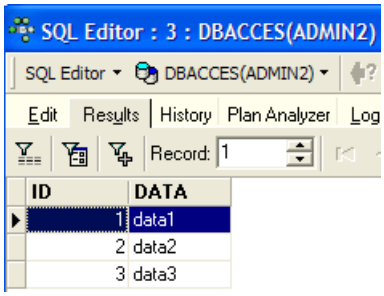
Revenons dans l'éditeur SQL(ADMIN1) pour faire le COMMIT :

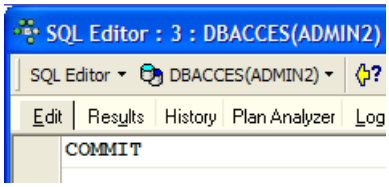


Revenons dans l'éditeur SQL(ADMIN2) pour rejouer la commande UPDATE :







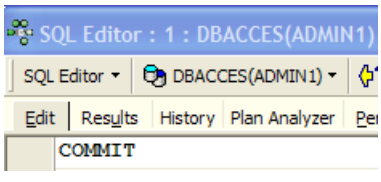


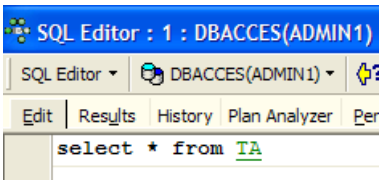
L'opération UPDATE se passe bien alors même que la ligne n° 4 n'existe plus comme le montre le SELECT qui suit. C'est à ce moment qu'ADMIN2 découvre que la ligne n'existe plus.

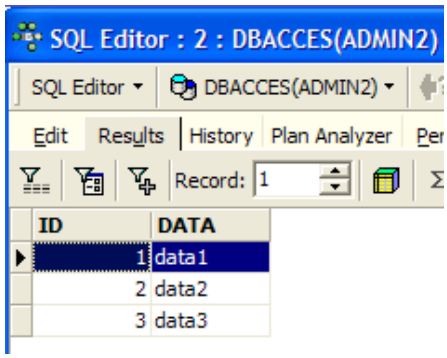
7.3.2.3 Le mode Repeatable Read

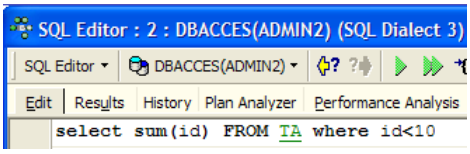
Illustrons maintenant le mode "Repeatable Read". Ce niveau d'isolation est fourni par le mode "snapshot". Il assure à une transaction d'obtenir toujours le même résultat lors de la lecture de la base.

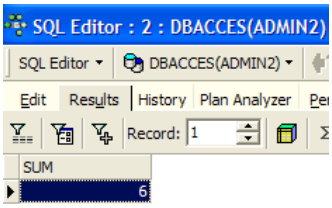
Commençons par travailler avec l'éditeur SQL d'ADMIN2 :



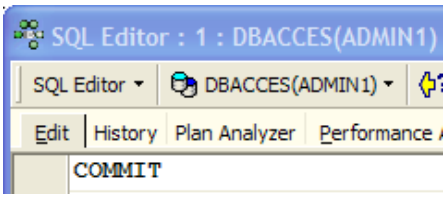


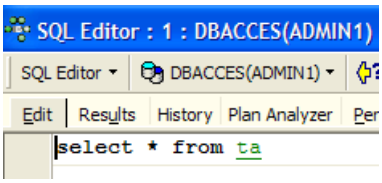


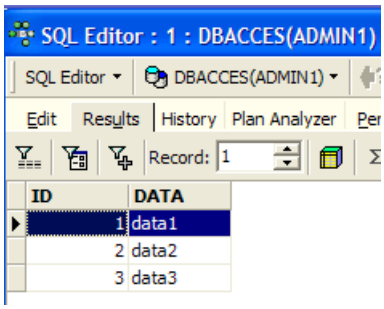


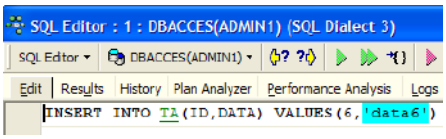


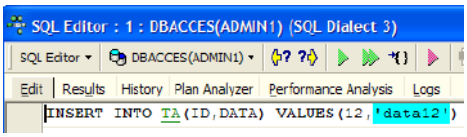
Passons maintenant à l'éditeur SQL d'ADMIN1 :

