



Guide Base

Chapitre 9

Macros

Droits d'auteur

Ce document est protégé par Copyright © 2020 par l'Équipe de Documentation de LibreOffice. Les contributeurs sont nommés ci-dessous. Vous pouvez le distribuer et/ou le modifier sous les termes de la Licence Publique Générale GNU (<https://www.gnu.org/licenses/gpl.html>), version 3 ou ultérieure, ou de la Licence Creative Commons Attribution (<https://creativecommons.org/licenses/by/4.0/>), version 4.0 ou ultérieure.

Toutes les marques déposées citées dans ce guide appartiennent à leurs légitimes propriétaires.

Contributeurs

De cette édition

Pulkit Krishna

Jean-Pierre Ledure

Jean Hollis Weber

Des précédentes éditions

Pulkit Krishna

Alain Romedenne

Jean-Pierre Ledure

Jochen Schiffers

Robert Großkopf

Jost Lange

Hazel Russman

Andrew Pitonyak

Jean Hollis Weber

Traduction

Jean-Michel COSTE

Relecteurs

Patrick AUCLAIR

Retours d'information

Veuillez adresser tout commentaire ou suggestion concernant ce document à la liste de diffusion de l'Équipe de Documentation : doc@fr.libreoffice.org



Note

Tout ce que vous envoyez à la liste de diffusion, y compris votre adresse mail et toute autre information personnelle incluse dans le message, est archivé publiquement et ne peut pas être effacé.

Date de publication et version du logiciel

Publié en avril 2021. Basé sur LibreOffice .

Table des matières

Droits d'auteur.....	2
Contributeurs.....	2
De cette édition.....	2
Des précédentes éditions.....	2
Traduction.....	2
Relecteurs.....	2
Retours d'information.....	2
Date de publication et version du logiciel.....	2
Remarques générales sur les macros.....	6
Macros dans Base.....	8
Utiliser des macros.....	8
Assigner des macros.....	8
Événements produits dans un formulaire lorsque la fenêtre est ouverte ou fermée.....	8
Événements dans un formulaire dans une fenêtre ouverte.....	9
Événements dans un formulaire.....	10
Composants des macros.....	10
Le « cadre » d'une macro.....	11
Définition des variables.....	11
Définition de tableaux (array).....	12
Accéder aux formulaires.....	12
Accéder aux éléments de formulaire.....	13
Accès à la base de données.....	14
Connexion à la base de données.....	14
Commandes SQL.....	15
Commandes SQL pré-préparées avec paramètres.....	15
Lecture et utilisation des enregistrements.....	16
Utiliser des formulaires.....	16
Résultat d'une requête.....	17
Utilisation d'un contrôle.....	18
Naviguer dans un ensemble de données.....	18
Édition d'enregistrements (ajout, modification, suppression).....	19
Modifier le contenu d'un contrôle.....	19
Modifier les lignes (enregistrements) d'un ensemble de données.....	20
Créer, modifier et supprimer des lignes.....	20
Tester et modifier les contrôles.....	21
Noms anglais dans les macros.....	21
L'utilitaire Xray.....	22
Propriétés des formulaires et des contrôles.....	22
Police de caractère.....	22
Formulaire.....	23
Propriétés applicables à tous les contrôles.....	23
S'appliquant à de nombreux types de contrôles.....	23
Champ de texte – autres propriétés (TextField).....	24
Champ numérique (NumericField).....	24
Champ Date (DateField).....	24
Champ durée (TimeField).....	25
Champ de devise (CurrencyField).....	26
Champ formaté (FormattedControl).....	26
Zone de liste (ListBox).....	26

Zone combinée (ComboBox).....	27
Cases à cocher (CheckBox) et Boutons Radio (RadioButton).....	28
Champ de motif (PatternField).....	28
Contrôle de Table (GridControl).....	28
Étiquette ou Texte fixe (Label).....	28
Zones de groupe (GroupBox).....	29
Boutons (CommandButton ou ImageButton).....	29
Barre de Navigation (NavigationBar).....	29
Méthodes pour les formulaires et les contrôles.....	29
Naviguer dans un ensemble de données.....	29
Modification des lignes (enregistrements) de données.....	30
Modifier des valeurs individuelles.....	32
Paramètres des commandes SQL préparées.....	33
Améliorer la convivialité.....	33
Mise à jour automatique des formulaires.....	34
Filtrage des enregistrements.....	34
Filtrage des données via le filtre de formulaires.....	37
Faire défiler les enregistrements avec une barre de défilement.....	38
Adapter les données des champs de textes aux conventions du langage SQL.....	39
Pré-calculs et calculs dans un formulaire.....	40
Pré-calculs grâce à une requête SQL.....	41
Calculs directs dans un formulaire.....	43
Fournir la version actuelle de LibreOffice.....	45
Renvoyer la valeur des zones de liste.....	45
Limiter les listes de sélection en saisissant les premières lettres.....	46
Conversion de dates d'un formulaire en une variable de date.....	49
Recherche d'enregistrements de données.....	50
Recherche dans les formulaires et mise en évidence des résultats.....	53
Vérification orthographique lors de la saisie des données.....	56
Boîtes combinées sous forme de Zone de liste avec une option d'entrée.....	59
Affichage du texte dans les boîtes combinées.....	59
Transférer une valeur de clé étrangère d'une zone combinée vers un champ numérique.....	61
Fonction pour mesurer la longueur de l'entrée zone combinée.....	67
Générer des actions de base de données.....	67
Navigation d'un formulaire à un autre.....	68
Ouvrir des rapports, tableaux, requêtes depuis un formulaire.....	69
Zones de liste hiérarchiques.....	71
Saisie des temps en millisecondes.....	74
Un événement – plusieurs implémentations.....	75
Enregistrer avec confirmation.....	76
Clé primaire à partir du numéro courant et de l'année.....	76
Tâches de base de données développées à l'aide de macros.....	78
Établir une connexion à une base de données.....	78
Copie de données d'une base de données à une autre.....	78
Importation directe des données de Calc.....	80
Accès aux requêtes.....	83
Sécuriser votre base de données.....	83
Compacter les bases de données.....	86

Diminution de l'index de table pour les champs à valeur automatique.....	87
Impression depuis Base.....	88
Impression d'un rapport à partir d'un formulaire interne.....	88
Lancement, formatage, impression directe et fermeture d'un rapport.....	88
Impression de rapports à partir d'un formulaire externe.....	90
Faire un publipostage depuis Base.....	91
Impression via des champs de texte.....	92
Appeler des applications externes pour ouvrir des fichiers.....	93
Appeler un programme de messagerie avec un contenu prédéfini.....	94
Changer le pointeur de la souris lors du parcours d'un contrôle.....	95
Affichage des formulaires sans barre d'outils.....	95
Formulaires sans barre d'outils dans la fenêtre.....	95
Formulaires en mode plein écran.....	97
Lancement des formulaires directement depuis l'ouverture de la base de données.....	98
Accéder à une base de données MySQL avec des macros.....	98
Code MySQL dans les macros.....	98
Tables temporaires comme stockage intermédiaire individuel.....	98
Dialogues.....	99
Lancement et fin des dialogues.....	99
Boîte de dialogue simple pour saisir de nouveaux enregistrements.....	100
Boîte de dialogue pour modifier les enregistrements dans une table.....	102
Utilisation d'une boîte de dialogue pour nettoyer les mauvaises données dans les tables.....	107
Écrire des macros avec Access2Base.....	115
Le modèle Objet.....	116
Quelques exemples.....	117
Imprimer une liste de noms de tables et de champs.....	117
Stocker les données produites par une requête dans un tableau Base ou un tuple Python.....	118
Définir les valeurs par défaut dans les entrées de formulaire.....	118
Fonctions de base de données.....	118
Commandes spéciales.....	119
L'objet "Basic" en Python.....	119
Index des Procédures et Fonctions.....	120

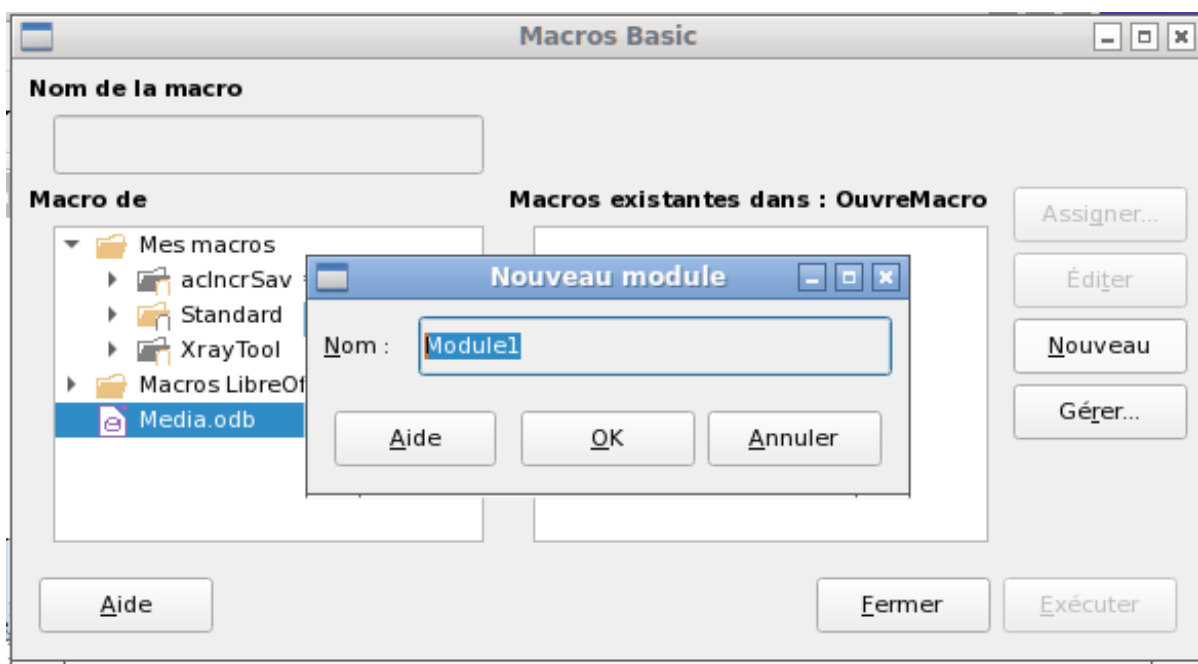
Remarques générales sur les macros

En principe, une base de données dans Base peut être gérée sans macros. Parfois, cependant, elles peuvent devenir nécessaires pour :

- Prévenir plus efficacement les erreurs de saisie ;
- Simplifier certaines tâches de traitement (passage d'un formulaire à un autre, mise à jour des données après saisie dans un formulaire, etc.) ;
- Permettre à certaines commandes SQL d'être appelées plus facilement qu'avec l'éditeur SQL séparé.

Vous devez décider vous-même de l'intensité avec laquelle vous souhaitez utiliser les macros dans Base. Les macros peuvent améliorer la convivialité mais sont toujours associées à de petites réductions de la vitesse du programme, et parfois à des plus grandes (lorsqu'elles sont mal codées). Il est toujours préférable de commencer par utiliser pleinement les possibilités de la base de données et les dispositions de configuration des formulaires avant d'essayer de fournir des fonctionnalités supplémentaires avec des macros. Les macros doivent toujours être testées sur des bases de données plus volumineuses pour déterminer leur effet sur les performances.

Les macros sont créées en utilisant **Outils> Macros> Gérer les macros> Basic**. Une fenêtre apparaît qui donne accès à toutes les macros. Pour Base, la zone importante correspond au nom de fichier du fichier Base.



Le bouton **Nouveau** de la boîte de dialogue Macros Basic ouvre la boîte de dialogue **Nouveau module**, qui demande le nom du module (le dossier dans lequel la macro sera classée). Le nom peut être modifié ultérieurement si vous le souhaitez.

Dès que cela est fait, l'éditeur de macro apparaît. Sa zone de saisie contient déjà le début et la fin d'un sous-programme :

```
REM ***** BASIC *****  
  
Sub Main  
  
End Sub
```

Si des macros doivent être utilisées, les étapes suivantes sont nécessaires :

- Sous **Outils> Options> LibreOffice > Sécurité> Sécurité des macros**, le niveau de sécurité doit être réduit à Moyen. La sécurité pourra être augmentée avec l'expérience. Si nécessaire, vous pouvez en outre utiliser l'onglet *Sources de confiance* pour définir le chemin d'accès à vos propres fichiers contenant des macros afin d'éviter les demandes d'activation ultérieures sur l'activation des macros.
- Les macros sont enregistrées au niveau où elles sont créées (Mes macros, Macros LibreOffice, fichier *.odb). Les macros adaptées à la gestion d'une base seront créées dans le fichier « .odb » de cette base

Quelques principes de base pour l'utilisation du code Basic dans LibreOffice :

- Les lignes n'ont pas de numéros de ligne, par défaut (il existe une option pour activer la numérotation : **Affichage > Numéro de ligne**) et doivent se terminer par un retour forcé. Elles peuvent également se terminer par un espace suivi du caractère « _ » et un retour forcé, soit parce que la ligne de code est trop longue, soit pour apporter plus de lisibilité au code.
- Les fonctions, expressions réservées et éléments similaires ne sont pas sensibles à la casse. Ainsi, « String » est identique à « STRING » ou « string » ou à toute autre combinaison de majuscules et minuscules. La casse ne doit être utilisée que pour améliorer la lisibilité. Cependant, les noms des constantes et des énumérations sont sensibles à la casse la première fois qu'ils sont vus par le compilateur de macros, il est donc préférable de toujours les écrire en utilisant la casse appropriée.
- On peut distinguer les procédures (commençant par Sub) et les fonctions (commençant par Function). Les procédures étaient à l'origine des segments de programme sans valeur de retour, tandis que les fonctions renvoient des valeurs qui peuvent être traitées ultérieurement. Mais cette distinction devient de moins en moins pertinente. Les gens utilisent aujourd'hui des termes tels que « méthode » ou « routine », qu'il y ait une valeur de retour ou non. Une procédure peut également avoir une valeur de retour (en dehors de « Variant »).

```
Sub maProcédure As Integer
End Sub
```

Pour plus de détails, reportez-vous au chapitre 13, Prise en main des macros, dans le Guide de mise en route.



Note

Les macros des versions PDF et ODT de ce chapitre sont colorées selon les règles de l'éditeur de macro LibreOffice :

```
Macro Identificateur
Macro commentaire
Macro opérateur
Macro Mot clé
Macro nombre
Macro chaîne de caractères
```

Macros dans Base

Utiliser des macros

La « voie directe », en utilisant **Outils> Macros> Exécuter la macro** est possible, mais pas habituelle pour les macros pour Base. Une macro est normalement affectée à un événement et lancée lorsque cet événement se produit. Les macros sont utilisées pour :

- Gestion des événements dans les formulaires

- Modifier une source de données dans un formulaire
- Basculer entre les contrôles de formulaire
- Réagir à ce que fait l'utilisateur à l'intérieur d'un contrôle

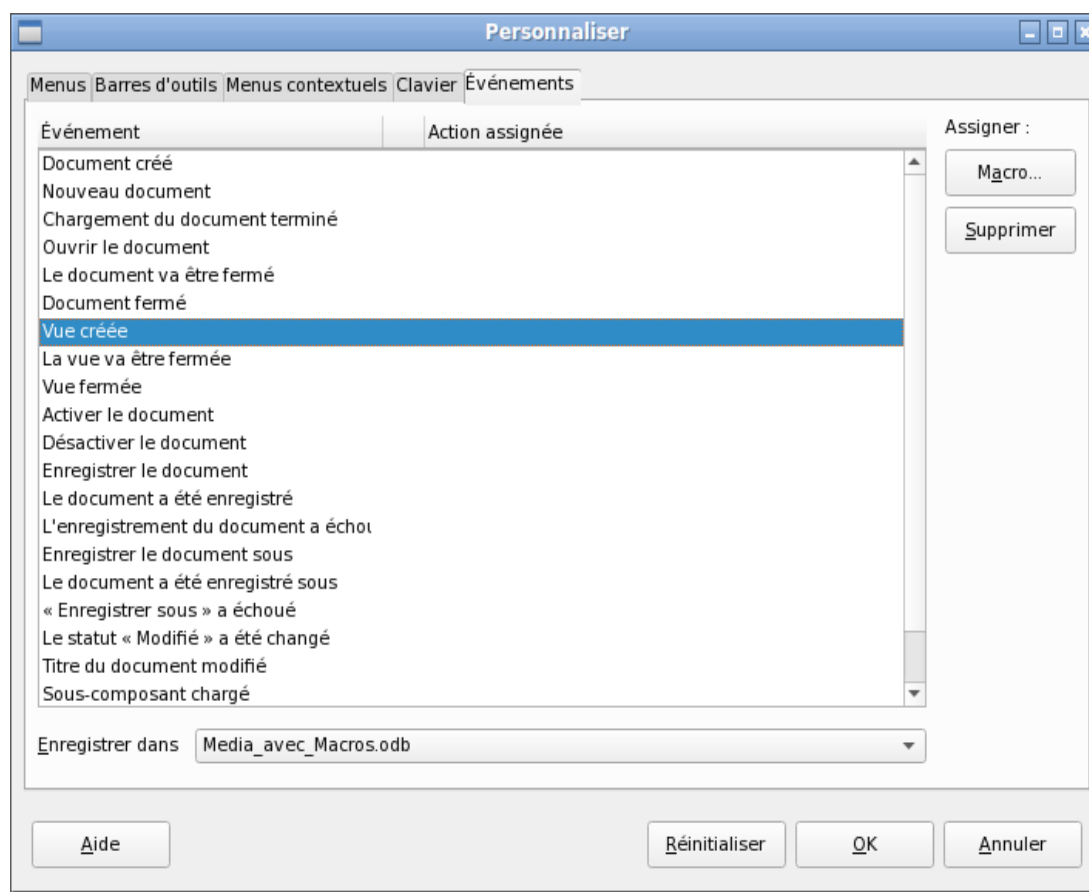
La "voie directe" n'est pas utilisable – pas même pour les tests – lorsque l'un des objets **thisComponent** (voir « Accéder aux formulaires » à la page 12) ou **oEvent** (voir « Accéder aux éléments de formulaire » à la page 13) doit être utilisé.

