Initiation à l'utilisation de SQL avec OOo Basic dans Base

Version 1.00 du 28.11.2006

Réalisé avec : **OOo 2.0.3**

Plate-forme / Os : **Toutes**

Distribué par le projet fr.OpenOffice.org

Sommaire

1 - Présentation ; objectifs

La suite bureautique libre OpenOffice.org comporte un langage de programmation appelé OpenOffice Basic (Ooo Basic), qui permet d' écrire des **macros** pour automatiser les tâches dans les divers modules de cette suite. Un de ces modules, appelé Base (gestion des bases de données) sert d'interface utilisateur (de client) à une base de données externe, ou à la base intégrée HSQLDB. Une façon de communiquer avec cette base de données est d'utiliser le langage SQL à travers OOo Basic. Il faut cependant pour cela connaître les deux langages en question ! On peut d'ailleurs utiliser d'autres langages (Python, Java, ...).

On pourra apprendre OOo Basic dans l'aide intégrée, avec divers tutoriels disponibles en ligne (par exemple voir http://wiki.services.openoffice.org/wiki/Extensions_development_basic_fr) ou encore en consultant le livre « Programmation OpenOffice.org 2 » [*Marcelly et Godard, 2005*] (voir Bibliographie). Le présent document essaie de répondre aux questions de base que se pose le débutant lorsqu'il essaie d'intégrer le langage SQL dans ses macros. Ce n'est qu'une initiation, qu'on utilisera avant de passer à des ouvrages plus complets. En particulier, on consultera le chapitre 9 du HSQLDB User Guide [*HSQLDB Development Group*] consacré à la syntaxe. On pourra aussi trouver divers cours sur le langage SQL sur Internet. Voir la Bibliographie.

Ce document concerne l'utilisation de la base intégrée HSQLDB, même si beaucoup d'informations peuvent être valables avec d'autres systèmes de base de données. L'essentiel devrait être adaptable à d'autres langages que OOo Basic (il faudra en particulier adapter la manière de spécifier les chaînes de caractères).

Si on veut exécuter des commandes SQL directement, on consultera le document « FAQ Base intégrée et requêtes SQL directes» écrit par Manuel Naudin [*Naudin, 2006*].

2 - Qu'est-ce que SQL ?

SQL (Structured Query Language – Langage Structuré d'interrogation) est un langage permettant de communiquer avec une base de données.

Il permet :

- la maintenance des tables (création, modification de la structure, suppression) ; il s'agit de la subdivision DDL (Data Definition Language) du SQL. Nous ne verons qu"une petite partie de cette subdivision.
- la manipulation des données contenues dans les tables : ajouter des enregistrements, les modifier, les supprimer ; c'est la subdivision DML (Data Modification Language) du SQL. Ce tutoriel se concentrera essentiellement à ce sujet après avoir traité la syntaxe.
- La gestion des droits d'accès (qui ne sera pas traitée dans ce document) : subdivision DCL (Data Control Language).
- d'éviter des problèmes d'incohérence dans les données : TCL (subdivision Transaction Control Language) ; nous aborderons succinctement ces instructions.
- Il existe enfin un « embedded SQL » que nous n'aborderons pas.

3 - Comment est construite une instruction SQL ?

Une instruction SQL est une chaîne de caractères, comprenant des **mots-clef** (qui indiquent au moteur de la base ce qu'il doit faire) et des **paramètres** sur lesquels il va travailler. Ces paramètres peuvent être des **identificateurs** (de tables, d'enregistrements, de requêtes, ...) ou des **expressions**.