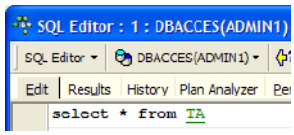


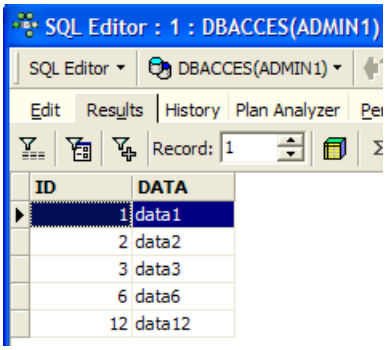


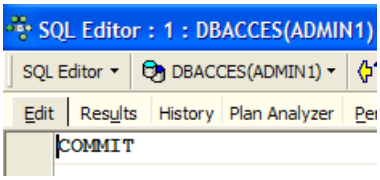




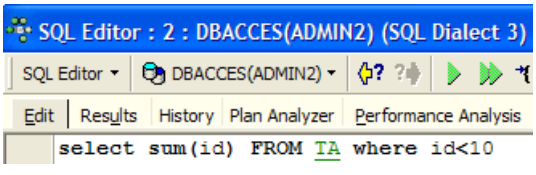


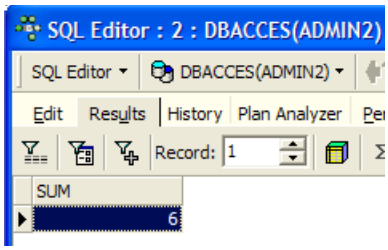






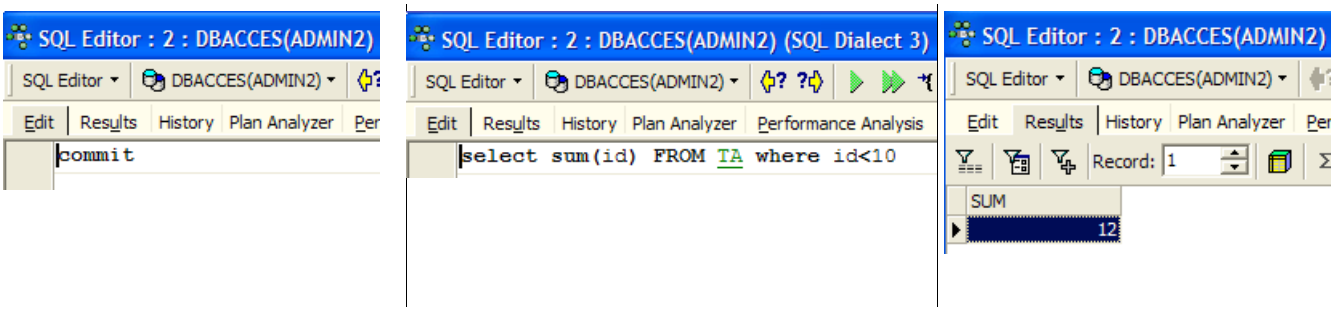
L'utilisateur ADMIN1 a ajouté deux lignes et validé sa transaction. Revenons maintenant sur l'éditeur SQL(ADMIN2) pour rejouer le SELECT SUM :





On voit qu'ADMIN2 ne voit pas les ajouts de lignes d'ADMIN1 bien qu'elles aient été validées par un COMMIT. Le SELECT SUM donne le même résultat qu'avant les ajouts. C'est le principe du Repeatable Read.

Maintenant, toujours dans l'éditeur SQL (ADMIN2), validons la transaction par un COMMIT puis rejouons le SELECT SUM :



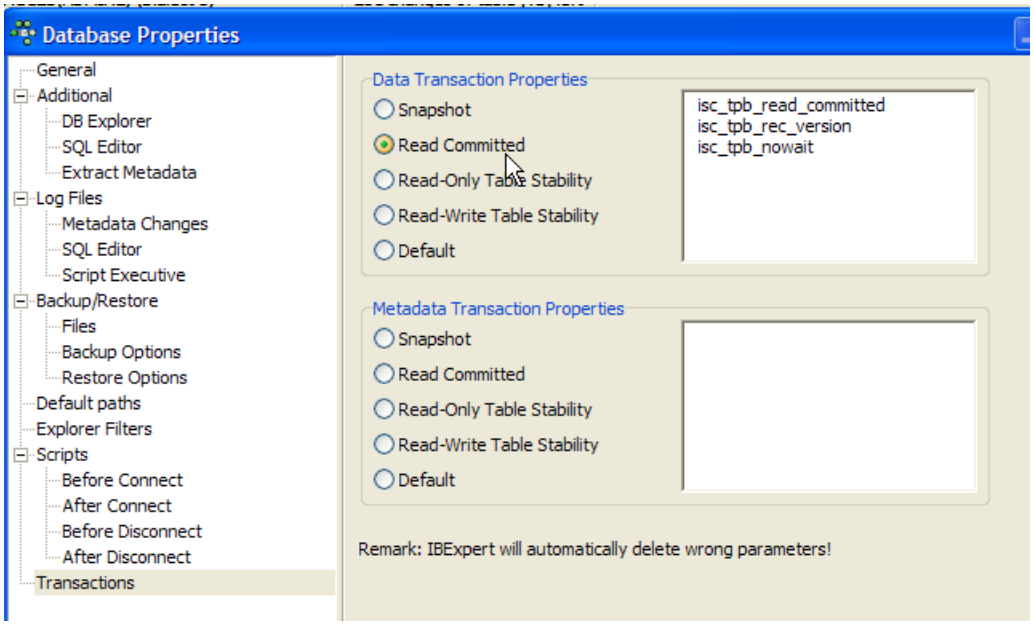
Les lignes ajoutées par ADMIN1 sont désormais prises en compte.

7.3.3 Le mode Committed Read

Illustrons maintenant le mode "Committed Read". Ce niveau d'isolation est analogue à celui du *snapshot* sauf en ce qui concerne le "Repeatable Read".