Assigner des macros

Si une macro doit être lancée par un événement, elle doit d'abord être définie. Ensuite, elle peut être affectée à un événement. Ces événements sont accessibles via deux emplacements.

Événements produits dans un formulaire lorsque la fenêtre est ouverte ou fermée

Les actions qui ont lieu lors de l'ouverture ou de la fermeture d'un formulaire sont enregistrées comme suit :



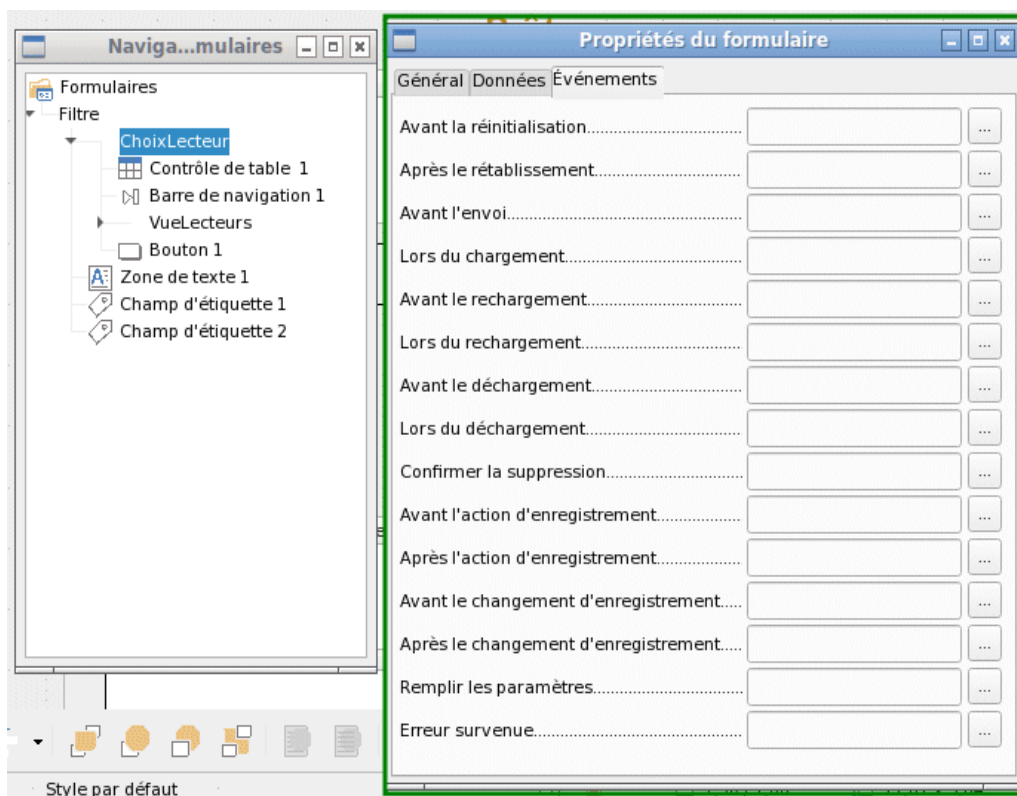
- 1) Lors de la conception du formulaire, ouvrez l'onglet Événements dans **Outils> Personnaliser**.
- 2) Choisissez l'événement approprié. Certaines macros ne peuvent être lancées que lorsque l'événement Vue créée est choisi. D'autres macros, par exemple pour créer un formulaire plein écran, doivent être lancées par Ouvrir le document.
- 3) Utilisez le bouton **Macro** pour trouver la macro souhaitée et confirmez votre choix.
- 4) En face de Enregistrer dans, choisissez le nom du formulaire.
- 5) Confirmez avec **OK**.

Événements dans un formulaire dans une fenêtre ouverte

Une fois la fenêtre ouverte pour afficher le contenu global du formulaire, des éléments individuels du formulaire sont accessibles. Cela inclut les éléments que vous avez attribués au formulaire.

Les éléments de formulaire sont accessibles à l'aide du navigateur de formulaires, comme indiqué dans l'illustration ci-dessous. Ils sont également accessibles en utilisant les menus contextuels des contrôles individuels dans l'interface du formulaire (clic droit sur le contrôle / Propriétés du formulaire).

Les événements répertoriés sous **Propriétés du formulaire > Événements** ont tous lieu lorsque la fenêtre du formulaire est ouverte. Ils peuvent être définis séparément pour chaque formulaire ou sous-formulaire dans la fenêtre de formulaire.



Note

Malheureusement, Base utilise le mot « formulaire » à la fois pour une fenêtre ouverte pour la saisie de données et pour les éléments de cette fenêtre liés à une source de données spécifique (table ou requête).

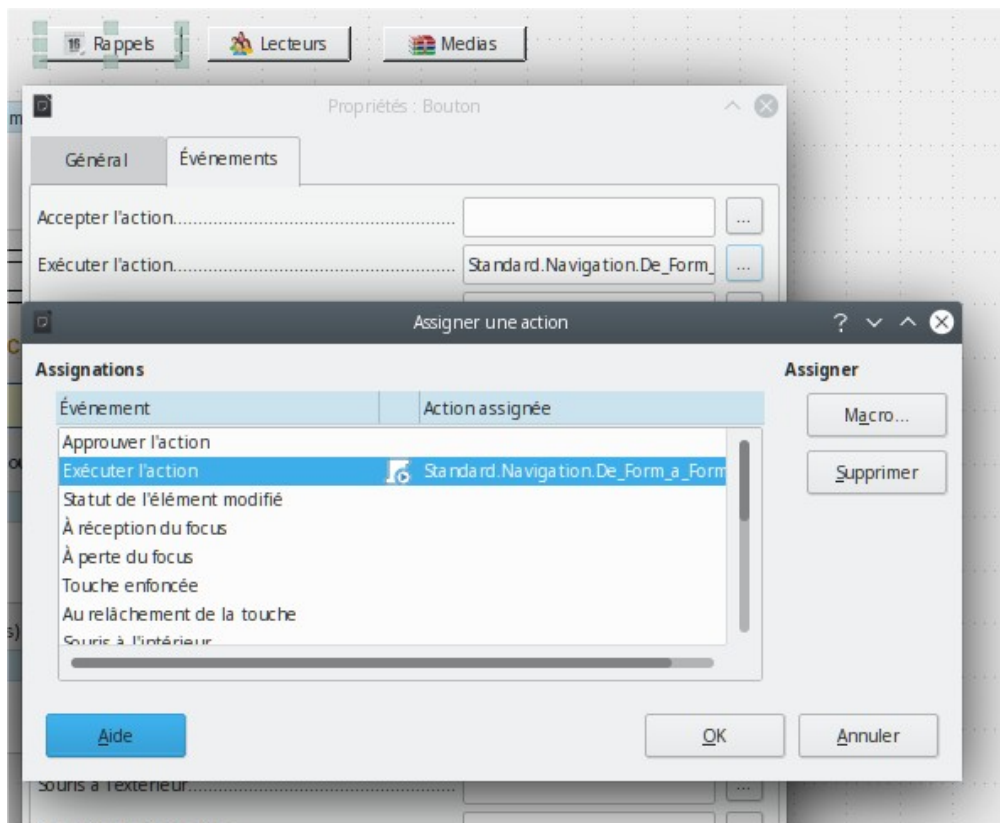
Une seule fenêtre de formulaire peut contenir plusieurs formulaires avec différentes sources de données. Dans le navigateur de formulaires, vous voyez toujours d'abord le terme Formulaires, qui, dans le cas d'un formulaire simple, ne contient qu'une seule entrée subordonnée.

Événements dans un formulaire

Toutes les autres macros sont connectées à l'aide des propriétés des sous-formulaires et des contrôles via l'onglet Événements.

- 1) Ouvrez la fenêtre des propriétés du champ/contrôle (si vous ne l'avez pas déjà fait).

- 2) Choisissez un événement approprié dans l'onglet Événements.
- 3) Pour modifier la source de données, utilisez des événements faisant référence à *Enregistrer* ou *Actualiser* ou *Réinitialiser*.
 - Pour les boutons ou les choix dans les champs de liste ou d'option, l'action *Exécuter l'action* sera le premier point d'entrée.
 - Tous les autres événements dépendent du type de contrôle et de l'action souhaitée.
- 4) Cliquez sur le bouton [...] à droite pour ouvrir la boîte de dialogue Assigner une action.
- 5) Cliquez sur le bouton Macro... pour choisir la macro définie pour l'action.
- 6) Cliquez sur **OK** pour confirmer l'affectation.



Composants des macros

Cette section explique certains des langages de macros couramment utilisés dans Base, en particulier dans les formulaires. Dans la mesure du possible (et du raisonnable), des exemples sont donnés dans toutes les sections suivantes.

Le « cadre » d'une macro

La définition d'une macro commence par son type – **Sub** ou **Function** – et se termine par **End Sub** ou **End Function**. Une macro affectée à un événement peut recevoir des arguments (valeurs), le seul utile est l'argument **oEvent**. Toutes les autres routines qui pourraient être appelées par une telle macro peuvent être définies avec ou sans valeur de retour, selon leur objectif, et fournies avec des arguments si nécessaire.

```
Sub Miseajour_pret
End Sub

Sub de_Formulaire_a_Formulaire(oEvent As Object)
```

```

End Sub

Function confirme_efface(oEvent As Object) As Boolean
    confirme_efface = False
End Function

```

Il est utile d'écrire ce cadre immédiatement et de rédiger le contenu par la suite. N'oubliez pas d'ajouter des commentaires pour expliquer la macro, en vous rappelant la règle "Autant que nécessaire, aussi peu que possible" en gardant à l'esprit qu'un code bien documenté est plus facile à lire. De plus, Basic ne fait pas la distinction entre les majuscules et les minuscules. Les termes habituellement fixes comme **SUB** sont écrits de préférence avec une majuscule, les autres concepts en casse mixte.

Définition des variables

Dans l'étape suivante, au début de la routine, la commande **Dim** est utilisée pour définir les variables qui seront employées dans la routine, chacune avec son type de données approprié. Basic lui-même ne l'exige pas ; il accepte toutes les nouvelles variables utilisées dans le programme sans avoir été définies (déclarées) au préalable. Cependant, le code du programme est « plus sûr » si les variables, en particulier leurs types de données, sont déclarées. De nombreux programmeurs en font une exigence, en utilisant l'instruction **Option Explicit** de Basic lorsqu'ils commencent à écrire un module. Cela signifie : Ne reconnaissez aucune variable, hormis celles que j'ai déclarées. Cela est plus contraignant, mais peut éviter des "plantages catastrophiques" et permet une recherche d'erreurs plus aisée.

```

Dim oDoc As Object
Dim oDrawpage As Object
Dim oFormulaire As Object
Dim sNom As String
Dim bOKActive As Boolean
Dim iCompteur As Integer
Dim dDateNais As Date

```

Seuls les caractères alphabétiques (A-Z ou a-z), les nombres et le caractère de soulignement « _ » peuvent être utilisés dans les noms de variables. Aucun caractère spécial n'est autorisé. Les espaces sont autorisés sous certaines conditions, mais il faut les éviter, pour ne pas créer de confusions. Le premier caractère doit être alphabétique.

Il est courant de spécifier le type de données dans le premier caractère¹. Ensuite, il peut être reconnu partout où la variable apparaît dans le code. Les « noms expressifs » sont également recommandés, de sorte que la signification de la variable soit évidente à partir de son nom.

Une liste des types de données possibles dans LibreOffice Basic se trouve dans l'annexe A de ce livre. Ils diffèrent à divers endroits des types de la base de données et de l'API LibreOffice. Ces changements sont clairement indiqués dans les exemples.

Définition de tableaux (array)

Pour les bases de données en particulier, l'assemblage de plusieurs variables dans un enregistrement est important. Si plusieurs variables sont stockées ensemble dans un seul emplacement commun, cela s'appelle un tableau. Un tableau doit être défini avant de pouvoir y écrire des données.

```
Dim arDonnees()
```

crée un tableau vide.

```
arDonnees = Array("Michèle", "Morgan")
```

crée un tableau d'une taille spécifique (2 éléments) et lui fournit des valeurs.

1 Vous devriez rendre cela plus précis si nécessaire, car une seule lettre ne vous permet pas toujours de faire la distinction entre les types de données "Double" et "Date" ou "Simple" et "String".

En utilisant :

```
Print arDonnees(0), arDonnees(1)
```

on provoque l'affichage à l'écran des deux éléments définis. La numérotation des éléments commence à 0.

```
Dim arDonnees(2)
arDonnees(0) = "Michèle"
arDonnees(1) = "Morgan"
arDonnees(2) = "Paris"
```

Cela crée un tableau dans lequel trois éléments de n'importe quel type peuvent être stockés, par exemple un enregistrement pour : Michèle, Morgan, Paris. Vous ne pouvez pas mettre plus de trois éléments dans ce tableau. Si vous souhaitez stocker plus d'éléments, vous devez agrandir le tableau. Cependant, si la taille d'un tableau est redéfinie pendant l'exécution d'une macro, le tableau est réinitialisé à vide, tout comme un nouveau tableau.

```
ReDim Preserve arDonnees(3)
arDonnees(3) = "29/02/1920"
```

L'ajout de **Preserve** conserve les données précédentes pour que le tableau soit véritablement étendu par l'entrée de la date (ici sous forme de texte).

Le tableau ci-dessus ne peut stocker qu'un seul enregistrement. Si vous souhaitez stocker plusieurs enregistrements, comme le fait une table dans une base de données, vous devez définir un tableau à deux dimensions.

```
Dim arDonnees(2,1)
arDonnees(0,0) = "Michèle"
arDonnees(1,0) = "Morgan"
arDonnees(2,0) = "Paris"
arDonnees(0,1) = "Jean"
arDonnees(1,1) = "Gabin"
arDonnees(2,1) = "Neuilly"
```

Ici aussi, il est possible d'étendre le tableau précédemment défini et de conserver le contenu existant en utilisant **Preserve**.

Accéder aux formulaires

Le formulaire se trouve dans le document actuellement actif. La région représentée ici est appelée **drawpage**. Le conteneur dans lequel tous les formulaires sont conservés est appelé **forms** ; dans le navigateur de formulaires, cela apparaît comme l'en-tête principal (*Formulaires*) avec tous les formulaires individuels attachés. Les variables nommées ci-dessus reçoivent leurs valeurs comme ceci :

```
oDoc = thisComponent
oDrawpage = oDoc.drawpage
oFormulaire = oDrawpage.forms.getByName("Filtre")
```

Le formulaire auquel accéder est appelé *Filtre*. C'est le nom qui est visible dans le niveau supérieur du navigateur de formulaires (par défaut, le premier formulaire est appelé *Formulaire*). Les sous-formulaires sont classés par ordre hiérarchique dans le formulaire principal et peuvent être atteints étape par étape :

```
Dim oSousFormulaire As Object
Dim oSousSousFormulaire As Object
oSousFormulaire = oFormulaire.getByName("SelectionLecteur")
oSousSousFormulaire = oSousFormulaire.getByName("AfficheLecteur")
```

Au lieu d'utiliser des variables intermédiaires, vous pouvez accéder directement à un formulaire particulier. Un objet intermédiaire, qui peut être utilisé plusieurs fois, doit être déclaré et se voir attribué une valeur distincte. Dans l'exemple suivant, oSousFormulaire n'est plus utilisé.

```
oFormulaire = thisComponent.drawpage.forms.getByName("Filtre")
```

```
oSousSousFormulaire = _  
oFormulaire.getByName("SelectionLecteur").getByName("AfficheLecteur")
```



Note

Si un nom se compose uniquement de lettres ASCII et de chiffres sans espaces ni caractères spéciaux, le nom peut être utilisé directement dans une instruction d'affectation.

```
oFormulaire = thisComponent.drawpage.forms.Filtre  
oSubSubForm = oFormulaire.SelectionLecteur.AfficheLecteur
```

Contrairement à l'usage habituel en Basic, ces noms doivent être écrits avec la casse correcte.