En voici un exemple (il s'agit d'une requête demandant les champs `PRENOM` et `NOM` de la table `ADHERENTS` pour tous les enregistrements où le champ`SOMMEDUE` est supérieur à 100):

```
SELECT PRENOM, NOM FROM ADHERENTS WHERE SOMMEDUE > 100.00
```

`SELECT`, `FROM` et `WHERE` sont des mots-clef, `PRENOM`, `NOM`, `ADHERENTS` et `SOMMEDUE` sont des identificateurs ; `SOMMEDUE > 100.00` est une expression.

a) Les **identificateurs** sont constitués d'une lettre suivie par un nombre quelconque de lettres et de chiffres. Les lettres sont uniquement des caractères ASCII (c'est-à-dire sans accents ou autre signe diacritique : a-z et A-Z). Ceci peut être contourné en mettant l'identificateur entre guillemets doubles (voir ci-dessous)

Lorsqu'une instruction SQL est transmise à HSQLDB, toutes les lettres sont transformées en majuscules ; ensuite, l'élément désigné par l'identificateur est recherché, en tenant compte de la casse. C'est pourquoi dans ce cas, seuls des éléments avec des noms comportant uniquement des majuscules et des chiffres seront reconnus. Ainsi, si vous avez une table `CLIENTS` comportant les champs `NOM` et `CODEPOSTAL`, les instruction suivantes seront toutes valables et fonctionneront de manière identique (le sens de l'instruction sera explicité plus loin ; SELECT et FROM sont des mots-clef):

```
SELECT NOM, CODEPOSTAL FROM CLIENTS(1)
select nom, codepostal from clients(2)
Select Nom, CodePostal from Clients (3)
```

En effet, les instructions (2) et (3) seront vues comme l'instruction (1) dans laquelle les identificateurs correspondent exactement aux noms de la table et des champs.

Par contre, si pour des raisons diverses vous avez défini une table `Clients` avec les champs `Nom` et `CodePostal`, aucune des trois instructions c-dessus ne fonctionnera, puisqu'il n'y aura jamais correspondance.

On peut contourner la difficulté en mettant les identificateurs entre guillemets. Dans ce cas, la conversion en majuscules n'est pas effectuée, et la recherche pourra réussir si les caractères sont correctement choisis. Ainsi, dans notre cas, il faudra (strictement) écrire

```
SELECT "Nom", "CodePostal" FROM "Clients" (4)
```

On peut aussi utiliser des espaces entre les guillemets, ainsi que des caractères non-ASCII. Mais ceci n'est pas recommandé, car si plus tard on veut porter notre base de données vers un autre système de gestion il n'est pas certain que celui-ci l'accepte.

Pour les puristes, le site Developpez.com présente une normalisation pour (entre autres) le choix des identificateurs (de table, de champ, ...), la taille des champs, ... : voir à l'adresse http://sql.developpez.com/standards/

b) les **mots-clef** ne se mettent jamais entre guillemets, ils peuvent donc indifféremment être en minuscules ou en majuscules..

c) Les expressions

Une expression peut avoir divers types : des nombres (la virgule sera remplacée par un point : `3.1415926`), des chaînes de caractères (encadrées par des apostrophes : `'Ceci est une chaine'`), des comparaisons (avec les signes `=, <, <=, >, >=, !=`) des calculs (avec les signes `+, -, /, *`), etc.

Les dates et les heures seront aussi mises entre apostrophes, aux formats `'AAAA-MM-JJ'` et `'HH:MM:SS'`. Il existe aussi le type `TIMESTAMP` ou `DATETIME`, représentant une date et une heure ensemble, représentées par le format `'AAAA-MM-JJ hh:mm:ss.sssssssss'`

Pour les booléens, on utilisera `TRUE` ou `FALSE` (sans apostrophes). Il est déconseillé d'utiliser `1` ou `0` (plus tard, vous souviendrez-vous ce qu'ils signifient?)

Remarque : ceci est assez simplifié, mais suffisant la plupart du temps.