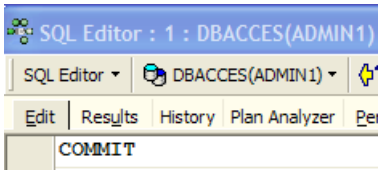
Nous commençons par changer le niveau d'isolation des transactions des deux connexions.

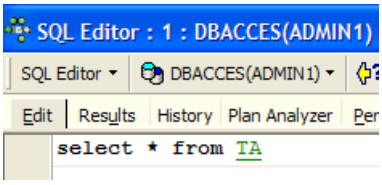
- nous déconnectons les deux utilisateurs ADMIN1 et ADMIN2
- nous changeons le niveau d'isolation de leurs transactions

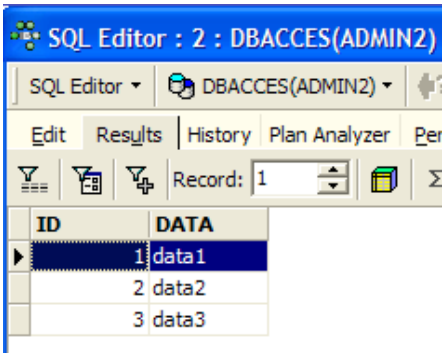


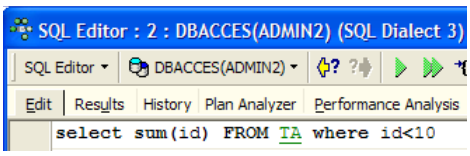
- nous reconnectons les utilisateurs ADMIN1 et ADMIN2

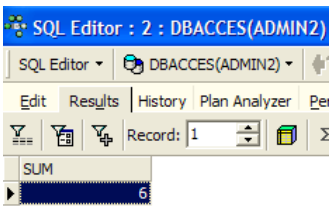
Nous reprenons maintenant l'exemple précédent qui illustrait le "Repeatable Read" afin de montrer que nous n'avons plus le même comportement. Commençons par travailler avec l'éditeur SQL d'ADMIN2 :



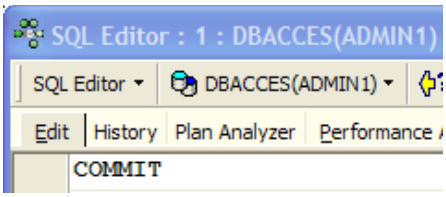


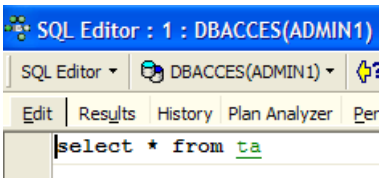


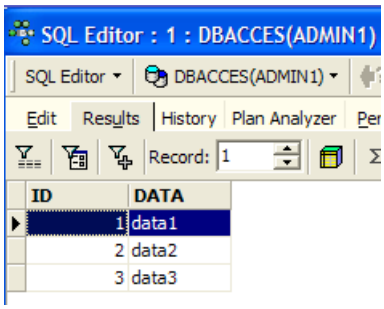


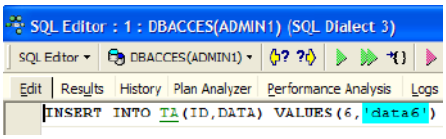


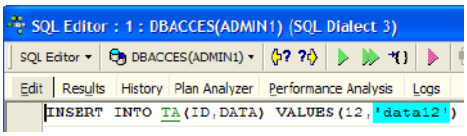
Passons maintenant à l'éditeur SQL d'ADMIN1 :

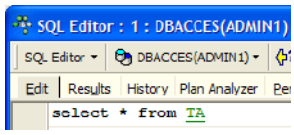


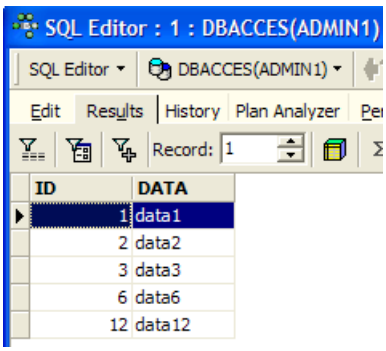


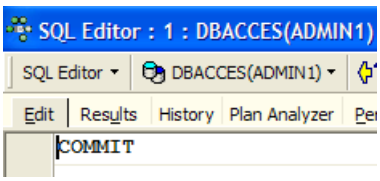




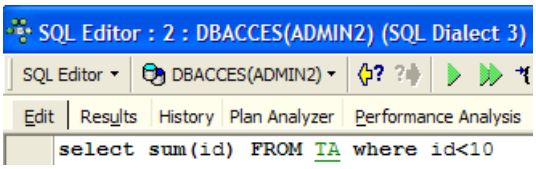


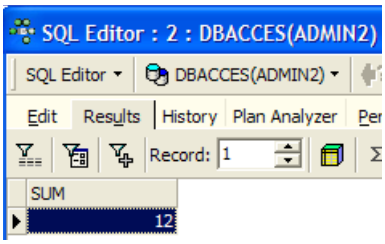






L'utilisateur ADMIN1 a ajouté deux lignes et validé sa transaction. Revenons maintenant sur l'éditeur SQL(ADMIN2) pour rejouer le SELECT SUM :