Un autre mode d'accès au formulaire est fourni par l'événement qui déclenche la macro.

Si une macro est lancée à partir d'un événement de formulaire tel que **Propriétés du formulaire> Avant l'action d'enregistrement**, le formulaire lui-même peut être atteint comme suit :

```
Sub Calculer(oEvent As Object)  
    oFormulaire = oEvent.Source  
    ...  
End Sub
```

Si la macro est lancée à partir d'un événement dans un contrôle de formulaire, tel que **Zone de texte> A la perte de focus**, le formulaire et le champ deviennent accessibles :

```
Sub Calculer(oEvent As Object)  
    oChamp = oEvent.Source.Model ' récupère la source de l'événement  
    oFormulaire = oChamp.Parent ' récupère le formulaire contenant la source  
    ...  
End Sub
```

L'accès aux événements présente l'avantage que vous n'avez pas à vous soucier de savoir si vous avez affaire à un formulaire principal ou à un sous-formulaire. De plus, le nom du formulaire n'a aucune importance pour le fonctionnement de la macro.

Accéder aux éléments de formulaire

Les éléments dans les formulaires sont accessibles de la même manière : déclarez une variable appropriée en tant qu'**object** et recherchez le contrôle approprié dans le formulaire :

```
Dim btnOK As Object ' Bouton OK  
btnOK = oSousSousFormulaire.getByName("Bouton 1") ' depuis le formulaire VueLecteur
```

Cette méthode fonctionne toujours lorsque vous savez avec quel élément la macro est censée fonctionner. Cependant, lorsque la première étape consiste à déterminer quel événement a lancé la macro, la variable **oEvent** présentée ci-dessus devient utile. La variable est déclarée dans le « cadre » de la macro et se voit attribuer une valeur lorsque la macro est lancée. La propriété **Source** renvoie toujours l'élément qui a lancé la macro, tandis que la propriété **Model** décrit le contrôle en détail :

```
Sub Confirme_Choix(oEvent As Object)  
    Dim btnOK As Object  
    btnOK = oEvent.Source.Model  
End Sub
```

Si vous le souhaitez, d'autres actions peuvent être effectuées avec l'objet obtenu par cette méthode.

Veuillez noter que les sous-formulaires comptent comme composants d'un formulaire.

Accès à la base de données

Normalement, l'accès à la base de données est contrôlé par des formulaires, des requêtes, des rapports ou la fonction publipostage, comme décrit dans les chapitres précédents. Si ces possibilités s'avèrent insuffisantes, une macro peut spécifiquement accéder à la base de données de plusieurs manières.

Connexion à la base de données

La méthode la plus simple utilise la même connexion que le formulaire. `oFormulaire` est déterminé comme indiqué ci-dessus.

```
Dim oConnexion As Object
oConnexion = oFormulaire.activeConnection()
```

Ou vous pouvez récupérer la source de données (c'est-à-dire la base de données) via le document et utiliser sa connexion existante pour la macro :

```
Dim oSourceDonnees As Object
Dim oConnexion As Object
oSourceDonnees = thisComponent.Parent.dataSource
oConnexion = oSourceDonnees.getConnection("", "") 'sans mot de passe
```

Un autre moyen permet de créer la connexion à la base de données à la volée :

```
Dim oSourceDonnees As Object
Dim oConnexion As Object
oSourceDonnees = thisComponent.Parent.CurrentController
If Not (oSourceDonnees.isConnected()) Then oSourceDonnees.connect()
oConnexion = oSourceDonnees.ActiveConnection()
```

La condition **If** ne contrôle qu'une seule ligne, donc **End If** n'est pas obligatoire. Mais cela peut conduire à des erreurs lorsqu'il s'avère nécessaire d'ajouter un **Else** ou d'autres instructions dans le **Then** par la suite .

Si la macro doit être lancée via l'interface utilisateur et non à partir d'un événement dans un formulaire, la variante suivante convient :

```
Dim oSourceDonnees As Object
Dim oConnexion As Object
oSourceDonnees = thisDatabaseDocument.CurrentController
If Not (oSourceDonnees.isConnected()) Then oSourceDonnees.connect()
oConnexion = oSourceDonnees.ActiveConnection()
```

L'accès aux bases de données en dehors de la base de données actuelle est possible comme suit :

```
Dim oContexteBase As Object
Dim oSourceDonnees As Object
Dim oConnexion As Object
oContexteBase = createUnoService("com.sun.star.sdb. DatabaseContext")
oSourceDonnees = oContexteBase.getByName("nom de la base enregistrée dans LibO")
oConnexion = oSourceDonnees.GetConnection("", "")
```

Des connexions à des bases de données non enregistrées avec LibreOffice sont également possibles. Dans de tels cas, au lieu du nom enregistré, le chemin d'accès à la base de données doit être indiqué sous forme de **file: ///...../NomdeLaBase.odt**.

Des instructions détaillées sur les connexions à la base de données sont données dans « Établir une connexion à une base de données » (page 78).

Commandes SQL

Vous travaillez avec la base de données à l'aide de commandes SQL. Celles-ci doivent être créées et envoyées à la base de données. Le résultat est déterminé en fonction du type de

commande et les résultats peuvent être traités ultérieurement. La directive **createStatement** crée un objet approprié à cet effet.

```
Dim oInstruction_SQL As Object ' l'objet qui contiendra la commande SQL-
Dim stSql As String ' Texte de la commande SQL actuelle
Dim oResultat As Object ' résultat de la fonction executeQuery
Dim iResultat As Integer ' résultat de la fonction executeUpdate
oInstruction_SQL = oConnexion.createStatement()
```

Pour *interroger des données*, vous appelez la méthode **executeQuery** ; le résultat est ensuite évalué. Les noms de table et de champ sont généralement entre guillemets. La macro doit les masquer avec des guillemets doubles supplémentaires pour s'assurer qu'elles apparaissent dans la commande.

```
stSql = "SELECT * FROM ""Table1""""
oResultat = oInstruction_SQL.executeQuery(stSql)
```

Pour *modifier les données* – c'est-à-dire **INSERT**, **UPDATE** ou **DELETE** – ou pour agir sur la *structure de la base de données*, vous appelez la méthode **executeUpdate**. Selon la commande et la base de données, cela ne donne rien d'utile (un zéro) ou bien le nombre d'enregistrements modifiés.

```
stSql = "DROP TABLE ""temporaire"" IF EXISTS"
iResultat = oInstruction_SQL.executeUpdate(stSql)
```

Par souci d'exhaustivité, il y a un autre cas particulier à mentionner : si **oInstruction_SQL** doit être utilisé de différentes manières pour **SELECT** ou à d'autres fins, il existe une autre méthode disponible, à savoir **execute**. Nous ne l'utiliserons pas ici. Pour plus d'informations, consultez la référence API ([Xstatement](#), [XPreparedStatement](#)).

Commandes SQL pré-préparées avec paramètres

Dans tous les cas où la saisie manuelle par un utilisateur doit être transférée dans une instruction SQL, il est plus facile et plus sûr de ne pas créer la commande sous la forme d'une longue chaîne de caractères mais de la concevoir à l'avance et de l'utiliser avec des paramètres. Cela facilite le formatage des nombres, des dates et des chaînes (les guillemets doubles constants disparaissent) et empêche les entrées malveillantes (injections SQL) de provoquer une perte de données.

Pour utiliser cette méthode, l'objet d'une commande SQL particulière est créé et préparé :

```
Dim oInstruction_SQL As Object ' l'objet, qui exécute la commande SQL
Dim stSql As String ' Texte de la commande SQL.
stSql = "UPDATE Auteurs " _
& "SET Nom = ?, Prenom = ?" _
& "WHERE ID = ?"
oInstruction_SQL = oConnexion.prepareStatement(stSql)
```

L'objet est créé avec **prepareStatement** afin que la commande SQL soit connue à l'avance. Chaque point d'interrogation indique une position qui plus tard – avant l'exécution de la commande – recevra une valeur réelle. Comme la commande est conçue à l'avance, la base de données sait quel type d'entrée – dans ce cas deux chaînes et un nombre – est attendu. Les différentes positions se distinguent par leur numéro (à partir de 1).

Ensuite, les valeurs sont transférées avec des instructions appropriées et la commande SQL est exécutée. Ici, les valeurs sont tirées des contrôles de formulaire, mais elles peuvent également provenir d'autres macros ou être données sous forme de texte brut :

```
oInstruction_SQL.setString(1, oZoneTexte1.Text) ' Texte pour le nom
oInstruction_SQL.setString(2, oZoneTexte2.Text) ' Texte pour le prénom
oInstruction_SQL.setLong(3, oChampNum1.Value) ' valeur pour l'ID approprié
```

La liste complète des affectations est en "Paramètres des commandes SQL préparées" (page 33).

Pour plus d'informations sur les avantages de cette méthode, voir ci-dessous (liens externes) :

- [Injection SQL \(Wikipedia\)](#)
- [Why use PreparedStatement \(Java JDBC\)](#)
- [Commandes SQL \(Wikilivres\)](#)
- [Utilité des requêtes préparées \(Openclassrooms\)](#)

Lecture et utilisation des enregistrements

Il existe plusieurs façons, selon les besoins, de transférer des informations d'une base de données vers une macro afin qu'elles puissent être traitées ultérieurement.

Remarque : si l'on parle de « formulaire », il peut également s'agir d'un sous-formulaire. Dans ce cas, il s'agit toujours du (sous-)formulaire qui est associé à un certain nombre de données.

Utiliser des formulaires

L'enregistrement courant et ses données sont toujours disponibles via le formulaire qui affiche les données pertinentes (table, requête, SELECT). Il existe plusieurs méthodes **getdata_type**, telles que :

```
Dim ID As Long
Dim sNom As String
Dim dValeur As Currency
Dim dEntree As New com.sun.star.util.Date ' objet UNO de type Date
ID = oFormulaire.getLong(1)
sNom = oFormulaire.getString(2)
dValeur = oFormulaire.getDouble(4)
dEntree = oFormulaire.getDate(7)
```



Note

Toutes les méthodes qui fonctionnent avec des bases de données comptent à partir de 1. Cela s'applique aussi bien aux colonnes qu'aux lignes.

Si vous préférez utiliser des noms de colonne au lieu de numéros de colonne pour travailler avec la source de données sous-jacente (table, requête, vue), le nom de colonne peut être déterminé à l'aide de **findColumn**. Voici un exemple de recherche de la colonne appelée *Nom*.

```
Dim sNom As String
numNom = oFormulaire.findColumn("Nom") ' numéro du champ
stNom = oFormulaire.getString(numNom) ' contenu du champ
```

Le type de valeur renvoyé correspond toujours au type de méthode, mais les cas spéciaux suivants doivent être notés :

- Il n'y a pas de méthodes pour les données des types **Decimal**, **Currency (devise)**, etc. qui sont utilisées pour des calculs commerciaux exacts. Comme Basic effectue automatiquement la conversion appropriée, vous pouvez utiliser **getDouble**
- Lorsque vous utilisez **getBoolean**, vous devez tenir compte de la façon dont TRUE et FALSE sont définis dans la base de données. Les définitions usuelles (valeurs logiques, 1 pour TRUE et 0 pour FALSE) sont traitées correctement.
- Les valeurs de date peuvent être définies non seulement avec le type de données **Date**, mais également comme **util.Date** (comme ci-dessus, objet UNO). Cela facilite la lecture et la modification de l'année, du mois et du jour.
- Avec des nombres entiers, méfiez-vous des différents types de données. L'exemple ci-dessus utilise **getLong** ; la variable Basic ID doit également avoir le type de données **Long**, car il correspond au type **Integer** dans la base de données.

La liste complète de ces méthodes se trouve dans « Modification des lignes (enregistrements) de données » (page 30).



Conseil

Si les valeurs d'un formulaire doivent être utilisées directement pour un traitement ultérieur en SQL (par exemple pour une entrée dans une autre table), il est beaucoup plus simple de ne pas avoir à interroger le type de champ.

La macro suivante, qui est liée à **Propriétés : Bouton> Événements> Exécuter l'action** lit le premier champ du formulaire indépendamment du type nécessaire pour le traitement futur dans Basic.

```
SUB LireValeurs(oEvent As Object)
    Dim oFormulaire As Object
    Dim stChamp1 As String
    oFormulaire = oEvent.Source.Model.Parent
    stChamp1 = oFormulaire.getString(1)
End Sub
```

Si les champs sont lus à l'aide de **getString()**, tout le formatage nécessaire pour un traitement SQL ultérieur est conservé. Une date qui s'affiche sous la forme 08.03.19 est lue au format 2019-03-08 et peut être utilisée directement dans SQL.

La lecture dans un format correspondant au type n'est obligatoire que si la valeur doit être traitée ultérieurement dans la macro, par exemple dans un calcul.

Résultat d'une requête

L'ensemble des résultats d'une requête peut être utilisé de la même manière. Dans la section des commandes SQL et ci-dessus, vous trouverez la variable **oResultat** pour cet ensemble de résultats, qui est généralement lue de cette manière :

```
While oResultat.next ' un enregistrement après l'autre
    REM transfère le résultat dans des variables
    stVar = oResultat.getString(1)
    inVar = oResultat.getLong(2)
    boVar = oResultat.getBoolean(3)
    REM faire quelque chose avec ces valeurs
Wend
```

Selon le type de commande SQL, le résultat attendu et son objectif, la boucle **WHILE** peut être raccourcie ou supprimée complètement. Mais fondamentalement, un jeu de résultats peut toujours être évalué de cette manière.

Si seul le premier enregistrement doit être évalué

```
oResultat.next
```

accède à la ligne de cet enregistrement et avec

```
stVar = oResultat.getString(1)
```

lit le contenu du premier champ. La boucle se termine ici.

La requête de l'exemple ci-dessus contient du texte dans la première colonne, un nombre entier dans la seconde (un **Integer** dans la base de données correspond à **Long** en Basic) et un champ Oui/Non dans la troisième. Les champs sont accessibles via un index de champ qui, contrairement à un index de tableau, commence à 1 (cf. Note page précédente).

La navigation dans un tel résultat n'est pas possible. Seules les étapes simples vers l'enregistrement suivant sont autorisées. Pour naviguer dans l'enregistrement, le **ResultSetType** doit être connu lorsque la requête est créée. Ceci est accessible en utilisant

```
oSQL_Resultat.ResultSetType = 1004
```

OU

```
oSQL_Resultat.ResultSetType = 1005
```

Le type **1004 – SCROLL_INTENSIVE** vous permet de naviguer librement mais ne prend pas en compte les changements dans les données d'origine. Le type **1005 – SCROLL_SENSITIVE** reconnaît les modifications des données d'origine susceptibles d'affecter le résultat de la requête.

Le nombre total de lignes dans l'ensemble de résultats ne peut être déterminé qu'après la spécification d'un type numérique pour le résultat. Il se déroule comme suit :

```
Dim iResultat As Long
If oResultat.last ' va au dernier enregistrement si possible
    iResultat = oResultat.getRow ' iResultat reçoit le nombre de lignes
                                ' qui est le numéro de la dernière.
Else
    iResultat = 0
End If
```

Utilisation d'un contrôle

Si un contrôle est lié à une source de données, la valeur peut être lue directement, comme décrit dans la section suivante. Cependant, cela peut entraîner des problèmes. Il est plus sûr d'utiliser la procédure décrite dans "Utiliser des formulaires" (page 16) ou bien la méthode suivante, qui est présentée pour plusieurs types de contrôle différents :

```
sValue = oZoneTexte1.BoundField.Text ' exemple pour une Zone de Texte
nValue = oChampNumerique.BoundField.Value ' exemple pour un champ numérique
dValue = oChampDate.BoundField.Date ' exemple pour un champ Date
```

BoundField représente le lien entre le contrôle visible et le contenu réel de l'ensemble de données.