4 - Conventions d'écriture pour les règles de syntaxe

Dans ce document, pour les définitions formelles, les conventions suivantes seront utilisées

- `[A]` signifie que A est optionnel
- `{A | B }` signifie qu'on doit utiliser soit A soit B
- `[ {A | B } ]` signifie qu'on peut utiliser A ou B ou rien
- `[ A, ... ]` signifie qu'on peut utiliser un nombre quelconque d'éléments de type A, ou aucun
- Lorsque des parenthèses apparaissent dans la syntaxe, elles doivent être utilisées dans l'instruction réelle.
- Les mots en `MAJUSCULES` sont les mots-clefs (mais voir § 3).
- Les mots en `minuscules` représentent tout ce qui n'est pas un mot-clef.
- `<abc>` représente un identificateur ; les caractères `<` et `>` ne sont pas à mettre dans l'instruction réelle.

5 - Comment écrire une instruction SQL en Basic ?

Pour créer une instruction SQL en Basic, on crée la chaîne de caractères correspondante. Nous nous trouvons donc en présence d'une sorte de fusée à deux étages, le premier étage étant OOo Basic, le second étant SQL. Ces deux fusées ne fonctionnent pas avec la même technique (la même syntaxe).

Une instruction SQL est une chaîne de caractères. Donc, il suffira de déclarer une variable chaîne, et de lui affecter la chaîne de caractères correspondante. Cependant, les **guillemets** contenus dans la chaîne devront être **doublés** pour être acceptés comme tels par OOo Basic ; ainsi, pour envoyer l'instruction SQL suivante:

```
SELECT "Nom", "CodePostal" FROM "Clients"
```

on écrira en Basic :

```
Dim sInstrSQL As String
sInstrSQL = "SELECT ""Nom"", ""CodePostal"" FROM ""Clients"""
```

6 - Quelques instructions SQL

La présentation faite ci-dessous de quelques instructions SQL est très simplifiée ; le Hsqldb User Guide donne la syntaxe complète de toutes les instructions disponibles, de toutes les fonctions intégrées, etc. Ce qui est présenté ici devrait cependant permettre une bonne approche.

6.1 - L'instruction INSERT

Cette instruction (qui fait partie du DML – Data Manipulation Language, Langage de Manipulation des Données) permet d'ajouter un enregistrement dans la table spécifiée. Elle se présente sous la forme :

```
INSERT INTO <table> (<NomDeChamp1> [, <NomDeChamp2>, ...] ) VALUES
  ( <Valeur1> [, [ <Valeur2>, ... ])
```

Les premières parenthèses contiennent la liste des champs affectés. Les secondes parenthèses contiennent les valeurs qui seront affectées à ces champ, dans le même ordre. On n'est pas obligé de remplir tous les champs de la table ; seuls les champs définis comme `NOT NULL` doivent être remplis..

Exemple :

```
INSERT INTO "Clients" ("ID", "Nom", "Prenom", "Avoir" ) VALUES ( 12,
  'Dupont', 'Pierre', 152.50)
```

En OOo Basic, supposons que les valeurs à mettre dans les trois champs ci-dessus soient reçus dans des variables, on écrira le code suivant:

```
Dim iID As Integer, sNom As String, sPrenom As String
Dim dAvoir As Integer
Dim sInstrSQL As String, sSavoir As String
REM Ici on déterminera les valeurs à donner aux variables
iID = 45
sNom = "Dupont"
sPrenom = "Pierre"
dAvoir = 152
REM Dans la pratique ces valeurs pourraient par exemple être
REM récupérées dans une boîte de dialogue
sInstrSQL = "INSERT INTO ""Clients"" (""ID"", ""Nom"", ""Prenom"""
sInstrSQL = sInstrSQL & ", ""Avoir"" ) VALUES ( "
sInstrSQL = sInstrSQL & iID & ", '" & sNom & "','"
sInstrSQL = sInstrSQL & sPrenom & "'," & dAvoir & " )"
REM Notez les apostrophes qui encadrent les chaines de
caractères
' MsgBox ( sInstrSQL ) ' pour la mise au point
```

```
REM Il reste à envoyer cette instruction à HSQLDB
REM selon la méthode exposée dans le livre de B. Marcelly et L. Godard.
```

On remarquera les **doubles guillemets** (pour que OOo Basic ne croie pas qu'il s'agit de la fin de la chaîne), et les apostrophes qui encadrent les valeurs des variables chaînes.