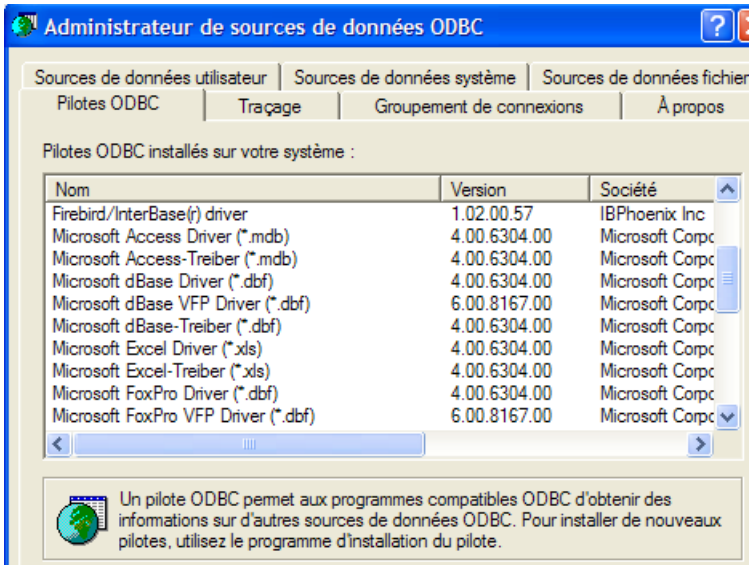


Le SELECT SUM ne donne pas le même résultat qu'avant les ajouts faits par ADMIN1. C'est la différence entre les modes **snapshot** et **read committed**.

8 Installer et utiliser un pilote ODBC pour [Firebird]

8.1 Installer le pilote

Il existe de nombreuses bases de données sur le marché. Afin d'uniformiser les accès aux bases de données sous MS Windows, Microsoft a développé une interface appelée ODBC (Open DataBase Connectivity). Cette couche cache les particularités de chaque base de données sous une interface standard. Il existe sous MS Windows de nombreux pilotes ODBC facilitant l'accès aux bases de données. Voici par exemple, quelques-uns des pilotes ODBC installés sur une machine Windows XP :

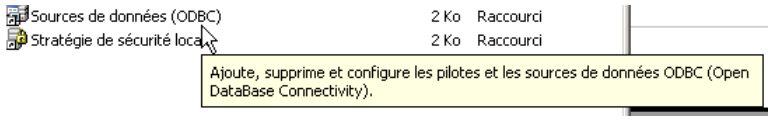


Une application s'appuyant sur ces pilotes peut utiliser n'importe quelle bases de données sans ré-écriture. Le pilote ODBC s'intercale entre l'application et le SGBD. Le dialogue Application <-> pilote ODBC est standard. Si on change de SGBD, on installe alors le pilote ODBC du nouveau SGBD et l'application reste inchangée.

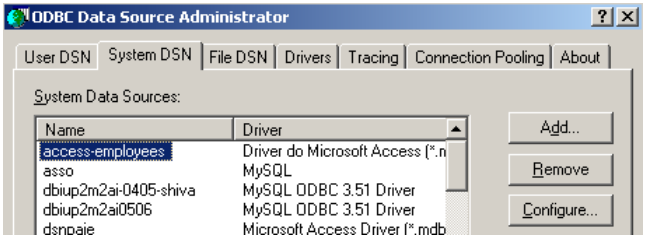
Le lien [firebird-odbc-provider] de la page de téléchargements de [Firebird] (paragraphe 2.1, page 5) donne accès à un pilote ODBC. Une fois celui-ci installé, il apparaît dans la liste des pilotes ODBC installés.

8.2 Créer une source ODBC

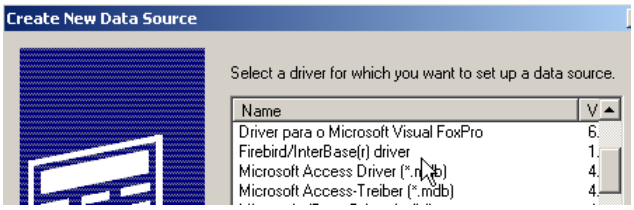
- lancer l'outil [Démarrer -> Paramètres -> Outil de configuration -> Outils d'administration -> Sources de données ODBC] :



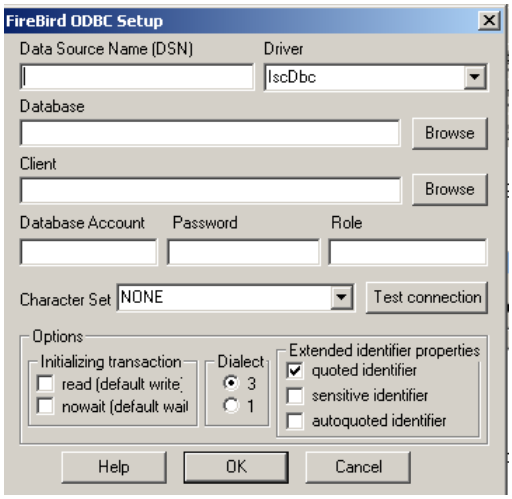
- on obtient la fenêtre suivante :