Naviguer dans un ensemble de données

Dans l'exemple page précédente (While... Wend), la méthode **Next** a été utilisée pour passer d'une ligne du jeu de résultats à la suivante. Il existe d'autres méthodes et tests similaires qui peuvent être utilisés à la fois pour les données dans un formulaire – représenté par la variable **oFormulaire** – et pour un ensemble de résultats. Par exemple, en utilisant la méthode décrite dans « Mise à jour automatique des formulaires » (page 33), l'enregistrement précédent peut être sélectionné à nouveau :

```
Dim laLigne As Long
laLigne = oFormulaire.getRow() ' enregistre le numéro de ligne courant
oFormulaire.reload() ' recharge l'ensemble d'enregistrements
oFormulaire.absolute(laLigne) ' revient à la même ligne
```

La section « Mise à jour automatique des formulaires » montre toutes les méthodes qui conviennent pour cela.



Note

Malheureusement, depuis le début de LibreOffice, il y a eu un bogue (reporté d'OpenOffice) qui affecte les formulaires. Il définit le numéro de ligne actuel lorsque les données sont modifiées dans un formulaire sur « 0 ». Voir https://bugs.documentfoundation.org/show_bug.cgi?id=82591. Pour obtenir le numéro de ligne actuel correct, liez la macro suivante à l'événement **Formulaire> Propriétés> Événements> Après le changement d'enregistrement**.

```
Global laLigne As Long
Sub CompteurLignes(oEvent As Object)
    laLigne = oEvent.Source.Row
End Sub
```

Le nouveau numéro de ligne est lu et affecté à la variable globale `laLigne`. Cette variable doit être placée au début de tous les modules et conservera son contenu jusqu'à ce que vous quittiez Base ou que vous changiez la valeur en appelant à nouveau `CompteurLignes`.

Il semblerait que ce bogue est corrigé depuis la version 5.0

Édition d'enregistrements (ajout, modification, suppression)

Pour éditer des enregistrements, plusieurs choses doivent fonctionner ensemble :

- Les informations doivent être saisies par l'utilisateur dans un champ, à l'aide du clavier ;
- L'ensemble de données sous-jacent au formulaire doit être informé du changement. Cela se produit lorsque vous quittez le champ pour un autre ;
- La base de données elle-même doit être modifiée. Cela se produit lorsque vous passez d'un enregistrement à un autre.

Lorsque vous effectuez cette opération via une macro, ces étapes partielles doivent toutes être prises en compte. Si l'une d'entre elles fait défaut ou est mal exécutée, les modifications seront perdues et ne finiront pas dans la base de données. Tout d'abord, la modification ne doit pas se trouver uniquement dans la valeur affichée du contrôle mais dans l'ensemble de données lui-même. Cela rend inutile la modification de la propriété **Text** d'un contrôle.

Veuillez noter que les tableaux sont les seuls ensembles de données qui peuvent être modifiés sans causer de problèmes. Pour les autres ensembles de données, l'édition n'est possible que dans des circonstances spéciales.

Modifier le contenu d'un contrôle

Si vous souhaitez modifier une seule valeur, la propriété **BoundField** du contrôle peut être utilisée avec une méthode appropriée. Ensuite, la modification doit être transmise à la base de données. Voici un exemple de champ de date dans lequel la date réelle doit être saisie :

```
Dim unoDate As New com.sun.star.util.Date
unoDate.Year = Year(Date)
unoDate.Month = Month(Date)
unoDate.Day = Day(Date)
oChampDate.BoundField.updateDate(unoDate)
oFormulaire.updateRow() ' la modification est transmise à la base
```

Pour **BoundField**, vous utilisez la méthode **updateXxx** qui correspond au type de données du champ. Dans cet exemple, le champ est un champ **Date**. La nouvelle valeur est transmise comme argument – dans ce cas la date actuelle, convertie au format requis par la macro.

Modifier les lignes (enregistrements) d'un ensemble de données

La méthode précédente ne convient pas lorsque plusieurs valeurs dans une ligne doivent être modifiées, d'une part, un contrôle devrait exister sur le formulaire pour chaque champ, ce qui n'est souvent pas souhaité et inutile. En outre, un objet doit être récupéré pour chaque champ. La manière simple et directe utilise le formulaire comme ceci :

```
Dim unoDate As New com.sun.star.util.Date
unoDate.Annee = Year(Date)
unoDate.Mois = Month(Date)
unoDate.Jour = Day(Date)
oFormulaire.updateDate(3, unoDate)
oFormulaire.updateString(4, "un Texte")
oFormulaire.updateDouble(6, 3.14)
oFormulaire.updateInt(7, 16)
oFormulaire.updateRow()
```

Pour chaque colonne de l'ensemble de données, la méthode **updateXxx** appropriée à son type est appelée. Les arguments sont le numéro de colonne (à partir de 1) et la valeur souhaitée. Ensuite, les modifications sont transmises à la base de données.

Créer, modifier et supprimer des lignes

Les modifications nommées font référence à la ligne actuelle de l'ensemble de données sous-jacent au formulaire. Dans certaines circonstances, il est nécessaire d'appeler une méthode depuis « Naviguer dans un ensemble de données » (page 29). Les étapes suivantes sont nécessaires :

- 1) Choisissez l'enregistrement actuel.
- 2) Modifiez les valeurs comme décrit dans la section précédente.
- 3) Confirmez la modification avec la commande suivante :
`oFormulaire.updateRow()`
- 4) Dans certains cas particuliers, il est possible d'annuler et de revenir à l'état précédent :
`oFormulaire.cancelRowUpdates()`

Pour un nouvel enregistrement, il existe une méthode spéciale, comparable au passage à une nouvelle ligne dans un contrôle de table. Cela se fait comme suit :

- 1) Préparez-vous pour un nouvel enregistrement :
`oFormulaire.moveToInsertRow()`
- 2) Saisissez toutes les valeurs souhaitées/requises. Cela se fait en utilisant les méthodes **updateXxx** comme indiqué dans la section précédente.
- 3) Confirmez les nouvelles données avec la commande suivante :
`oFormulaire.insertRow()`
- 4) La nouvelle entrée ne peut pas être facilement annulée. Au lieu de cela, vous devrez supprimer le nouvel enregistrement.

Il existe une commande simple pour supprimer un enregistrement, procédez comme suit :

- 1) Choisissez l'enregistrement souhaité et rendez le courant, comme pour une modification.
- 2) Utilisez la commande suivante pour le supprimer :
`oFormulaire.deleteRow()`



Conseil

Pour garantir que les modifications sont transférées dans la base de données, elles doivent être confirmées explicitement avec **updateRow** ou **insertRow**, selon le cas. Appuyer sur le bouton Enregistrer utilisera automatiquement la fonction appropriée, avec une macro, vous devez déterminer avant d'enregistrer si l'enregistrement est nouveau (**Insert**) ou une modification d'un existant (**Update**).

```
If oFormulaire.isNew Then
    oFormulaire.insertRow()
Else
    oFormulaire.updateRow()
End If
```

Tester et modifier les contrôles

Outre le contenu de l'ensemble de données, beaucoup plus d'informations peuvent être lues à partir d'un contrôle, éditées et modifiées. Cela est particulièrement vrai pour les propriétés, comme décrit dans le chapitre 4, Formulaires.

Divers exemples dans « Améliorer la convivialité » (page 33) utilisent la propriété *Complément d'information* du champ :

```
Dim stTag As String
stTag = oEvent.Source.Model.Tag
```

Comme mentionné dans la section précédente, la propriété **Text** ne peut être modifiée utilement que si le contrôle n'est pas lié à un ensemble de données. Cependant, il existe d'autres propriétés qui sont déterminées dans le cadre de la définition du formulaire mais qui peuvent être adaptées au moment de l'exécution. Par exemple, une étiquette peut avoir une couleur de texte différente si elle représente un avertissement plutôt qu'une information :

```
Sub MontreAvertissement(oChamp As Object, iType As Integer)
    Select Case iType
        Case 1
            oChamp.TextColor = RGB(0, 0, 255) ' 1 = bleu
        Case 2
            oChamp.TextColor = RGB(255, 0, 0) ' 2 = rouge
        Case Else
            oChamp.TextColor = RGB(0, 255, 0) ' 0 = vert (ni 1 ni 2)
    End Select
End Sub
```

Noms anglais dans les macros

Alors que le concepteur d'un formulaire peut utiliser des désignations en langue native pour les contrôles et l'accès aux données, seuls les noms anglais peuvent être utilisés dans Basic. Ceux-ci sont présentés dans le synopsis suivant.

Les propriétés qui ne sont normalement déterminées que dans la définition de formulaire ne sont pas incluses ici. Il en va de même pour les méthodes (fonctions et/ou procédures) qui ne sont que rarement utilisées ou requises uniquement pour des déclarations plus complexes.

Le synopsis comprend les éléments suivants :

- **Nom** Nom à utiliser pour la propriété dans le code de macro
- **Type de donnée** Un type de données Basic. Pour les fonctions, le type de retour. Non inclus pour les procédures.
- **L/E** Indique comment vous pouvez utiliser la valeur:
 - L Lecture seule (read only)
 - E Écriture seule (modification) (write)
 - (L) Lecture possible, pas adapté pour l'édition
 - (E) Écriture possible mais pas utile
 - L+E Adapté à la lecture et à l'écriture

Vous trouverez de plus amples informations dans la référence API en recherchant le nom anglais du contrôle. Il existe un outil plus qu'utile, indispensable, appelé Xray pour découvrir quelles propriétés et méthodes sont disponibles pour un élément.

L'utilitaire Xray

Xray a été initialement conçue pour Apache Open Office, mais continue de fonctionner sur Libre Office après, quelques aménagements. Cet outil peut-être téléchargé sur la page personnelle de Bernard Marcelly, son concepteur ([Lien vers Xray](#)). Pour l'installation et l'utilisation de Xray se reporter au document « XrayTool60_fr.odt » téléchargé.

Pour la configuration de Xray, se reporter au chapitre 3.5, page 22, de ce document. On y constate qu'il est souhaitable d'installer le SDK (Software development kit – Kit de développement logiciel) de LibreOffice en local (sur l'ordinateur utilisé).

Cette installation n'est pas obligatoire mais à l'avantage de donner accès, en local, à la documentation de l'API de LibreOffice.

Depuis la version 4.2 de LibreOffice, le SDK de LibreOffice ne fournit plus les fichiers html de l'IDL, ni les références croisées des éléments de l'API, utilisés par Xray. Pour pallier à cela, mais malheureusement uniquement sous Windows, il existe un utilitaire (createLibO_IDL.vbs) contenu dans « LibO_IDL_creator pour XRay.zip », à télécharger sur la même page que précédemment.

Sous Windows, il n'est pas utile d'installer le SDK dans un répertoire où l'utilisateur a tous les droits. Il suffit, après l'installation dans le répertoire par défaut (LibreOffice), de modifier les autorisations accordées à l'utilisateur (Clic droit sur le répertoire « sdk » > Propriétés > Onglet sécurité > bouton Modifier > Sélectionner l'utilisateur > Cocher la case Contrôle total).

Il est alors possible d'appliquer les consignes d'utilisation de ce script « .vbs » contenues dans le fichier ReadMe.txt.

On veillera à appliquer la modification du code Basic de Xray indiqué à la dernière ligne de ce fichier ReadMe.

Pour les utilisateurs de Linux, de Mac OS X, et autres, se reporter aux documentations respectives, relatives à la gestion des droits sur les dossiers ou répertoires.

Pour ceux-ci, qui ne peuvent exécuter le script « .vbs », et pour les utilisateurs Windows qui ne souhaitent pas l'exécuter, un fichier « LibO6.4_SDK_Addon4Xray.zip » a été préparé. Il suffit de le dézipper, puis de copier-coller le répertoire « Common » et son contenu dans le sous-répertoire « docs » du répertoire dans lequel le SDK a été installé, par exemple :

- sous Windows 10 : « C:\Program Files\LibreOffice\sdk\docs »
- sous Linux Debian, Ubuntu : « /opt/libreoffice6.4/sdk/docs/ »

Le fichier LibO6.4_SDK_Addon4Xray.zip pourra être trouvé dans les « **Fichiers additionnels** » à la page : https://wiki.documentfoundation.org/FR/Doc#Base_6.4.

Propriétés des formulaires et des contrôles

Le modèle d'un contrôle décrit ses propriétés. Selon la situation, la valeur d'une propriété est accessible en lecture seule ou en écriture seule. L'ordre suit celui des listes de « Propriétés des champs de contrôle » du chapitre 4, Formulaires.

Police de caractère

Dans chaque contrôle qui affiche du texte, les propriétés de police peuvent être personnalisées.

Nom	Type	L/E	Propriété
FontName	string	L+E	Nom de la police
FontHeight	single	L+E	Taille de la police
FontWeight	single	L+E	Gras ou normal
FontSlant	integer	L+E	Italique ou roman
FontUnderline	integer	L+E	Souligné
FontStrikeout	integer	L+E	Barré

Formulaire

Terme anglais : Form

Nom	Type	L/E	Propriété
ApplyFilter	boolean	L+E	Filtre appliqué
Filter	string	L+E	Filtre actuel pour l'enregistrement
FetchSize	long	L+E	Nombre d'enregistrements chargés à la fois
Row	long	L	Numéro de la ligne actuelle
RowCount	long	L	Nombre d'enregistrements

Propriétés applicables à tous les contrôles

Control – voir aussi FormComponent

Nom	Type	L/E	Propriété
Name	string	L+(E)	Nom du champ/contrôle
Enabled	boolean	L+E	Actif : le champ peut être sélectionné
EnableVisible	boolean	L+E	Le champ est affiché
ReadOnly	boolean	L+E	Le contenu du champ ne peut être modifié
TabStop	boolean	L+E	Le champ est accessible via la touche Tab
Align	integer	L+E	Alignement horizontal : 0 = gauche, 1 = centré, 2 = droite
BackgroundColor	long	L+E	Couleur de l'arrière plan
Tag	string	L+E	Information additionnelle
HelpText	string	L+E	Texte d'aide sous forme d'info-bulle

S'appliquant à de nombreux types de contrôles

Nom	Type	L/E	Propriété
Text	string	(L+E)	Contenu affiché du champ. Dans les champs de texte, cela peut être lu et traité ultérieurement, mais cela ne fonctionne généralement pas pour les autres types.
Spin	boolean	L+E	Compteur incorporé dans un champ formaté.
TextColor	long	L+E	Couleur du texte (premier plan).
DataField	string	L	Nom du champ dans l'ensemble de données.
BoundField	object	L	Objet représentant la connexion à l'ensemble de données et donnant accès au contenu du champ.

Champ de texte – autres propriétés (TextField)

Nom	Type	L/E	Propriété
String	string	L+E	Contenu du champ affiché.
MaxTextLen	integer	L+E	Longueur maximale du texte.
DefaultText	string	L+E	Texte par défaut.
MultiLine	boolean	L+E	Indique s'il peut contenir plus d'une ligne

Nom	Type	L/E	Propriété
EchoChar	(integer)	L+E	Caractère affiché pour la saisie du mot de passe.

Champ numérique (NumericField)

Nom	Type	L/E	Propriété
ValueMin	double	L+E	Valeur d'entrée minimale acceptable
ValueMax	double	L+E	Valeur d'entrée maximale acceptable
Value	double	L+(E)	Valeur actuelle (Ne pas utiliser pour les valeurs de l'ensemble de données).
ValueStep	double	L+E	Intervalle correspondant à un clic pour la molette de la souris ou le compteur.
DefaultValue	double	L+E	Valeur par défaut..
DecimalAccuracy	integer	L+E	Nombre de décimales.
ShowThousandsSeparator	boolean	L+E	Afficher le séparateur de paramètres régionaux pour des milliers.

Champ Date (DateField)

Les valeurs de date sont définies par le type de données **long** et sont stockées au format ISO : AAAAMMJJ, par exemple 20190304 pour le 4 mars 2019. Pour utiliser ce type avec **getDate** et **updateDate**, et avec le type **com.sun.star.util. Date**, voir les exemples.

Nom	Type de données	Type de données depuis LO 4.1.1	L/E	Propriété
DateMin	long	com.sun.star.util.Date	L+E	Date d'entrée minimale acceptable.
DateMax	long	com.sun.star.util.Date	L+E	Date d'entrée maximale acceptable.
Date	long	com.sun.star.util.Date	L+(E)	Valeur actuelle (Ne pas utiliser pour les valeurs de l'ensemble de données).
DateFormat	integer		L+E	Format de date spécifique au système d'exploitation: 0 = date courte (simple) 1 = date courte jj.mm.aa (année à 2 chiffres) 2 = date courte jj.mm.aaaa (année à 4 chiffres) 3 = date longue (comprend le jour de la semaine et le nom du mois) D'autres possibilités peuvent être trouvées dans la définition du formulaire ou dans la référence API.
DefaultDate	long	com.sun.star.util.Date	L+E	Valeur par défaut

Nom	Type de données	Type de données depuis LO 4.1.1	L/E	Propriété
DropDown	boolean		L+E	Afficher un calendrier mensuel déroulant.