L'instruction MsgBox nous donnera la chaîne de caractères qui sera transmise ensuite à SQL

```
INSERT INTO "Clients" ("ID", "Nom", "Prenom", "Avoir" ) VALUES ( 45, 'Dupont ', 'Pierre', 152.5)
```

Supposez maintenant que `dAvoir` ait été défini comme `Double` (ce qui est plus réaliste, car il peut y avoir des centimes):

```
dAvoir = 152.5
```

Le programme ci-dessus ne marchera pas ; en effet, la transformation (implicite) en chaîne de de caractères mettra une virgule. Or, pour transmettre un nombre décimal, il faut veiller à **remplacer la virgule par un point**; pour cela, on peut utiliser la fonction`VirgToPoint` :

```
sAvoir =  VirgToPoint  ( dAvoir )

Function VirgToPoint  ( sValeur As String ) As String
REM Conversion implicite de format Réel en Chaîne
    Dim iPos As Integer
    iPos = InStr (  sValeur, ","  )
    If iPos > 0 then
         Mid ( sValeur,  iPos, 1, "." )
    End if
    VirgToPoint  =  sValeur
End Function
```

et la dernière instruction deviendra:

```
sInstrSQL = sInstrSQL & sPrenom & "',"  & sAvoir & " )"
```

Autre petit problème : supposez qu'un jour nous ayons à introduire un client dont le nom serait «Dumont d'Urville», avec une apostrophe dans le nom. Comme l'apostrophe est reconnue par HSQLDB comme délimiteur de chaîne de caractères, notre bout de programme ci-dessus ne marchera pas et nous aurons un message d'erreur nous indiquant que Urville n'est pas reconnu. Comment faire ? Tout simple : avant l'utilisation d'une chaîne de caractères, **doubler les apostrophes internes à la chaîne** Pour cela on peut procéder ainsi:

```
sNom = DbleApostr(sNom)
```

avec :

```
Function DbleApostr ( sEntree As String ) As String
    Dim lLongueur As Long, sChar As String
    Dim sSortie As String, i As Long
    lLongueur = Len ( sEntree )
    sSortie = ""
    For i = 1 to lLongueur
        sChar = Mid ( sEntree, i, 1 )
        If sChar = "'" then
            sChar = "''"
        End If
        sSortie = sSortie & sChar
    Next i
    DbleApostr = sSortie
End Function
```

Les dates seront mises au bon format, différent du format normal de OooBasic. Là encore, une fonction de conversion sera utile:

```
Function ConvDate (sEntree As String) As String
    ConvDate = Right(sDate, 4) & "-" & Mid(sDate, 4, 2) & "-" & _
            Left(sDate, 2) & "'"
End Function
```

Ne pas oublier d'encadrer le résultat avec des apostrophes!

Pour les heures et les «timestamps», on pourra écrire des fonctions similaires.

Concernant les les booléens, on enverra les valeurs `TRUE` (pour vrai) et `FALSE` (pour faux) (sans apostrophes !). On peut aussi utiliser respectivement `-1` et `0`.

Tout cela sera valable pour toutes les instructions qui transmettront des chaînes de caractères, des nombres, des dates, des booléens, .. Ce peut être une valeur à écrire dans une table (instructions `INSERT` ou `UPDATE`), mais aussi pour créer une expression (pour une recherche par l'instruction `SELECT`, ou pour un calcul).

6.2 - L'instruction SELECT

Elle fait partie du DML. Elle permet de créer des requêtes. Elle se présente sous la forme (simplifiée) :

```
SELECT <NomDeChamp1> [, <NomDeChamp2>, ...] FROM <table> [ WHERE
  Expression ] [ORDER BY <NomDeChamp> { ASC | DESC } ]
```

La clause `WHERE` spécifie les enregistrements à sélectionner : tous les enregistrements pour lesquels `Expression` est vraie seront concernés, et leurs champs spécifiés seront transmis(une clause est une partie de l'instruction qui précise la manière de l'exécuter) ; ici nous avons une clause `WHERE` et une clause `ORDER BY`). Si elle `WHERE` est absente, tous les enregistrements de la table sont sélectionnés.