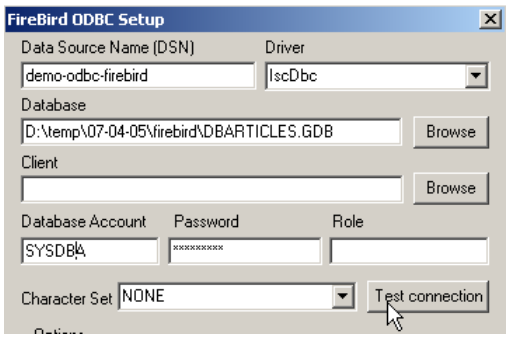
- ajoutons [Add] une nouvelle source de données système (panneau [System DSN]) qu'on associera à la base Firebird [dbarticles] que nous avons créée au paragraphe 2.3, page 9 :



- il nous faut tout d'abord préciser le pilote ODBC à utiliser. Ci-dessus, nous choisissons le pilote pour Firebird puis nous faisons [Terminer]. L'assistant du pilote ODBC de Firebird prend alors la main :

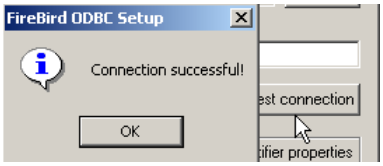


- nous remplissons les différents champs :



`DSN [demo-odbc-firebird]`	le nom DSN de la source ODBC - peut être quelconque
`Database [D:\..\DBARTICLES.GDB]`	le nom de la BD Firebird à exploiter - utiliser [Browse] pour désigner le fichier .gbd correspondant. Nous utilisons ici la base d'articles [dbarticles] créée page 9.
`Database Account [SYSDBA]`	identifiant à utiliser pour se connecter à la base
`Password [masterkey]`	le mot de passe associé à cet identifiant

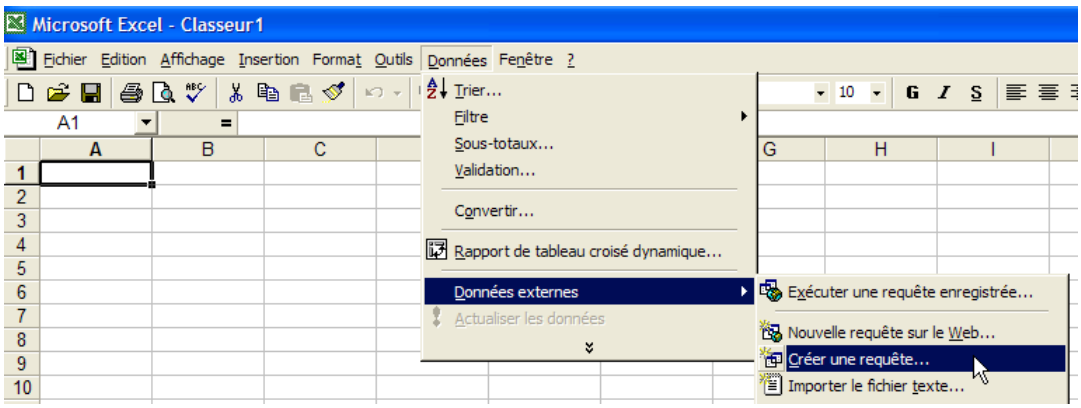
Le bouton [Test connection] permet de vérifier la validité des informations que nous avons données. Avant de l'utiliser, lancer le SGBD [Firebird] :



- valider l'assistant ODBC, en faisant [OK] autant de fois que nécessaire

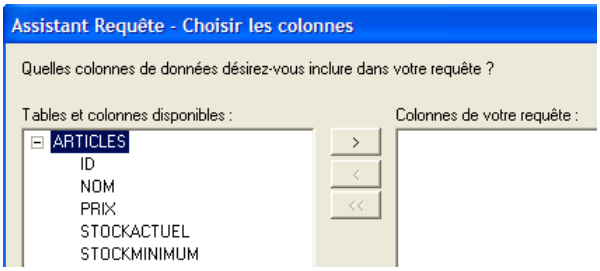
8.3 Tester la source ODBC

Il y a diverses façons de vérifier le bon fonctionnement d'une source ODBC. Nous allons ici utiliser Excel :

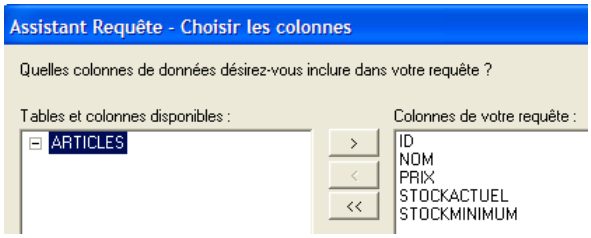


- utilisons l'option [Données -> Données externes -> Créer une requête] ci-dessus. Nous obtenons la première fenêtre d'un assistant de définition de la source de données. Le panneau [Bases de données] liste les sources ODBC actuellement définies sur la machine :