Champ durée (TimeField)

Les valeurs de temps sont également du type **long**.

Nom	Type de données	Type de données depuis LO 4.1.1	L/E	Propriété
TimeMin	long	com.sun.star.util.Time	L+E	Valeur d'entrée minimale acceptable
TimeMax	long	com.sun.star.util.Time	L+E	Valeur d'entrée maximale acceptable
Time	long	com.sun.star.util.Time	L+(E)	Valeur actuelle (Ne pas utiliser pour les valeurs de l'ensemble de données).
TimeFormat	integer		L+E	Format de l'heure : 0 = court comme hh : mm (heures, minutes, 24 heures) 1 = long hh : mm : ss (même chose avec Secondes, horloge sur 24 heures) 2 = court comme hh : mm (horloge 12 heures avec AM / PM) 3 = long comme hh : mm : ss (horloge 12 heures avec AM / PM) 4 = entrée courte pour une durée 5 = entrée longue pour une durée
DefaultTime	long	com.sun.star.util.Time	L+E	Valeur par défaut.

Champ de devise (CurrencyField)

Un champ monétaire est un champ numérique avec les possibilités supplémentaires suivantes.

Nom	Type	L/E	Propriété
CurrencySymbol	string	L+E	Symbole de la monnaie pour affichage seulement.
PrependCurrencySymbol	boolean	L+E	Le symbole est affiché avant le chiffre.

Champ formaté (FormattedControl)

Un contrôle formaté peut être utilisé à volonté pour les nombres, la devise ou la date/heure. De très nombreuses propriétés déjà décrites s'appliquent ici mais avec des noms différents.

Nom	Type	L/E	Propriété
CurrentValue	variant	L	Valeur actuelle du contenu. Le type de données réel dépend du contenu et du format.
EffectiveValue	variant	L+(E)	
EffectiveMin	double	L+E	Valeur d'entrée minimale acceptable
EffectiveMax	double	L+E	Valeur d'entrée maximale acceptable
EffectiveDefault	variant	L+E	Valeur par défaut
FormatKey	long	L+(E)	Format d'affichage et de saisie. Il n'y a pas de moyen facile de modifier cela à l'aide d'une macro.
EnforceFormat	boolean	L+E	Le format est testé lors de la saisie. Seuls certains caractères et combinaisons sont autorisés.

Zone de liste (ListBox)

L'accès en lecture et en écriture à la valeur située derrière la ligne sélectionnée est quelque peu compliqué mais possible.

Nom	Type	L/E	Propriété
ListSource	array of string	L+E	Source de données : source du contenu de la liste ou nom de l'ensemble de données qui fournit l'entrée visible.
ListSourceType	integer	L+E	Type de source de données : 0 = Liste de valeurs 1 = Tableau 2 = Requête 3 = Ensemble de résultats d'une commande SQL 4 = Résultat d'une commande de base de données 5 = Noms de champs d'une table de base de données
StringItemList	array of string	L	Liste des entrées disponibles pour la sélection.
ItemCount	integer	L	Nombre d'entrées de liste disponibles
ValueItemList	array of string	L	Liste des valeurs à passer du formulaire à la table.
DropDown	boolean	L+E	La liste déroulante.
LineCount	integer	L+E	Nombre total des lignes affichées une fois complètement déroulées.
MultiSelection	boolean	L+E	Sélection multiple prévue.
SelectedItems	array of integer	L+E	Liste des entrées sélectionnées sous forme de liste de positions dans la liste globale des entrées.

Le premier élément sélectionné dans le champ de liste est obtenu comme ceci :

```
oContrôle = oFormulaire.getByName("Nom de la Zone de Liste")
sEntree = oControl.ValueItemList(oControl.SelectedItems(0))
```



Note

Depuis LibreOffice 4.1, la valeur transmise à la base de données peut être déterminée directement.

```
oContrôle = oFormulaire.getByName("Nom de la Zone de Liste")
iD = oContrôle.getCurrentValue()
```

`getCurrentValue()` renvoie la valeur qui sera stockée dans la table de base de données. Dans les Zones de liste, cela dépend du champ auquel elles sont liées (`BoundField`).

Jusqu'à LibreOffice 4.0 inclus, cette fonction renvoyait le contenu affiché, pas la valeur sous-jacente dans la table.

Veuillez noter que l'entrée est un « tableau de chaînes » (array of string), si la requête d'un champ de liste est exécutée pour restreindre une option de sélection :

```
Sub FiltreZoneListe
    Dim stSql(0) As String
    Dim oDoc As Object
    Dim oDrawpage As Object
    Dim oFormulaire As Object
    Dim oChamp As Object
    oDoc = thisComponent
    oDrawpage = oDoc.drawpage
    oFormulaire = oDrawpage.forms.getByName("Formulaire")
    oChamp = oFormulaire.getByName("Zone de liste")
    stSql(0) = "SELECT ""Nom"", ""ID"" FROM ""Nom_Requete_Filtre"" ORDER BY ""Nom"""
    oChamp.ListSource = stSql
    oChamp.refresh
End Sub
```

Zone combinée (ComboBox)

Malgré le fait d'avoir des fonctionnalités similaires à celles des Zones de liste, les propriétés des Combobox sont quelque peu différentes. Voir l'exemple « Boîtes combinées sous forme de Zone de liste avec une option d'entrée » en page 59.

Nom	Type	L/E	Propriété
Autocomplete	boolean	L+E	Remplissage automatique.
StringItemList	array of string	L+E	Liste des entrées disponibles pour utilisation.
ItemCount	integer	L	Nombre d'entrées de liste disponibles.
DropDown	boolean	L+E	Liste déroulante.
LineCount	integer	L+E	Nombre de lignes affichées lorsqu'elle est déroulée.
Text	string	L+E	Texte actuellement affiché.
DefaultText	string	L+E	Entrée par défaut
ListSource	string	L+E	Nom de la source de données qui fournit les entrées de liste.
ListSourceType	integer	L+E	Type de source de données. Mêmes possibilités que pour les Zones de liste (seul le choix de ValueItemList est ignoré).

Cases à cocher (CheckBox) et Boutons Radio (RadioButton)

Les boutons d'options peuvent également être utilisés.

Nom	Type	L/E	Propriété
Label	string	L+E	Titre (label)
State	short	L+E	Statut 0 = non sélectionné 1 = sélectionné 2 = indéfini
MultiLine	boolean	L+E	Sauts de ligne pour le texte long.

Champ de motif (PatternField)

Outre les propriétés du texte simple, les éléments suivants sont intéressants :

Nom	Type	L/E	Propriété
EditMask	string	L+E	Masque de saisie.
LiteralMask	string	L+E	Masque de caractères.
StrictFormat	boolean	L+E	Test de format lors de la saisie.

Contrôle de Table (GridControl)

Nom	Type	L/E	Propriété
Count	long	L	Le nombre de colonnes.
ElementNames	array of string	L	Liste des noms de colonnes.
HasNavigationBar	boolean	L+E	Barre de navigation disponible.
RowHeight	long	L+E	Hauteur de ligne

Étiquette ou Texte fixe (Label)

Nom	Type	L/E	Propriété
Label	string	L+E	Text displayed.
MultiLine	boolean	L+E	Sauts de ligne pour le texte long.

Zones de groupe (GroupBox)

Il n'y a pas de propriétés pour les zones de groupe qui sont traitées normalement à l'aide de macros. C'est le statut des champs d'options individuels qui compte.

Boutons (CommandButton ou ImageButton)

Nom	Type	L/E	Propriété
Label	string	L+E	Titre – Texte de l'étiquette.
State	short	L+E	État par défaut sélectionné pour le basculement.
MultiLine	boolean	L+E	Sauts de ligne pour le texte long.
DefaultButton	boolean	L+E	S'il s'agit d'un bouton par défaut.

Barre de Navigation (NavigationBar)

D'autres propriétés et méthodes associées à la navigation – par exemple les filtres et la modification du pointeur d'enregistrement – sont contrôlées à l'aide du formulaire.

Nom	Type	L/E	Propriété
IconSize	short	L+E	Taille des icônes (Petites, Grandes)
ShowPosition	boolean	L+E	La position peut être saisie et s'affiche.
ShowNavigation	boolean	L+E	Permet la navigation.
ShowRecordActions	boolean	L+E	Permet des actions sur les enregistrements.
ShowFilterSort	boolean	L+E	Permet le tri par filtre.

Méthodes pour les formulaires et les contrôles

Le type de données du paramètre est indiqué par une abréviation :

- c numéro de colonne pour le champ souhaité dans l'ensemble de données, à partir de 1
- n valeur numérique – peut être un entier ou un nombre décimal
- s Chaîne, la longueur maximale dépend de la définition de la table.
- b booléen (logique) – vrai ou faux
- d Valeur de date

Naviguer dans un ensemble de données

Ces méthodes fonctionnent à la fois dans les formulaires et dans l'ensemble de résultats d'une requête.

« Curseur » dans la description signifie le pointeur d'enregistrement.

Nom	Type	Description
Test de la position du curseur		
isBeforeFirst	boolean	Le curseur se trouve avant le premier enregistrement. C'est le cas s'il n'a pas encore été réinitialisé après l'entrée.
isFirst	boolean	Indique si le curseur est sur la première entrée.
isLast	boolean	Indique si le curseur se trouve sur la dernière entrée.
isAfterLast	boolean	Le curseur se trouve après la dernière ligne lorsqu'il est déplacé avec suivant.
getRow	long	Numéro de ligne actuel.
Positionnement du curseur		
Pour les types de données booléens, True signifie que la navigation a réussi.		
beforeFirst	–	Se déplace avant la première ligne.
first	boolean	Passe à la première ligne.
previous	boolean	Remonte d'une ligne.
next	boolean	Avance d'une ligne.
last	boolean	Va au dernier enregistrement.

Nom	Type	Description
afterLast	–	Va après le dernier enregistrement.
absolute(n)	boolean	Accède à la ligne avec le numéro de ligne donné.
relative(n)	boolean	Va en arrière ou en avant du nombre donné : en avant pour les arguments positifs et en arrière pour les arguments négatifs.
Méthodes affectant l'état actuel de l'enregistrement		
refreshRow	–	Relit les valeurs d'origine de la ligne.
rowInserted	boolean	Indique s'il s'agit d'une nouvelle ligne.
rowUpdated	boolean	Indique si la ligne actuelle a été modifiée.
rowDeleted	boolean	Indique si la ligne actuelle a été supprimée.

Modification des lignes (enregistrements) de données

Les méthodes utilisées pour la lecture sont disponibles pour n'importe quel formulaire ou ensemble de données. Les méthodes de modification et de stockage ne peuvent être utilisées que pour les ensembles de données modifiables (généralement des tables, pas des requêtes).

Nom	Type	Description
Méthodes pour ligne entière.		
insertRow	–	Enregistre une nouvelle ligne.
updateRow	–	Confirme la modification de la ligne actuelle.
deleteRow	–	Supprime la ligne actuelle.
cancelRowUpdates	–	Annule les modifications de la ligne actuelle.
moveToInsertRow	–	Déplace le curseur sur une ligne correspondant à un nouvel enregistrement.
moveToCurrentRow	–	Après la saisie d'un nouvel enregistrement, ramène le curseur à sa position précédente.
Lecture des valeurs		
getString(c)	string	Donne le contenu de la colonne sous forme de chaîne de caractères.
getBoolean(c)	boolean	Donne le contenu de la colonne sous forme de valeur booléenne.
getBytes(c)	byte	Donne le contenu de la colonne sous la forme d'un octet unique.
getShort(c)	short	Donne le contenu de la colonne sous forme d'entier.
getInt(c)	integer	
getLong(c)	long	
getFloat(c)	float	Donne le contenu de la colonne sous forme de nombre décimal de précision unique.

Nom	Type	Description
getDouble(c)	double	Donne le contenu de la colonne sous forme de nombre décimal à double précision. Les conversions automatiques effectuées par Basic en font un type approprié pour les champs décimaux et monétaires.
getBytes(c)	array of bytes	Donne le contenu de la colonne sous forme de tableau d'octets uniques.
getDate(c)	Date	Donne le contenu de la colonne sous forme de date.
getTime(c)	Time	Donne le contenu de la colonne sous forme de valeur de temps.
getTimestamp(c)	DateTime	Donne le contenu de la colonne sous forme d'horodatage (date et heure).
Dans Basic lui-même, les valeurs de date et d'heure ont toutes deux le type DATE. Pour accéder aux dates dans les ensembles de données, différents types sont disponibles : com.sun.star.util.Date pour une date, com.sun.star.util.Time pour une heure et com.sun.star.util.DateTime pour un horodatage.		
wasNull	boolean	Indique si la valeur de la colonne la plus récemment lue était NULL.
Enregistrer les valeurs		
updateNull(c)	–	Définit le contenu de la colonne sur NULL.
updateBoolean(c,b)	–	Modifie le contenu de la colonne c en valeur logique b.
updateByte(c,x)	–	Stocke l'octet x dans la colonne c.
updateShort(c,n)	–	Stocke n (Integer, Long ou Float) dans la colonne c.
updateInt(c,n)	–	
updateLong(c,n)	–	
updateFloat(c,n)	–	Stocke le nombre décimal n dans la colonne c.
updateDouble(c,n)	–	
updateString(c,s)	–	Stocke la chaîne s dans la colonne c.
updateBytes(c,x)	–	Stocke le tableau d'octets x dans la colonne c.
updateDate(c,d)	–	Stocke la date d dans la colonne c.
updateTime(c,d)	–	Stocke l'heure d dans la colonne c.
updateTimestamp(c,d)	–	Stocke l'horodatage d dans la colonne c.

Modifier des valeurs individuelles

Cette méthode utilise la propriété **BoundField** d'un contrôle pour lire ou modifier le contenu de la colonne correspondante. Cela correspond presque exactement à la méthode décrite dans la section précédente, sauf que le numéro de colonne n'est pas indiqué.

Nom	Type	Description
Lecture des valeurs		

Nom	Type	Description
getString	string	Donne le contenu du champ sous forme de chaîne de caractères.
getBoolean	boolean	Donne le contenu du champ sous forme de valeur logique.
getByte	byte	Donne le contenu du champ sous la forme d'un octet unique.
getShort	short	Donne le contenu du champ sous forme d'entier.
getInt	integer	
getLong	long	
getFloat	float	Donne le contenu du champ sous la forme d'une valeur décimale simple précision.
getDouble	double	Donne le contenu du champ sous forme de nombre décimal à double précision. Les conversions automatiques effectuées par Basic en font un type approprié pour les champs décimaux et monétaires.
getBytes	array of bytes	Donne le contenu du champ sous forme de tableau d'octets.
getDate	Date	Donne le contenu du champ sous forme de date.
getTime	Time	Donne le contenu du champ sous forme d'heure.
getTimestamp	DateTime	Donne le contenu du champ sous forme d'horodatage.
Dans Basic lui-même, les valeurs de date et d'heure ont toutes deux le type DATE. Pour accéder aux dates dans les ensembles de données, différents types sont disponibles : com.sun.star.util.Date pour une date, com.sun.star.util.Time pour une heure et com.sun.star.util.DateTime pour un horodatage.		
wasNull	boolean	Indique si la valeur de la colonne la plus récemment lue était NULL.
Stockage des valeurs		
updateNull	–	Définit le contenu de la colonne sur NULL.
updateBoolean(b)	–	Définit le contenu de la colonne sur la valeur logique b.
updateByte(x)	–	Stocke l'octet x dans la colonne.
updateShort(n)	–	Stocke l'entier n dans la colonne.
updateInt(n)	–	
updateLong(n)	–	
updateFloat(n)	–	Stocke le nombre décimal n dans la colonne.
updateDouble(n)	–	
updateString(s)	–	Stocke la chaîne de caractères s dans la colonne.
updateBytes(x)	–	Stocke le tableau d'octets x dans la colonne.
updateDate(d)	–	Stocke la date d dans la colonne.

<i>Nom</i>	<i>Type</i>	<i>Description</i>
updateTime(d)	–	Stocke l'heure d dans la colonne.
updateTimestamp(d)	–	Stocke l'horodatage d dans la colonne.

Paramètres des commandes SQL préparées

Les méthodes qui transfèrent la valeur d'une commande SQL pré-préparée (voir « Commandes SQL pré-préparées avec paramètres » en page 15) sont similaires à celles de la section précédente. Le premier paramètre (indiqué par i) est une position numérotée dans la commande SQL.