Expression peut être une combinaison de diverses comparaisons; par exemple :

```
(( ( "Prix" > 40 ) AND ( "Quantite" < 1000 ) ) OR ("Montant" >= 10000))
```

On peut aussi comparer des dates, des chaînes de caractères, ... Pour ces dernières,

- les caractères génériques sont % (qui remplace un nombre quelconque, y compris 0, de caractères) et _ (qui remplace un caractère unique).
- l'opérateur `LIKE` permet de rechercher la présence d'une chaîne à l'intérieur du champ

`"Nom" LIKE 'Du%'` permet de trouver Du, Dupont, Durand, Dubois, ...

`"Nom" LIKE 'Du_'` permet de trouver Duc, Dul mais pas Du ni Durand.

On trouvera le détail des possibilités dans l'aide (en cherchant « Ebauche de requête ») ainsi qu'à l'adresse :

http://hsqldb.sourceforge.net/doc/guide/ch09.html#expression-section

Exemples de requêtes complètes:

La requête suivante nous donnera tous les prénoms des clients dont le champ Nom commence par « Du ».

```
SELECT "Prenom" FROM "Clients" WHERE "Nom" like 'Du%'
```

La prochaine permettra de trouver toutes les commandes où le prix du produit est supérieur à 40 € et en même temps la quantité commandée est inférieure à 1000 pièces, ainsi que celles où le montant est au moins de 10000€ :

```
SELECT "Numero" FROM "Commandes" WHERE (( ( "Prix" > 40 ) AND
  ( "Quantite" < 1000 ) ) OR ("Montant" >= 10000))
```

Pour trouver facilement la syntaxe des requêtes SQL, on peut créer la requête dans l'assistant ; quand la requête est testée et fonctionne correctement, on désactive l'affichage de l'assistant (Requête => Modifier puis Affichage => (Dés)activer le mode Ébauche)

La manière de récupérer le résultat de la requête dans Basic est expliquée au chapitre 17 du livre « Programmation OpenOffice.org2 ». Sans faire double emploi avec le livre, voici un aperçu de la méthode :

```
Sub ObtenirResultatRequete
    Dim oRequete As Object, oResultat As Object
    Dim sInstrSQL As String
    Dim bOK As Boolean
```

```
    ConnecterSource
    sInstrSQL = "SELECT ""Nom"" FROM ""Clients"" WHERE ""ID"" <=
12"
    oRequete = oMaConnexion.createStatement()
    oResultat = oRequete.executeQuery( sInstrSQL )
    bOK = oResultat.next  ' accès à la première ligne
    If bOK then
        MsgBox oResultat.Columns.getByName("Nom").String
     else
        MsgBox "Problème à l'exécution !"
    End if
End Sub
```

L'objet `oResultat` contient l'ensemble des ligne correspondant à la requête. On y accède ligne par ligne (enregistrement par enregistrement). On peut se déplacer dans cet ensemble par les méthodes `first`, `last`, `next`, `previous`, `Bookmark(vSignet)` (pour mémoriser une position dans la variable `vSignet`), `moveToBookmark`, `moveRelativeToBookmark`. On peut tester la position où on se trouve par les méthodes`isFirst`, `isLast`, `isBeforeFirst`, `isAfterLast`.

6.3 - L'instruction UPDATE

Elle aussi fait partie du DML. Elle sert à modifier un enregistrement existant.

Sa syntaxe est :

```
UPDATE <table> SET <champ1> = expression1 [ , <champ2> = expression2,
  ...] [ WHERE expression ]
```

Cette instruction change la valeur des champs indiqués dans la table en question.