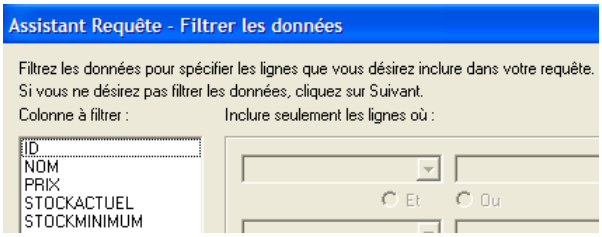
- choisissons la source ODBC [odbc-firebird-articles] que nous venons de créer et passons à l'étape suivante avec [OK] :



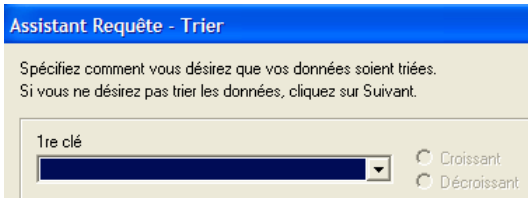
- cette fenêtre liste les tables et colonnes disponibles dans la source ODBC. Nous prenons toute la table :



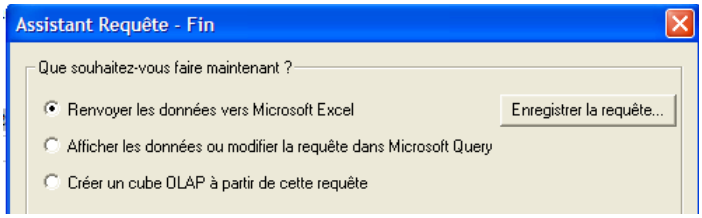
- passons à l'étape suivante avec [Suivant] :



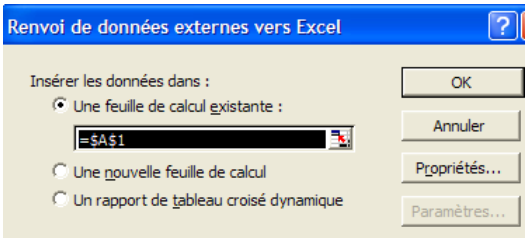
- cette étape nous permet de filtrer les données. Ici nous ne filtrons rien et passons à l'étape suivante :



- cette étape nous permet de trier les données. Nous ne le faisons pas et passons à l'étape suivante :



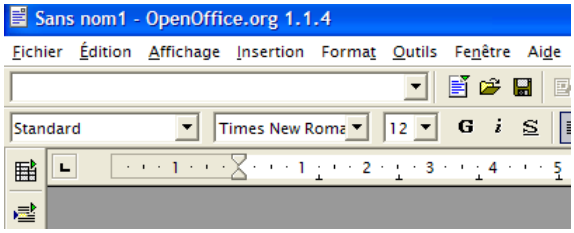
- la dernière étape nous demande ce qu'on veut faire des données. Ici, nous les renvoyons vers Excel :

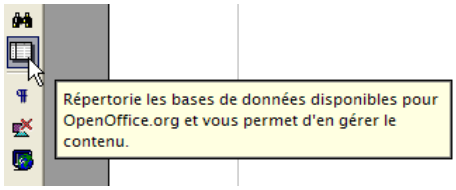


- ici, Excel demande où on veut mettre les données récupérées. On les met dans la feuille active à partir de la cellule A1. Les données sont alors récupérées dans la feuille Excel :

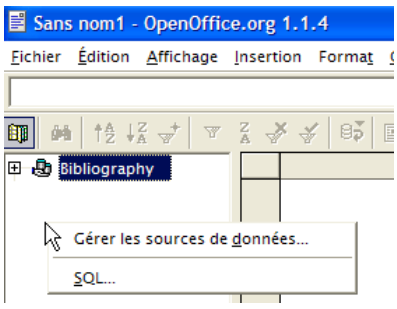
	A	B	C	D	E
1	ID	NOM	PRIX	STOCKACTUEL	STOCKMINIMUM
2	1	parapluie	50	1	1
3	2	bottes	20	2	3
4	3	chapeau	100	5	2

Il y a d'autres façons de tester la validité d'une source ODBC. On pourra par exemple utilliser la suite gratuite OpenOffice disponible à l'url [http://www.openoffice.org]. Voici un exemple avec un texte OpenOffice :

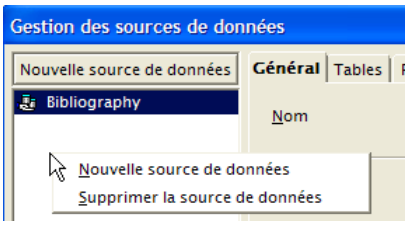




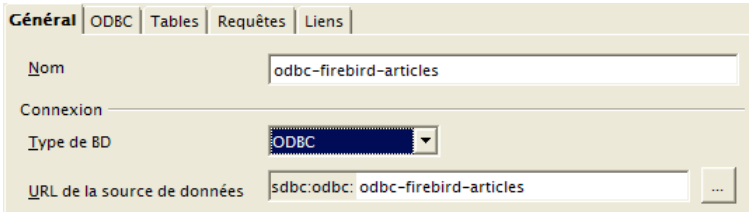
- une icône sur le côté gauche de la fenêtre d'OpenOffice donne accès aux sources de données. L'interface change alors pour introduire une zone de gestion des sources de données :



- une source de données est prédéfinie, la source [Bibliography]. Un clic droit sur la zone des sources de données nous permet d'en créer une nouvelle avec l'option [Gérer les sources de données] :