<i>Nom</i>	<i>Type</i>	<i>Description</i>
setNull(i,n)	–	Définit le contenu de la colonne sur NULL. N est le type de données SQL comme indiqué dans la référence API.
setBoolean(i,b)	–	Place la valeur logique b donnée dans la commande SQL.
setByte(i,x)	–	Place l'octet x donné dans la commande SQL.
setShort(i,n)	–	Place l'entier n donné dans la commande SQL.
setInt(i,n)		
setLong(i,n)		
setFloat(i,n)	–	Place le nombre décimal donné dans la commande SQL.
setDouble(i,n)		
setString(i,s)	–	Place la chaîne de caractères s dans la commande SQL.
setBytes(i,x)	–	Place le tableau d'octets x donné dans la commande SQL.
setDate(i,d)	–	Place la date donnée d dans la commande SQL.
setTime(i,d)	–	Place le temps donné d dans la commande SQL.
setTimestamp(i,d)	–	Place l'horodatage donné d dans la commande SQL.
clearParameters	–	Supprime les valeurs précédentes de tous les paramètres d'une commande SQL.

Améliorer la convivialité

Pour cette première catégorie d'utilisation de macro, nous montrons différentes possibilités pour améliorer la convivialité des formulaires de base.

Mise à jour automatique des formulaires

Souvent, quelque chose est modifié dans un formulaire et cette modification doit apparaître dans un deuxième formulaire sur la même page. L'extrait de code suivant appelle la méthode de rechargement sur le deuxième formulaire, provoquant son actualisation.

Sub MiseAJour

Tout d'abord, la macro est nommée. La désignation par défaut d'une macro est **Sub**. Cela peut être écrit en majuscules ou en minuscules. Sub permet à un sous-programme de s'exécuter sans renvoyer de valeur. Plus bas, en revanche, une fonction est décrite, qui renvoie une valeur.

La macro porte le nom *MiseAJour*. Vous n'avez pas besoin de déclarer des variables car LibreOffice Basic crée automatiquement des variables lorsqu'elles sont utilisées. Malheureusement, si vous avez mal orthographié une variable, LibreOffice Basic crée silencieusement une nouvelle variable sans se plaindre. Utilisez **Option Explicit** Pour empêcher LibreOffice Basic de créer automatiquement des variables, ceci est recommandé par la plupart des programmeurs.

Par conséquent, nous commençons généralement par déclarer des variables. Toutes les variables déclarées ici sont des objets (pas des nombres ou du texte), nous ajoutons donc **As Object** à la fin de la déclaration. Pour nous rappeler plus tard le type des variables, nous faisons précéder leurs noms d'un "o" (pour Object). En principe, cependant, vous pouvez choisir presque tous les noms de variables que vous désirez.

```
Dim oDoc As Object
Dim oDrawpage As Object
Dim oFormulaire As Object
```

Le formulaire se trouve dans le document actuellement actif. Le conteneur, dans lequel tous les formulaires sont stockés, est nommé **drawpage**. Dans le navigateur de formulaires, il s'agit du concept de niveau supérieur duquel tous les formulaires sont subsidiaires.

Dans cet exemple, le formulaire auquel accéder est nommé Affiche, nom visible dans le navigateur de formulaire. Par défaut, le premier formulaire par défaut est appelé Formulaire1.

```
oDoc = thisComponent
oDrawpage = oDoc.drawpage
oFormulaire = oDrawpage.forms.getByName("Affiche")
```

Le formulaire étant désormais accessible et le point par lequel il est accessible est enregistré dans la variable **oFormulaire**, il est maintenant rechargé (rafraîchi) avec la commande **reload()**.

```
oFormulaire.reload()
End Sub
```

Le sous-programme commence par **Sub** et doit donc se terminer par **End Sub**.

Cette macro peut maintenant être sélectionnée pour s'exécuter lorsqu'un autre formulaire est enregistré. Par exemple, sur une caisse enregistreuse (caisse), si le nombre total d'articles vendus et leurs numéros de stock (lus par un lecteur de codes-barres) sont entrés dans un formulaire, un autre formulaire dans la même fenêtre ouverte peut afficher les noms de tous les articles, et le coût total, dès que le formulaire est enregistré.

Filtrage des enregistrements

Le filtre lui-même peut parfaitement fonctionner sous la forme décrite au Chapitre 8, Astuces. La variante ci-dessous remplace le bouton Enregistrer et lit à nouveau les zones de liste, de sorte qu'un filtre choisi dans une zone puisse restreindre les choix disponibles dans l'autre zone².

```
Sub Filtre
```

En premier lieu, les variables sont déclarées.

```
Dim oDoc As Object
Dim oDrawpage As Object
Dim oFormulaire1 As Object
Dim oFormulaire2 As Object
Dim oZoneListe1 As Object
Dim oZoneListe2 As Object
```

Puis, elles sont paramétrées pour accéder à l'ensemble des formulaires. Cet ensemble comprend les deux formulaires « Filtre » et « Affichage ». Les zones de liste sont dans le formulaire « Filtre » et portent les noms « ZoneListe_1 » et « ZoneListe_2 ».

2 Voir la base de données Exemple_Cherche_et_Filtre.odt associée à ce manuel.

```

oDoc = thisComponent
oDrawpage = oDoc.drawpage
oFormulaire1 = oDrawpage.forms.getByName("Filtre")
oFormulaire2 = oDrawpage.forms.getByName("Affiche")
oZoneListe1 = oFormulaire1.getByName("ZoneListe_1")
oZoneListe2 = oFormulaire1.getByName("ZoneListe_2")

```

Ensuite, le contenu des zones de liste est transféré vers le formulaire sous-jacent en utilisant **commit()**. Le transfert est nécessaire, car sinon le changement dans une Zone de liste ne sera pas reconnu lors de l'enregistrement. L'instruction **commit()** n'a besoin d'être appliquée qu'à la Zone de liste qui vient d'être accédée. Après cela, l'enregistrement est réalisé à l'aide de **updateRow()**.

```

oZoneListe1.commit()
oZoneListe2.commit()
oFormulaire1.updateRow()

```

En principe, notre table de filtres ne contient qu'un seul enregistrement, qui est écrit une fois au début. Cet enregistrement est donc écrasé en permanence à l'aide d'une commande de mise à jour.

```

oZoneListe1.refresh()
oZoneListe2.refresh()
oFormulaire2.reload()

```

End Sub

Les listes de sélection sont destinées à s'influencer mutuellement. Par exemple, si une zone de liste est utilisée pour limiter les médias affichés aux CD, l'autre zone de liste ne doit pas inclure tous les auteurs de livres dans sa liste d'auteurs. Une sélection dans la deuxième zone de liste aboutirait alors trop souvent à un filtre vide. C'est pourquoi les zones de liste doivent être relues. À proprement parler, la commande **refresh()** ne doit être exécutée que sur la zone de liste qui n'a pas été accédée.

Après cela, Formulaire2, qui devrait afficher le contenu filtré, est à nouveau lu (reload).

Les zones de liste qui doivent être influencées à l'aide de cette méthode peuvent recevoir du contenu à l'aide de diverses requêtes.

La variante la plus simple consiste à faire en sorte que la zone de liste tire son contenu des résultats du filtre. Ensuite, un seul filtre détermine quel contenu de données sera filtré davantage.

```

SELECT "Champ_1" || ' - ' || "Nombre" AS "Affiche", "Champ_1"
FROM
  (SELECT COUNT("ID") AS "Nombre", "Champ_1"
   FROM "TableRecherche" GROUP BY "Champ_1")
ORDER BY "Champ_1"

```

Le contenu du champ et le nombre de résultats s'affichent. Pour obtenir le nombre de résultats, une sous-requête est utilisée. Ceci est nécessaire car sinon, seul le nombre de résultats, sans plus d'informations sur le champ, sera affiché dans la zone de liste.

La macro crée des zones de liste assez rapidement par cette action, elles ne contiennent qu'une seule valeur. Si une zone de liste n'est pas NULL, elle est prise en compte par le filtrage. Après l'activation de la deuxième zone de liste, seuls les champs vides et une valeur affichée sont disponibles pour les deux zones de liste. Cela peut sembler pratique pour une recherche limitée. Mais que se passe-t-il si un catalogue de bibliothèque montre clairement la classification d'un élément, mais ne montre pas uniquement s'il s'agit d'un livre, d'un CD ou d'un DVD ? Si la classification est choisie en premier et que la seconde zone de liste est alors mise à « CD », elle doit être remise à NULL afin d'effectuer une recherche ultérieure qui inclut des livres. Il serait plus pratique que la deuxième zone de liste affiche directement les différents types de médias disponibles, avec le nombre de succès correspondant.

Pour atteindre cet objectif, la requête suivante est construite, qui n'est plus alimentée directement à partir des résultats du filtre. Le nombre de succès doit être obtenu d'une manière différente.

```
SELECT
IFNULL("Champ_1" || ' ' || "Compte", 'vide - ' || "Compte")
AS "Affiche", "Champ_1"
FROM
  (SELECT COUNT("ID") AS "Compte", "Champ_1" FROM "Table"
   WHERE "ID" IN
     (SELECT "Table"."ID" FROM "Filtre", "Table"
      WHERE "Table"."Champ_2" = IFNULL("Filtre"."Filtre_2",
        "Table"."Champ_2"))
   GROUP BY "Champ_1")
ORDER BY "Champ_1"
```

Cette requête très complexe peut être décomposée. En pratique, il est courant d'utiliser une **VUE** pour la sous-requête. La zone de liste reçoit son contenu à partir d'une requête relative à cette **VUE**.

La requête en détail : La requête présente deux colonnes. La première colonne contient la vue, vue par une personne qui a ouvert le formulaire. Cette vue montre le contenu du champ et, séparés par un tiret, les résultats pour ce contenu de champ. La deuxième colonne transfère son contenu vers la table sous-jacente du formulaire. Ici, nous n'avons que le contenu du champ. Les zones de liste tirent ainsi leur contenu de la requête, qui est présentée comme résultat du filtre dans le formulaire. Seuls ces champs sont disponibles pour un filtrage supplémentaire.

La table à partir de laquelle ces informations sont tirées est en fait une requête. Dans cette requête, les champs de clé primaire sont comptés (`SELECT COUNT("ID") AS "Nombre"`). Ceci est ensuite regroupé par le terme de recherche dans le champ (`GROUP BY "Champ_1"`). Cette requête présente le terme dans le champ lui-même en tant que deuxième colonne. Cette requête à son tour est basée sur une autre sous-requête :

```
SELECT "Table"."ID" FROM "Filtre", "Table"
WHERE
  "Table"."Champ_2" = IFNULL("Filtre"."Filtre_2", "Table"."Champ_2")
```

Cette sous-requête traite de l'autre champ à filtrer. En principe, cet autre champ doit également correspondre à la clé primaire. S'il existe d'autres filtres, cette requête peut être étendue :

```
SELECT "Table"."ID" FROM "Filtre", "Table"
WHERE
  "Table"."Champ_2" = IFNULL("Filtre"."Filtre_2", "Table"."Champ_2")
AND
  "Table"."Champ_3" = IFNULL("Filtre"."Filtre_3", "Table"."Champ_3")
```

Cela permet à tous les champs supplémentaires qui doivent être filtrés de contrôler ce qui apparaît finalement dans la zone de liste du premier champ, "Champ_1".

Enfin, toute la requête est triée par le champ sous-jacent.

À quoi ressemble réellement la requête finale sous-jacente au formulaire affiché, peut être vu dans le Chapitre 8, « Trucs et Astuces ».

La macro suivante peut contrôler, via une zone de liste, quelles zones de liste doivent être enregistrées et lesquelles doivent être lues à nouveau. Le sous-programme suivant suppose que la propriété Complément d'information pour chaque zone de liste contient une liste séparée par des virgules de tous les noms de zone de liste sans espaces. Le premier nom de la liste doit être le nom de cette zone de liste elle-même.

```
Sub Filtre_info_Additionnelle(oEvent As Object)
  Dim oDoc As Object
  Dim oDrawpage As Object
  Dim oFormulaire1 As Object
```

```

Dim oFormulaire2 As Object
Dim sTag As String
Dim arListe()
sTag = oEvent.Source.Model.Tag

```

Un tableau (une collection de données accessible via un numéro d'index) est établi et rempli avec les noms de champs des zones de liste. Le premier nom de la liste est le nom de la zone de liste liée à l'événement.

```

arListe() = Split(sTag, ",")
oDoc = thisComponent
oDrawpage = oDoc.drawpage
oFormulaire1 = oDrawpage.forms.getByName("Filtre")
oFormulaire2 = oDrawpage.forms.getByName("Affiche")

```

Le tableau est parcouru de sa borne inférieure ('**Lbound()**') à sa borne supérieure ('**Ubound()**') en une seule boucle. Toutes les valeurs séparées par des virgules dans les informations supplémentaires sont maintenant transférées successivement.

```

For i = LBound(arListe()) To UBound(arListe())
    If i = 0 Then

```

La zone de liste qui a déclenché la macro doit être enregistrée. Elle se trouve dans la variable **arListe(0)**. Tout d'abord, les informations de la zone de liste sont transmises à la table sous-jacente, puis l'enregistrement est enregistré.

```

oFormulaire1.getByName(arListe(i)).commit()
oFormulaire1.updateRow()
Else

```

Les autres zones de liste doivent être actualisées, car elles contiennent désormais des valeurs différentes selon le choix effectué dans la première zone de liste.

```

oFormulaire1.getByName(arListe(i)).refresh()
End If
Next
oFormulaire2.reload()
End Sub

```

Les requêtes pour cette macro plus utilisable sont naturellement les mêmes que celles déjà présentées dans la section précédente.

Filtrage des données via le filtre de formulaires

Comme alternative à cette procédure, il est également possible de modifier directement la fonction de filtrage du formulaire.

```

SUB DefinirFiltrage
    DIM oDoc AS OBJECT
    DIM oForm AS OBJECT
    DIM oChamp AS OBJECT
    DIM stFiltre As String
    oForm = thisComponent.Drawpage.Forms.getByName("Formulaire")
    oChamp = oForm.getByName("Filtre")
    stFiltre = oChamp.Text
    oForm.filter = " UPPER('Nom') LIKE '%" + UCase(stFiltre) + "%'"
    oForm.ApplyFilter = TRUE
    oForm.reload()
End Sub

```

Le champ **Filtre** est recherché dans le formulaire Formulaire et le contenu est lu dans la variable **stFiltre**. Le filtre est réglé en conséquence (**oForm.filter**). Le filtre est activé et le formulaire est rechargé.

```

SUB SupprimeFiltre
    DIM oForm AS OBJECT

```

```

        oForm = thisComponent.Drawpage.Forms.getByName("Formulaire")
        oForm.ApplyFilter = False
        oForm.reload()
    END SUB

```

Bien entendu, le filtre peut également être supprimé via la barre de navigation.

Avec cette fonction de filtrage, un formulaire peut également être lancé directement avec un filtre, par exemple « Dupont » pour un seul enregistrement.

Faire défiler les enregistrements avec une barre de défilement

Un contrôle de barre de défilement ne peut être utilisé que par le biais de macros. L'exemple_ suivant montre comment faire défiler des enregistrements de données avec une telle barre de défilement. Elle peut alors être utilisée à la place de la barre de navigation³.

```

GLOBAL loPosistion AS LONG

```

La position de l'enregistrement en cours est stockée sous la forme d'une variable globale, de sorte qu'elle peut être lue dans toutes les procédures et modifiée dans toutes les procédures.

```

SUB LigneMax(oEvent AS OBJECT)
    ' Déclenché les événements du formulaire
    ' "Lors du chargement" et "Après l'action d'enregistrement".
    DIM oFormulaire AS OBJECT
    DIM oDefileChamp AS OBJECT
    DIM loMax AS LONG
    oFormulaire = oEvent.Source
    oDefileChamp = oFormulaire.getByName("Barre_defilement")
    loPosition = oFormulaire.getRow
    oFormulaire.last
    loMax = oFormulaire.getRow
    oFormulaire.absolute(loPos)
    oDefileChamp.ScrollValueMax = loMax
    oDefileChamp.ScrollValue = loPos
END SUB

```

Le nombre total d'enregistrements de données ne peut pas être déterminé à l'aide de la fonction RowCount du formulaire, car cette fonction ne compte que les enregistrements de données qui ont déjà été chargés dans la mémoire cache. Par conséquent, le numéro de la ligne actuelle du formulaire est lu en premier, puis le saut à la fin des données à lire est effectué et le numéro de la ligne est déterminé comme valeur maximale. Ensuite, il est nécessaire de revenir à la ligne d'origine avec **oFormulaire.absolute()**.