Exemple :

```
UPDATE "Clients" SET "Nom" = 'Dubois'  WHERE "ID" = 12
```

Le livre cité indique que l'instruction `UPDATE` ne fonctionne pas ; il semble que maintenant (avec la version 2.0.3) elle fonctionne même avec OOo Basic.

6.4 - L'instruction DELETE

Elle fait partie du DML. Elle permet de supprimer des enregistrements dans une table.

La syntaxe est :

```
DELETE  FROM <table> [ WHERE expression ]
```

Par exemple, pour supprimer le client numéro 12:

```
DELETE FROM "Clients" WHERE "ID" = 12
```

Si vous omettez la clause `WHERE`, HSQLDB supprimera **tous** les enregistrements, sans état d'âme, sans vous demander confirmation, et sans possibilité de récupérer ce qui a été effacé !

Attention donc à ce genre d'instruction. Faites des sauvegardes de vos tables avant de faire vos essais.

6.5 L'instruction CREATE

C'est une instruction DDL (Data Definition Language, Langage de Définition des Données). Le mot-clef `CREATE` est suivi de l'indication de ce qu'on veut créer: `TABLE, INDEX, ALIAS, ...`

Pour créer une table, on utilisera la syntaxe suivante (simplifiée)

```
CREATE TABLE <table> ( <NomDeChamp> type[ ( dimension [, precision])]
  [IDENTITY] [PRIMARY KEY] [[NOT] NULL] [ , ...] )
```

`IDENTITY` signifie que le champ est auto-incrémenté, et c'est automatiquement une clef primaire (donc on ne peut avoir qu'un seul champ de ce type par table). C'est obligatoirement un entier (`INTEGER` ou `BIGINT`).

Exemple :

```
CREATE TABLE "Clients" ( "ID" INTEGER IDENTITY, "Nom" VARCHAR(50),
  "Prenom" VARCHAR(50) )
```

Ceci nous créera une table avec trois champs : un champ entier auto-incrémenté clef primaire, et deux champs alphabétiques d'au maximum 50 caractères chacun.

Remarquez l'écriture `"Prenom"` et non pas `"Prénom"` pour les raisons indiquées plus haut.

6.6 - L'instruction DROP

C'est aussi une instruction DDL. Elle sert à supprimer un objet (table, index, ...). Syntaxe (pour suppression d'une table):

```
DROP TABLE <table> [ IF EXISTS ] [ RESTRICT | CASCADE ]
```

Le `IF EXISTS` évite le message d'erreur si la table n'existe pas. `RESTRICT` signifie que l'instruction ne sera pas effectuée si une vue ou une table se réfère à la table à effacer ; au contraire, `CASCADE` provoquera l'effacement des vues sur cette table et des contraintes qui lie cette table avec d'autres, sans message d'alerte.

On pourra par exemple supprimer la table créée ci-dessus:

```
DROP TABLE "Clients"
```

Là encore, **attention** à ne pas effacer une table contenant le stock magasin de votre entreprise, avec 15.000 références ! Faites des sauvegardes avant de faire vos essais, ou, mieux, travaillez sur une copie.

6.7 - L'instruction ALTER TABLE

C'est une instruction DDL. Elle sert à ajouter un champ ou a supprimer dans une table.

La syntaxe (simplifiée) de la première forme (celle qui permet d'ajouter un champ) ressemble à celle de `CREATE TABLE`:

```
ALTER TABLE <table> ADD [COLUMN] <NomDeChamp> type [ ( dimension [,
  precision])] [NOT]NULL] [IDENTITY] [PRIMARY KEY] ) [BEFORE <Champ
  Existant>]
```

Si on utilise la clause `BEFORE` le champ sera ajouté à l'emplacement indiqué, sinon il sera ajouté à la fin. Exemple :

```
ALTER TABLE "Clients" ADD "Titre" VARCHAR(5) BEFORE "Nom"
```

Une autre forme de cette instruction, permettant cette fois de supprimer un champ existant

```
ALTER TABLE <table> DROP [COLUMN] <NomDeChamp>
```

Exemple :

```
ALTER TABLE "Clients" DROP "Titre"
```

Ainsi, avec ces deux formes de l'instruction `ALTER TABLE`, on peut ajouter et supprimer des champs dans les tables.