- un assistant [Gestion des sources de données] permet de créer des sources de données. Un clic droit sur la zone des sources de données nous permet d'en créer une nouvelle avec l'option [Nouvelle source de données] :



`Nom [odbc-firebird-articles]` un nom quelconque. Ici on a repris le nom de la source ODBC

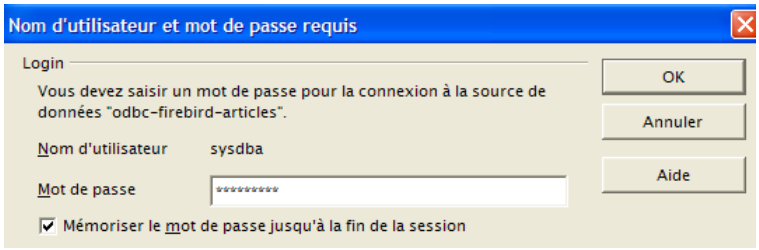
`Type de BD [odbc]` OpenOffice sait gérer différents types de BD via JDBC, ODBC ou directement (MySQL, Dbase, ...). Pour notre exemple, il faut choisir ODBC

`URL de la source de données` le bouton à droite du champ de saisie nous donne accès à la liste des sources ODBC de la machine. Nous choisissons la source [odbc-firebird-articles]

- nous passons au panneau [ODBC] pour y définir l'utilisateur sous l'identité duquel se fera la connexion :

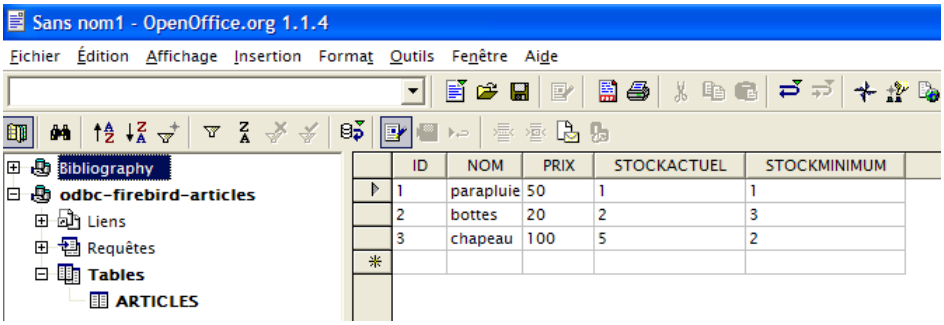
`Nom d'utilisateur [sysdba]` le propriétaire de la source ODBC

- on passe au panneau [Tables]. Le mot de passe est demandé. Ici c'est [masterkey] :



- on fait [OK]. La liste des tables de la source ODBC est alors présentée :

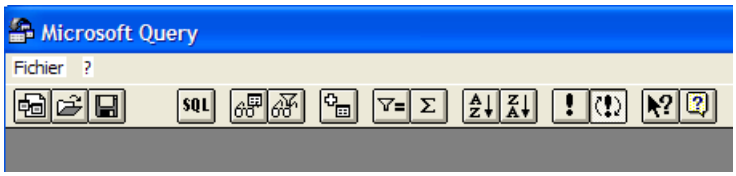
- on peut définir les tables qui seront présentées au document [OpenOffice]. Ici nous choisissons la table [ARTICLES] et nous faisons [OK]. La définition de la source de données est terminée. Elle apparaît alors dans la liste des sources de données du document actif :



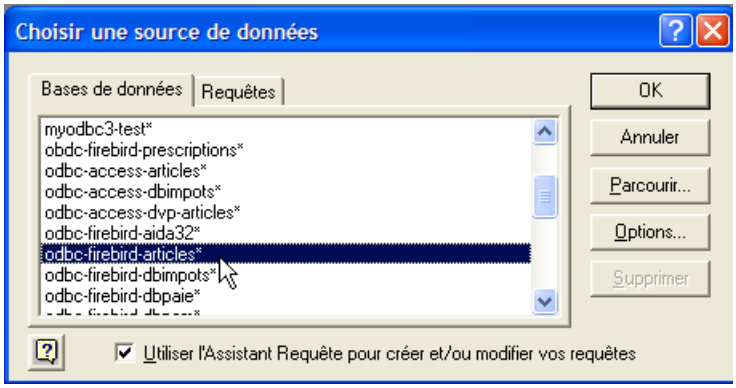
- on peut avec la souris faire glisser la table [ARTICLES] ci-dessus dans le texte [OpenOffice].