La barre de défilement est informée de la valeur maximale déterminée en tant que ScrollValueMax et de la position actuelle en tant que loPosition. Si cette valeur n'est pas affectée, la position dans la barre de défilement ne correspond pas nécessairement à l'ensemble des données actuelles.

```

SUB Navigation(oEvent AS OBJECT)
    ' Déclenché par l'événement du formulaire "Après le changement d'enregistrement".
    ' Synchronise la position du défilement avec la position de l'ensemble des données
    DIM oFormulaire AS OBJECT
    DIM oDefileChamp AS OBJECT
    oFormulaire = oEvent.Source
    oDefileChamp = oFormulaire.getByName("Barre_defilement")
    loPos = oFormulaire.getRow
    IF loPos = 0 THEN
        loPos = oFormulaire.RowCount + 1
    END IF
    oDefileChamp.ScrollValue = loPos
END SUB

```

Lors de la navigation dans les enregistrements de données, l'affichage de la barre de défilement et l'affichage de la barre de navigation doivent toujours correspondre. C'est pourquoi le numéro de la

³ Voir la base de données Exemple_Defiler_Enregistrements.odt associée à ce manuel.

ligne en cours est toujours déterminé après le changement d'enregistrement de données. Pour le numéro de la ligne en cours, un "0" est affiché lorsque le curseur se déplace au-delà de la dernière ligne pour ajouter un nouvel enregistrement. Dans ce cas, cependant, la barre de défilement ne doit pas revenir à la position de départ, mais doit être positionnée à 1 au-dessus de la valeur maximale précédente, comme la barre de navigation.

```
SUB Defilement(oEvent AS OBJECT)
    ' Déclenché par l'événement de la "Barre_defilement" - "A l'ajustement"
    DIM oFormulaire AS OBJECT
    DIM oDefiler AS OBJECT
    oDefiler = oEvent.Source
    oFormulaire = oDefiler.Model.Parent
    loPos = oDefiler.GetValue()
    oFormulaire.absolute(loPos)
END SUB
```

Dans cette procédure, une nouvelle valeur pour la ligne du formulaire est déterminée à partir de la barre de défilement. La valeur est lue dans la barre de défilement avec `GetValue()` et affectée au formulaire avec `oFormulaire.absolute(loPos)`.

Adapter les données des champs de textes aux conventions du langage SQL

Lorsque les données sont stockées dans une commande SQL, les guillemets simples (') dans des noms tels que "O'Connor" peuvent poser des problèmes. Cela est dû au fait que les guillemets simples (' et ') sont utilisés pour encadrer le texte à saisir dans les enregistrements. Dans de tels cas, nous avons besoin d'une fonction intermédiaire pour préparer les données de manière appropriée⁴.

```
Function String_to_SQL(st As String)
    If InStr(st, "'") Then
        st = Join(Split(st, "'"), "''")
    End If
    String_to_SQL = st
End Function
```

Notez que c'est une fonction, pas une procédure. Une fonction prend une valeur comme argument, puis renvoie une valeur.

Le texte à transférer est d'abord scanné pour voir s'il contient une apostrophe. Si tel est le cas, le texte est divisé à ce stade – l'apostrophe elle-même est le délimiteur de la division – et joint à nouveau avec deux apostrophes. Cela masque le code SQL. La fonction donne son résultat via l'appel suivant :

```
stNouveauTexte = String_to_SQL(stAncienTexte)
```

Cela signifie simplement que la variable `stAncienTexte` est retravaillée et le résultat stocké dans `stNouveauTexte`. Les deux variables n'ont pas besoin d'avoir des noms différents. L'appel peut être fait avec :

```
stTexte = String_to_SQL(stTexte)
```

Cette fonction est utilisée à plusieurs reprises dans les macros suivantes afin que les apostrophes puissent également être stockées à l'aide de commandes SQL.

Enregistrer les commandes SQL et les exécuter à la demande

Seules les requêtes peuvent être lancées via le module d'interrogation. Le code SQL qui a également un effet modificateur sur les données ne peut pas y être exécuté, sauf si le pilote de la base de données l'autorise, comme c'est le cas pour le pilote direct de MySQL. L'éditeur disponible via **Outils** → **SQL** permet l'exécution de code, mais n'offre pas de possibilité de

4 Voir la base de données `Exemple_ZoneCombinée_ListeChamps.odt` associée à ce manuel.

stockage. Cela signifie que les commandes récurrentes doivent être stockées séparément quelque part et peuvent ensuite être copiées dans l'éditeur via le presse-papiers. La macro suivante fournit un remède⁵.

Le code destiné à une opération telle que UPDATE, DELETE ou INSERT est stocké dans une table dont le premier champ contient la clé primaire **ID** et dont le second champ contient le code SQL. Un formulaire présente, dans un contrôle de table, l'ensemble des requêtes enregistrées. Après sélection de l'enregistrement souhaité la requête SQL qu'il contient est exécutée via un bouton.

```
Sub ChangeDonnees(oEvent As Object)
    Dim oConnexion As Object
    Dim oForm As Object
    Dim stSql As String
    Dim oSql_Instruction As Object
    Dim inRetourBouton As Integer
    oForm = oEvent.Source.Model.Parent
    oConnexion = oForm.activeConnection()
    stSQL = oForm.getString(2) 'c'est le 2ème champ qui contient le code SQL.
    inRetourBouton = MsgBox("Exécuter le code SQL suivant ?" & CHR(13) & _
        stSQL & CHR(13), 20, "Exécuter une requête SQL")
    If inRetourBouton = 6 Then
        oSQL_Instruction = oConnexion.createStatement()
        oSQL_Instruction.executeUpdate(stSql)
    End If
End Sub
```

Pour des raisons de sécurité, ce code SQL est d'abord affiché à nouveau dans une boîte de dialogue. Si vous ne confirmez pas par « Oui », le code n'est pas exécuté. Cette confirmation est déterminée par la valeur de retour de la boîte de dialogue (6 correspond à Oui ou la constante IDYES). Ensuite, la connexion pour la déclaration est d'abord établie, puis la déclaration est exécutée.

Pré-calculs et calculs dans un formulaire

Les remarques suivantes s'appliquent à tous les formulaires utilisés dans Base, pas seulement à celui figurant dans la base d'exemple sur laquelle s'appuie ce chapitre⁶

Les valeurs calculées, dans un formulaire, ne sont pas stockées dans la base de données. Elles ne sont utiles que pour leur visualisation à l'intérieur du formulaire, ou lors de l'impression de ce dernier.

Le ou les calculs peuvent être réalisés lors :

- de l'ouverture du formulaire lorsqu'il utilise des données provenant d'une requête SQL réalisant ce ou ces calculs. Il serait possible de produire le même effet à l'aide d'une macro, déclenchée à l'ouverture du formulaire, mais cela est beaucoup plus complexe à mettre en œuvre ;
- de la saisie ou de la modification d'un champ numérique lorsqu'une procédure basic (macro) liée à ce champ est exécutée.

Les valeurs calculées peuvent être lues :

- immédiatement après l'ouverture du formulaire lorsqu'il se réfère à une requête SQL (cf. ci-dessus) ;
- dès qu'une valeur est modifiée dans un champ "numérique" grâce à une macro Basic.

5 Voir la base de données Exemple_InsertUpdateDelete_SQL.odt associée à ce manuel.

6 Voir la base de données Exemple_Calcul_Direct_Formulaire.odt associée à ce manuel.

Pré-calculs grâce à une requête SQL

Dans la base exemple il existe une requête SQL dénommée "Requête".

Elle peut être créée de deux manières :

- en mode ébauche, où après ajout de la table "Table", on définit les champs calculés et les formules de calcul (Voir figure ci-dessous).

Table ID Produit Prix_Ht Quantite Taxe					
Champ	Table.*	"Prix_Ht" * (1 + "Taxe" / 100)	"Quantite" * "Prix_Ht"	"Quantite" * "Prix_Ht" * (1 + "Taxe" / 100)	("Quantite" * "Prix_Ht" * (1 + "Taxe" / 100)) - ("Quantite" * "Prix_Ht")
Alias		Prix_TTC	Total_Ht	Total_TTC	Total_Taxe
Table	Table				
Tri					
Visible	☒	☒	☒	☒	☒

- ou directement, de manière plus simple, en mode « Créer une requête en mode SQL ». La requête listée ci-après montre comment cela est possible.

```
SELECT "Table".*,
       "Prix_Ht" * ( 1 + "Taxe" / 100 ) AS "Prix_TTC",
       "Quantite" * "Prix_Ht" AS "Total_Ht",
       "Quantite" * "Prix_Ht" * ( 1 + "Taxe" / 100 ) AS "Total_TTC",
       ( "Quantite" * "Prix_Ht" * ( 1 + "Taxe" / 100 ) ) - ( "Quantite" *
       "Prix_Ht" ) AS "Total_Taxe"
FROM "Table"
```

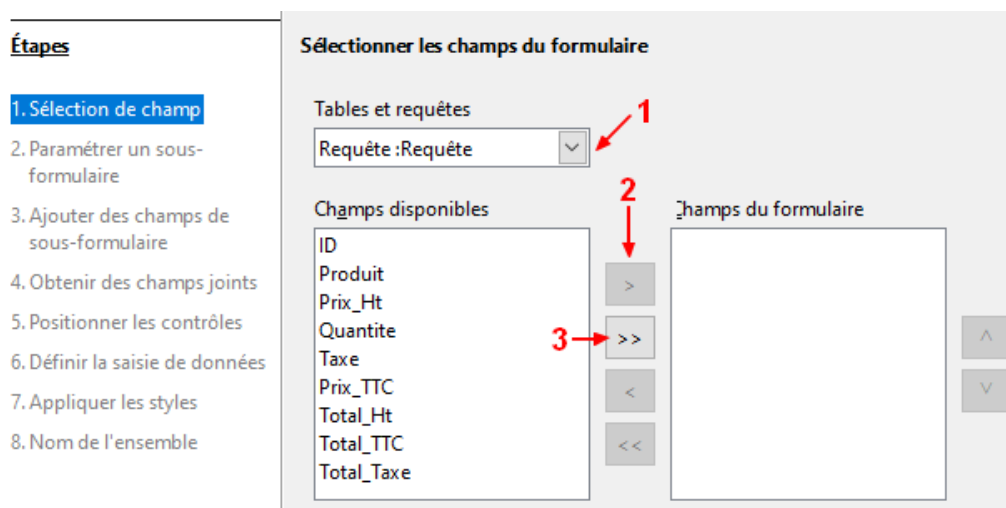
<pre>SELECT "Table".*, FROM "Table"</pre>	On sélectionne tous les champs de la table "Table"
<pre>"Prix_Ht" * (1 + "Taxe" / 100) AS "Prix_TTC",</pre>	On calcule la valeur du prix TTC de l'article. Valeur qui est affecté à "Prix_TTC"
<pre>"Quantite" * "Prix_Ht" AS "Total_Ht", "Quantite" * "Prix_Ht" * (1 + "Taxe" / 100) AS "Total_TTC", ("Quantite" * "Prix_Ht" * (1 + "Taxe" / 100)) - ("Quantite" * "Prix_Ht") AS "Total_Taxe"</pre>	On procède de même pour les calculs des : "Total_Ht", total hors taxe de l'article, "Total_TTC", total toute taxe comprise, "Total_Taxe", cout total de la taxe appliquée.

Lors de l'exécution de cette requête, on obtient le tableau suivant.

	ID	Produit	Prix_Ht	Quantite	Taxe	Prix_TTC	Total_Ht	Total_TTC	Total_Taxe
▶	1	Papier	3,24	2	20	3,89	6,48	7,78	1,3
	2	Equerre	0,79	3	19	0,94	2,37	2,82	0,45
	3	Dossier	0,7	24	20	0,84	16,8	20,16	3,36
	5	Bloc	2,34	6	5,5	2,47	14,04	14,81	0,77

Ensuite un formulaire est créé. Le plus simple est d'utiliser "l'assistant pour créer un formulaire", qui permet de sélectionner la source de données (1), puis de transférer les champs disponibles, vers champs du formulaire à l'aide de divers boutons > , >> , < , << (2). Ici, on transfère tous les

champs (3). Pour les étapes suivantes, se reporter au chapitre "Créer des formulaires" du guide Base "GB6404-Formulaires.odt").



On obtient, après mise en page, etc. un formulaire tel que celui-ci :

ID	Produit		
Prix hors Taxe		Prix TTC	
Quantité		Total TTC	
Taxe en %		Total Taxe	

Afin qu'il affiche les valeurs résultantes de la requête il est nécessaire de modifier ses propriétés par le menu **Formulaire > Propriétés du Formulaire**, ou l'icône  dans la barre d'outils ébauche de formulaire. Dans l'onglet « **Données** » on sélectionne : pour « **Type de contenu** » Requête, pour « **Contenu** » Requête, etc. comme sur la figure ci-dessous.

Général	Données	Événements
Type de contenu.....	Requête	▼
Contenu.....	Requête	▼ ...
Analyser l'instruction SQL.....	Oui	▼
Filtrer.....		▼ ...
Trier.....		▼ ...
Autoriser les ajouts.....	Oui	▼
Autoriser les modifications.....	Oui	▼
Autoriser les suppressions.....	Oui	▼
Ajouter uniquement des données.....	Non	▼
Barre de navigation.....	Oui	▼
Cycle.....	Par défaut	▼

Ainsi, lors de l'ouverture du formulaire, positionné sur le premier enregistrement de la table, on pourra lire les champs calculés par la requête SQL.

Calculs directs dans un formulaire

Dans le même formulaire, il est possible de réaliser des calculs à l'aide de procédures Basic (Macros). Les deux macros suivantes montrent comment réaliser cela. Elles sont liées à la sortie de certains champs(événement perte de focus). La compréhension des lignes de code est facilitée par les commentaires, mais sera développée par la suite.

Option Explicit

```

' -----Sub Calcul_Prix_TTC-----
' Lancé par : évènement perte de focus des champs : Prix_Ht & Taxe
' Calcule le prix Toutes Taxes Comprises
' Champs pour le calcul : Prix_Ht (hors taxe) & Taxe
' Champ modifié : Prix_TTC (Toutes Taxes Comprises)
' Lance ensuite le calcul des totaux
' -----
Sub Calcul_Prix_TTC(oEvent As Object)
    ' Déclaration des variables
    Dim oChamp As Object
    Dim oFormulaire As Object
    Dim oChamp1 As Object
    Dim oChamp2 As Object
    Dim oChamp3 As Object
    Dim dPrix_Ht As Double ' d : décimal
    Dim dTaxe As Double
    Dim dPrix_TTC As Double
    ' Procédure principale
    oChamp = oEvent.Source.Model
    oFormulaire = oChamp.Parent
    ' Récupère dans dPrix_Ht le montant du prix hors taxe du produit
    oChamp1 = oFormulaire.GetByName("Prix_Ht")
    dPrix_Ht = oChamp1.GetCurrentValue
    ' Récupère dans dTaxe le montant de la Taxe (en france TVA)
    oChamp2 = oFormulaire.GetByName("Taxe")
    dTaxe = oChamp2.GetCurrentValue
    ' Calcul du prix_TTC et affichage dans le champ Adhoc du formulaire
    dPrix_TTC = dPrix_Ht * (1 + dTaxe / 100)
    oChamp3 = oFormulaire.GetByName("Prix_TTC")
    oChamp3.BoundField.UpdateDouble(dPrix_TTC)
    If Not IsEmpty(oFormulaire.GetByName("Quantite").GetCurrentValue()) Then
        Calcul_Totaux(oEvent)
    End If
End Sub

```

```

End If
End Sub

'-----Sub Calcul_Totaux-----
' Lancé par : - Calcul_Prix_TTC ;
'           - événement perte de focus du champ : Quantite
' Calcule le Total_TTC & le Total_Taxe
' Champs pour le calcul : Prix_TTC, Quantite, Total_TTC, Prix_Ht
' Champs modifiés : Total_TTC & Total_Taxe
'-----
Sub Calcul_Totaux(oEvent As Object)
    Dim oChamp As Object
    Dim oFormulaire As Object
    Dim oChamp1 As Object
    Dim oChamp2 As Object
    Dim oChamp3 As Object
    Dim oChamp4 As Object
    Dim oChamp5 As Object
    Dim dPrix_TTC As Double
    Dim iQuantite As Integer
    Dim dPrix_Ht As Double
    Dim dTotal_TTC As Double
    Dim dTotal_Taxe As Double
    oChamp = oEvent.Source.Model
    oFormulaire = oChamp.Parent
    oChamp1 = oFormulaire.GetByName("Prix_TTC")
    dPrix_TTC = oChamp1.GetCurrentValue
    oChamp2 = oFormulaire.GetByName("Quantite")
    iQuantite = oChamp2.GetCurrentValue
    oChamp3 = oFormulaire.GetByName("Prix_Ht")
    dPrix_Ht = oChamp3.GetCurrentValue
    oChamp4 = oFormulaire.GetByName("Total_TTC")
    oChamp4.BoundField.UpdateDouble(dPrix_TTC * iQuantite)
    oChamp5 = oFormulaire.GetByName("Total_Taxe")
    oChamp5.BoundField.UpdateDouble((dPrix_TTC * iQuantite) - (dPrix_Ht * iQuantite))
End Sub

```

Pour que ces macros fonctionnent automatiquement, il faut paramétrer les propriétés des champs du formulaire "Prix_Ht", "Quantite", et "Taxe". Pour cela, en mode édition, on affectera une des macros à l'évènement "Perte de Focus" de ces champs (Voir "Évènement dans un formulaire" du présent guide).