6.8 - Les instructions COMMIT, ROLLBACK et SAVEPOINT et les TRANSACTIONS

Les instructions COMMIT, ROLLBACK et SAVEPOINT sont des instructions TCL (Transaction Control Language, Langage de Contrôle des Transactions), car elles permettent d'effectuer des transactions.

Elles sont utilisées par exemple lorsque plusieurs instructions SQL doivent être exécutées, et que si l'une d'elle ne l'est pas à cause d'une erreur la base de données ne sera plus cohérente.

Dans ce cas, on commence par placer un point de sauvegarde (par `SAVEPOINT`) ; puis on envoie les instructions, en gérant les erreurs ; en cas d'erreur on revient à l'état du point de sauvegarde (par `ROLLBACK`) ; si tout s'est bien passé, on valide le tout par `COMMIT`. Voir le Hsqldb User Guide pour la syntaxe.

Une base de données est généralement accessible à plusieurs utilisateurs en même temps. Supposez une base de données pour gérer les places d'avion. S'il reste une seule place dans l'avion et que 2 utilisateurs se connectent en même temps sur la base, comment être sûr que les deux ne vont pas la réserver ? Il faut pour cela une technique appelée « **transaction** » : le premier qui se connecte commence une transaction, et toutes les instructions qui vont suivre seront considérées comme une instruction unique et indivisible ; ainsi, le second utilisateur ne pourra pas réserver avant que la transaction du premier soit terminée, et à ce moment il recevra l'information qu'il n'y a plus de places.

Pour réaliser cela, on utilise les instructions `COMMIT` et `ROLLBACK`, mais il faut avant cela une instruction `SET AUTOCOMMIT FALSE` (qu'on devra effectuer indirectement ; voir le Hsqldb User Guide (Bibliographie), ceci sortant du cadre de cette initiation).

Voir aussi au sujet des transactions un excellent article (« *A quoi ça sert ???* ») sur le site Developpez.com, à l'adresse http://sql.developpez.com/sqlaz/techniques/

Remarque

La description de ces instructions est volontairement simplifiée et ne donne qu'une petite partie des possibilités qu'offre le langage SQL et le moteur de base de données HSQLDB. De plus, il existe bien d'autres instructions SQL (moins utilisées), qu'on trouvera dans le Hsqldb User Guide (Bibliographie). Cependant il n'est pas certain que toutes fonctionnent avec OOo Basic ; il faudra donc faire des essais

7 - Les fonctions et procédures intégrées dans HSQLDB

7.1 - Fonctions d'agrégation

Il s'agit de fonctions qui utilisent les valeurs provenant de **plusieurs** enregistrements d'une table. Par exemple, la fonction `MAX` calcule le valeur maximale d'un champ pour tous les enregistrements sélectionnés (par une clause `WHERE`)

Les fonctions suivantes sont disponibles : `COUNT` (nombre d'enregistrements dans la table), `SUM` (somme des valeurs), `AVG` (valeur moyenne), `MAX` et `MIN` (valeurs maximale et minimale).

```
SELECT MAX("Montant") FROM "Clients"
```

L'instruction `COUNT` est particulière car elle ne porte pas sur un champ, mais sur l'ensemble de l'enregistrement ; on l'utilise ainsi :

```
SELECT COUNT(*) FROM "Clients" WHERE "Nom" LIKE 'Du%'
```

7.2 - Fonctions et procédures numériques

On peut faire effectuer des calculs sur des variables numériques. Pour cela, un grand nombre de fonctions et procédures sont intégrées à HSQLDB : fonctions trigonométriques, exponentielles et logarithmes, arrondis, ...