8.4 Microsoft Query

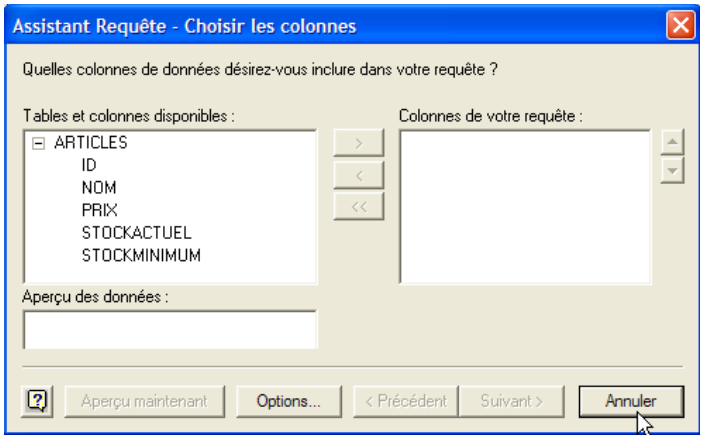
Quoique MS Query soit livré avec MS Office, il n'existe pas toujours de lien vers ce programme. On le trouve dans le dossier *Office* de MS Office sous le nom *MSQRY32.EXE*. Par exemple *"C:\Program Files\Office 2000\Office\MSQRY32.EXE"*. MS Query permet d'interroger toute source de données ODBC avec des requêtes SQL. Celles-ci peuvent être construites graphiquement ou tapées directement au clavier. Comme la plupart des bases de données pour Windows fournissent des pilotes ODBC, elles peuvent donc toutes être interrogées avec MS Query. Lorsque MS Query est lancé, on a l'affichage suivant :



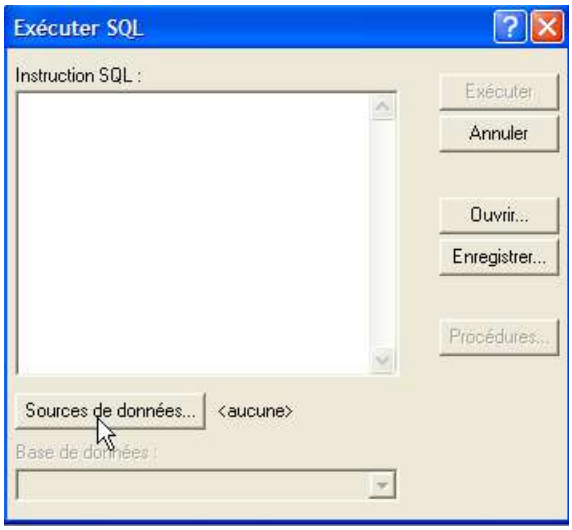
Il nous faut tout d'abord désigner la source de données ODBC qui va être interrogée On utilise pour cela l'option : *Fichier/Nouvelle* :



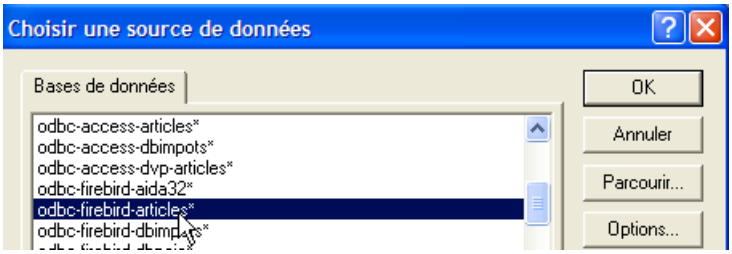
Nous utiliserons la source ODBC créée précédemment. MS Query nous présente alors la structure de la source :



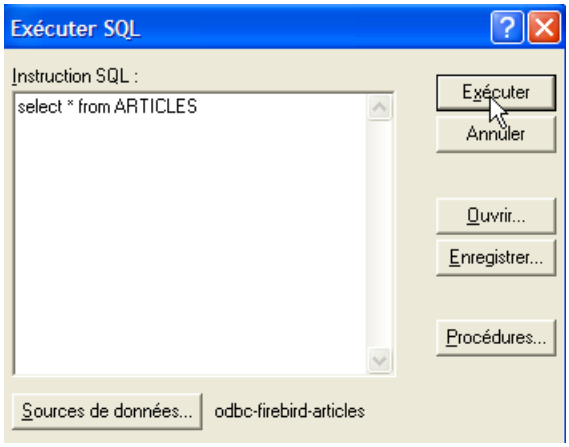
Nous utilisons le bouton [Annuler] car l'assistant proposé n'est guère utile si on connaît le langage SQL. Nous pouvons émettre des requêtes SQL sur la source ODBC sélectionnée avec l'option [Fichier / Exécuter SQL] :



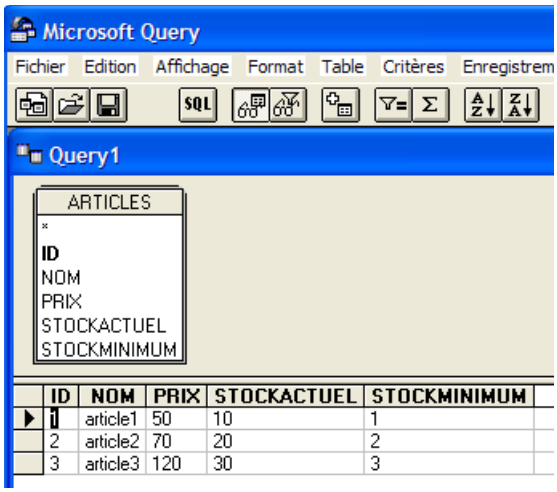
Nous sommes encore amenés à sélectionner la source ODBC :



Une fois la source ODBC choisie, on peut émettre des ordres SQL dessus :



Nous obtenons le résultat suivant :



Le lecteur est invité à créer une source ODBC avec la base de données Firebird DBBIBLIO et à rejouer les exemples précédents.