Par exemple, la fonction `FLOOR(Nombre)` permet l'arrondi.

Voir le Manuel pour connaître toutes ces fonctions et leur emploi.

7.3 - Fonctions et procédures pour les chaînes de caractères

HSQLDB sait effectuer divers calculs sur des chaînes de caractères. En particulier, on peut concaténer des chaînes par l'opérateur ||(voir un exemple dans le paragraphe «Alias »)

Il existe beaucoup d'autres fonctions intégrées pour les chaînes de caractères, voir le Manuel.

7.4 - Fonctions et procédures pour les dates et les heures

On peut obtenir la date courante (`CURDATE()`) et l'heure courante (`CURTIME()`), la différence entre deux dates, le jour, le mois, l'année correspondant à une date, ... Le Manuel donnera tous les détails nécessaires.

7.5 - Autres fonctions et procédures

Il en existe bien d'autres, pour toutes sortes d'usages.

Une fonction souvent utile permet de convertir une donnée en un autre type de données (fonction `CAST ( valeur AS type )`) ; d'autres permettent de tester si une expression est nulle ou a une valeur donnée, ... Voir le Manuel pour connaître ces fonctions.

La fonction `SCRIPT` permet de créer un fichier texte décrivant la base de données en une suite d'instructions SQL et permettant de la recréer à l'identique. Ce fichier contient toutes les instructions SQL générées par Base : un **bon moyen** pour les connaître et les analyser. Il suffit d'envoyer l'instruction :

```
SCRIPT 'MonScript.txt'
```

et le fichier texte sera créé dans le répertoire où se trouve la base de données.

8 - Les ALIAS

On peut demander à recevoir la valeur d'un champ sous un « **alias** » (un nom d'emprunt). Le mot-clef est `AS`.

Une utilisation possible est de faire afficher le nom du champ en français alors que le nom d'origine est en langue étrangère, ou peu explicite:

```
SELECT "ZipCode" AS "CodePostal" FROM "Adresses"
```

On l'utilise aussi pour pouvoir accéder à un résultat d'un calcul (c'est la seule manière d'y parvenir) :

```
SELECT "Prenom" || ' ' || "Nom" AS "NomComplet" FROM "Clients"
```

```
SELECT SUM ("Quantite" * "Prix" ) AS "Montant" FROM "Commandes" WHERE
  "ID" = 147
```

9 - Bibliographie

GAGNEUX, Benjamin, DUBOIS, Frédéric, et al., Cours SQL dans « Developez.com - Club d'entraide des développeurs francophones », 2004, disponible à l'adresse http://sql.developpez.com/

HSQLDB Development Group, « Hsqldb User Guide », disponible en ligne aux adresses : http://hsqldb.org/web/hsqlDocsFrame.html ou http://hsqldb.sourceforge.net/doc/guide/

MARCELLY, Bernard et GODARD, Laurent, « Programmation OpenOffice.org 2 », Eyrolles, Paris, 2005, 724 pages

NAUDIN, Manuel, « FAQ Base intégrée et requêtes SQL directes », 2006, disponible avec d'autres tutoriels à l'adresse :http://fr.openoffice.org/Documentation/How-to/indexht-base.html

Wiki OpenOffice.org, site de de ressources pour développeurs d'OpenOffice.org, en anglais et aussi pour partie en français,disponible à l'adresse :
http://wiki.services.openoffice.org/wiki/Extensions_development_basic_fr

10 - Crédits

Auteur : **Louis Vidonne**

Remerciements : **à Manuel Naudin, Jean-François Nifenecker et Jean-Louis Argente** pour leurs remarques et suggestions pertinentes

Intégré par : **Tony Galmiche**

Contacts : **Projet Documentation OpenOffice.org - fr.OpenOffice.org**

Traduction :

Historique des modifications**:**

Version	***Date***	***Commentaire***
1.00	28/11/2006	Première version publiée

11 - Licence