La macro Calcul_Prix_TTC sera affectée aux pertes de focus des champs : Prix_Ht et Taxe. La seconde, Calcul_Totaux le sera, pour le même type d'évènement, pour le champ "Quantite".

Ainsi, si une valeur est saisie, modifiée, supprimée dans les champs précités, la macro liée à la sortie (Perte de Focus) de ces champs est exécutée et modifie les valeurs des champs calculés (Prix_TTC, Total_TTC, et Total_Taxe).

Par exemple, le champ Prix_TTC est mis à jour suite à l'exécution de l'instruction :

```
oChamp3.BoundField.UpdateDouble(dPrix_TTC)
```

UpdateDouble est une méthode de **BoundField** (littéralement champ lié), elle-même propriété de l'objet oChamp3.

On peut traduire cette instruction, en langage courant, par :

Mets à jour au format double (**UpdateDouble**), le champ lié (**BoundField**) à l'objet oChamp3, c'est-à-dire le champ "Prix_TTC", avec la valeur calculée dPrix_TTC.

La liaison entre oChamp3 et "Prix_TTC" étant réalisée par l'instruction :

```
oChamp3 = oFormulaire.GetByName("Prix_TTC")
```

Si le champ "Quantite" contient une valeur (**Not IsEmpty** - n'est pas vide)

```
If Not IsEmpty(oFormulaire.getByName("Quantite").getCurrentValue()) Then
    Calcul_Totaux(oEvent)
End If
```

des calculs supplémentaires sont effectués pour les champs qui lui sont liés, à savoir les totaux. Cela est fait en lançant la seconde macro "Calcul_Totaux".

Cette seconde procédure n'appelle aucun commentaire, car les instructions qui la composent sont similaires à celles de la première.

Enfin, il est possible de réaliser la même chose avec un formulaire ne comportant qu'un champ unique de Contrôle de Table. Voir pour cela dans la base exemple, le formulaire Form_Controler_Table.

Concernant l'initialisation des deux variables object, au début des macros,

```
oChamp = oEvent.Source.Model
oFormulaire = oChamp.Parent
```

se référer aux deux derniers paragraphes du sous-chapitre "Accéder aux formulaires" de ce guide.

Fournir la version actuelle de LibreOffice

LibreOffice version 4.1 a apporté des modifications aux champs de liste et aux valeurs de date qui rendent nécessaire la détermination de la version actuelle lors de l'exécution de macros dans ces zones. Le code suivant sert cet objectif :

```
Function OfficeVersion()
    Dim aSettings, aConfigProvider
    Dim aParams2(0) As New com.sun.star.beans.PropertyValue
    Dim sProvider$, sAccess$
    sProvider = "com.sun.star.configuration.ConfigurationProvider"
    sAccess = "com.sun.star.configuration.ConfigurationAccess"
    aConfigProvider = createUnoService(sProvider)
    aParams2(0).Name = "nodepath"
    aParams2(0).Value = "/org.openoffice.Setup/Product"
    aSettings = aConfigProvider.createInstanceWithArguments(sAccess, aParams2())
    OfficeVersion() = Array(aSettings.oName, aSettings.oSetupVersionAboutBox)
End Function
```

Cette fonction renvoie un tableau dans lequel le premier élément est "LibreOffice" et le second est le numéro de version complet, par exemple 6.4.6.2.



Attention

On ne peut pas récupérer directement la valeur de la fonction. Il faut d'abord l'affecter à une variable tableau.

```
Dim aVersion()
aVersion = OfficeVersion() ' on peut ensuite utiliser les deux valeurs
```

Renvoyer la valeur des zones de liste

Depuis LibreOffice 4.1, la valeur renvoyée par une zone de liste à la base de données est stockée dans *CurrentValue*. Ce n'était pas le cas dans les versions précédentes, ni dans OpenOffice ou Apache OpenOffice. La fonction suivante fera le calcul. La version de LibreOffice doit être vérifiée pour voir si elle est postérieure à LibreOffice 4.0.

```
Function ID_Determination(oChamp As Object) As Integer
    a() = OfficeVersion()
    If a(0) = "LibreOffice" And (LEFT(a(1),1) = 4 And RIGHT(LEFT(a(1),3),1) > 0) Or
        LEFT(a(1),1) > 4 Then
```

```

        stContenu = oChamp.currentValue
    Else

```

Avant LibreOffice 4.1, la valeur transmise était lue dans la liste de valeurs de la zone de liste. L'enregistrement visiblement choisi est SelectedItems (0). "0" car plusieurs valeurs supplémentaires peuvent être sélectionnées dans une zone de liste.

```

        stContenu = oChamp.ValueItemList(oChamp.SelectedItems(0))
    End If
    If IsEmpty(stContenu) Then

```

- 1 est une valeur qui n'est pas utilisée comme valeur automatique et qui n'existera donc pas dans la plupart des tables en tant que clé étrangère.

```

        ID_Determination = -1
    Else
        ID_Determination = Cint(stContenu)

```

Converti en entier

```

    End If
End Function

```

La fonction transmet la valeur sous forme d'entier. La plupart des clés primaires incrémentent automatiquement des entiers. Lorsqu'une clé étrangère ne satisfait pas ce critère, la valeur de retour doit être ajustée au type approprié.

La valeur affichée d'un champ de liste peut être autrement déterminée à l'aide de la propriété d'affichage du champ.

```

Sub ListeChampAffiche
    Dim oDoc As Object
    Dim oFormulaire As Object
    Dim oZoneListe As Object
    Dim oController As Object
    Dim oVue As Object
    oDoc = thisComponent
    oFormulaire = oDoc.Drawpage.Forms(0)
    oZoneListe = oFormulaire.getByName("ZoneListe")
    oController = oDoc.getCurrentController()
    oVue = oController.getControl(oZoneListe)
    print "Contenu affiché : " & oVue.SelectedItem
End Sub

```

Le contrôleur est utilisé pour accéder à la vue du formulaire. Cela détermine ce qui apparaît dans l'interface visuelle. La valeur sélectionnée est **SelectedItem**.

Limiter les listes de sélection en saisissant les premières lettres

Il peut parfois arriver que le contenu des listes de sélection devienne interminable et trop volumineux à gérer. Dans de tels cas, il est nécessaire de positionner la liste déroulante sur le premier champ correspondant à des critères saisis par l'utilisateur, ou de limiter le contenu d'une zone de liste avec ces mêmes critères⁷. Ceci est réalisé grâce aux procédures décrites par la suite.

Une première solution est utilisée dans le formulaire « Filtre_Macro_Tableau_Direct », qui comporte deux contrôles, une zone de liste (liste déroulante) et un contrôle de table.

Pour la première, dans ses **Propriétés**, onglet **Données**, une requête SQL est nécessaire.

```

SELECT DISTINCT "Auteur" FROM "TableCherche" ORDER BY "Auteur" ASC

```

L'ensemble du paramétrage de la zone de liste figure sur l'illustration suivante :

⁷ Pour l'utilisation de cette macro, voir l'exemple Exemple_Cherche_et_Filtre, associée à ce manuel.

Général	Données	Événements
Champ de données.....		▼
Saisie requise.....	Non	▼
Type du contenu de liste....	Sql	▼
Contenu de liste.....	"SELECT DISTINCT "Auteur" FROM "TableCherche" ORDER BY "Auteur" ASC"	▼ ...
Champ lié.....	0	▼

Champ Lié est défini sur « 0 », car il n'y a qu'une seule colonne dans la requête (Auteur).

La macro (Standard.Module1.Filtre_direct (document, Basic)) ci-après est liée à **Événements – Modifié(es)**. La chaîne suivante **Auteur = ''** est lié aux **Propriétés** du formulaire, onglet **Données**, rubrique **Filtre**.

```
Sub Filtre_direct(oEvent AS Object)
' Le filtre du formulaire est défini directement, il peut être
' activé et désactivé dans la barre de navigation
Dim oFormulaire AS Object
Dim oChamp AS Object
oChamp = oEvent.Source.Model
oFormulaire = oChamp.Parent
stListeValeurs = oChamp.GetCurrentValue()
If stListeValeurs = "" Then
    oFormulaire.Filter = ""
Else
    oFormulaire.Filter = "Auteur = '" + stListeValeurs + "'"
End If
oFormulaire.reload()
End Sub
```

Après l'initialisation des variables oChamp et oFormulaire, stListeValeurs reçoit la valeur courante de oChamp, c'est-à-dire le ou les caractères saisis par l'utilisateur. Ensuite, le champ « Auteur » est filtré à l'aide de cette chaîne de caractère, puis le formulaire est rechargé (reload()).

Après cela le pointeur dans la liste de sélection se place automatiquement sur le premier « Auteur » commençant par les caractères saisis. Exemple : **Ber** amène l'affichage sur **Beran**, **Armgard**, et le contrôle de table n'affiche que les enregistrements concernant cet auteur. Le choix d'un auteur différent, dans la liste de sélection, modifie l'affichage dans le contrôle de table.

La seconde solution (formulaire : Limite_Liste) proposée ci-dessous passe par la saisie d'un critère de recherche d'**Auteur**, toujours par la saisie de quelques caractères dans une « zone de texte » indépendante. Ces caractères pouvant être situés au début, dans, ou à la fin de la chaîne de caractères. Ces choix étant possibles à l'aide de trois boutons radio liés entre eux en un groupe dénommé « Choix » (**Propriétés – Général – Nom du groupe**).

Les propriétés du formulaire et des divers contrôles pourront aisément être vues en ouvrant le formulaire en mode édition.

La première procédure est lancée par l'appui sur la touche TAB ou sur la touche Entrée, après la saisie de quelques caractères. Le commentaire se suffit à lui-même.

```
Sub Lanceur (oEvent As Object)
'-----
' Verifie si le KeyCode de la touche enfoncée est
' 1280 - Entrée ou 1282 - Tabulation
'-----
If oEvent.KeyCode = 1280 or oEvent.KeyCode = 1282 Then
    Limiter (oEvent)
Else
    Exit Sub
End If
```

End Sub

Si l'une des touches attendue est actionnée la macro « Limiter » est déclenchée avec oEvent en paramètre, sinon on sort de la macro (Exit Sub). La liste complète des « Key_Code » peut être obtenue dans l'API : [com::sun::star::awt::Key Constant Group Reference](#)

En premier lieu de nombreuses variables sont déclarées avec leurs types respectifs.

```
Sub Limiter (oEvent As Object)
    Dim oDrawpage As Object
    Dim oFormulaire As Object
    Dim oZone_Texte As Object
    Dim oZone_Liste As Object
    Dim oChamp As Object
    Dim oRadio As Object
    Dim oRadio1 As Object
    Dim oRadio2 As Object
    Dim oRadio3 As Object
    Dim oSourceDonnees As Object
    Dim oConnexion As Object
    Dim oCommande_Sql As Object
    Dim oResultat As Object
    Dim stCritere As String
    Dim stRequete As String
    Dim arContenu()
    Dim i As Integer
    ' - Initialise les zones de texte et de liste à "" (vide)
    oDrawpage = thisComponent.drawpage
    oFormulaire = oDrawpage.forms.getByName("Formulaire")
    oZone_Texte = oFormulaire.getByName("Critere")
    oZone_Texte.DefaultText = ""
```

Ici le texte par défaut de « oZone_Texte » est fixé à vide(""), il s'agit du texte par défaut contenu dans la zone de texte, nommée « Critere », dédié à la saisie de caractères par l'utilisateur.

```
oZone_Liste = oFormulaire.getByName("Liste")
oZone_Liste.StringItemList = arContenu()
```

Là, pour le contrôle de liste déroulante, nommée « Liste » c'est également un contenu vide qui est attribué. Vide car « arContenu » a été déclaré sans qu'aucune valeur ne lui soit affecté.

```
' - Récupère les caractères entrés comme critère
oChamp = oEvent.Source.Model
stCritere = oChamp.getCurrentValue()
' - Récupère l'état des boutons radio pour connaître celui qui est activé
oRadio = oChamp.Parent
oRadio1 = oRadio.getByName("Bouton radio 1")
oRadio2 = oRadio.getByName("Bouton radio 2")
oRadio3 = oRadio.getByName("Bouton radio 3")
```

Les commentaires pour les lignes au-dessus sont suffisants à la compréhension.

```
If stCritere <> "" Then
    If oRadio1.State = 1 Then ' - Critère au début - bouton 1
        stCritere = stCritere & "%"
    ElseIf oRadio2.State = 1 Then ' - Critère au milieu - bouton 2
        stCritere = "%" & stCritere & "%"
    ElseIf oRadio3.State = 1 Then ' - Critère à la fin - bouton 3
        stCritere = "%" & stCritere
    End If
Else
    MsgBox ("Vous n'avez saisi aucun critère")
End If
```

Si « stCritere » n'est pas vide alors, suivant l'état (State) des boutons radio (activé = 1 – non activé = 0), on modifie cette chaîne de caractère pour qu'elle soit ensuite utilisée dans la requête SQL (interne à la procédure) suivante.

Pour rappel le symbole « % » remplace un ou plusieurs caractères dans une chaîne utilisée avec un LIKE.

Ainsi :

- xyz% = toute chaîne commençant par "xyz" ;
- %xyz% = toute chaîne dans laquelle se trouve "xyz" ;
- %xyz = toute chaîne se terminant par "xyz".

```
' - Construction de la Requête
stRequete = "SELECT DISTINCT ""Auteur"" FROM ""TableCherche"" WHERE _
            ""Auteur"" LIKE ' " & stCritere & "' ORDER BY ""Auteur"" ASC"
' - Connexion à la source de donnée si ce n'est pas le cas
oSourceDonnees = ThisComponent.Parent.CurrentController
If Not (oSourceDonnees.isConnected()) Then
    oSourceDonnees.connect()
End IF
oConnexion = ThisComponent.Parent.CurrentController.ActiveConnection()
' - Exécution de la requête SQL
oCommande_Sql = oConnexion.createStatement()
oResultat = oCommande_Sql.executeQuery(stRequete)
```

Sans commentaires.

```
' Crée un tableau 'arContenu' avec l'ensemble des données résultant de la requête
i = 0
While oResultat.next
    ReDim Preserve arContenu(i)
    arContenu(i)=oResultat.getString(1)
    i=i+1
Wend
oZone_Liste.StringItemList = arContenu()
End Sub
```

Après la boucle de lecture des résultats de la requête et leur affectation à la variable tableau arContenu() on remplit la liste déroulante avec ces résultats. Ainsi cette zone de liste sera réduite aux seuls champs « Auteur » correspondants aux critères saisis.

Lors de la sélection d'un « Auteur » dans la liste déroulante, la même macro que précédemment (Filtre_direct(oEvent AS Object)) est utilisée.

Il est possible de procéder à plusieurs recherches les unes après les autres.

```
Sub Ferme_Formulaire_Sans_Sauvegarder(oEvent As Object)
    oEvent.Source.Model.Parent.isModified = False
    ThisDatabaseDocument.FormDocuments.getByName("Limite_Liste").close
End Sub
```

Enfin, pour quitter le formulaire la macro précédente est liée au bouton « Quitter la recherche » (**Propriétés – Événement – Exécuter l'action**) ainsi qu'au formulaire lui-même (**Propriétés – Événement – Lors du déchargement**). En effet, il n'est pas souhaitable ici d'enregistrer les éléments contenus dans les divers contrôles.

Conversion de dates d'un formulaire en une variable de date

```
Function ValeurDate(oChamp As Object) As Date
    a() = OfficeVersion()
    If a(0) = "LibreOffice" And (LEFT(a(1),1) =4 And RIGHT(LEFT(a(1),3),1) > 0) _
        Or LEFT(a(1),1) > 4 Then
```

Ici, toutes les versions de LibreOffice à partir de la 4.1 sont interceptées. À cette fin, le numéro de version est divisé en ses éléments individuels, et les numéros de version majeure et mineure sont vérifiés. Cela fonctionnera jusqu'à LibreOffice 9.

```
Dim stAn As String
Dim stMois As String
Dim stJour As String
stAn = Right(Ltrim(Str(oChamp.CurrentValue.Year)), 4)
stMois = Right("0" & Ltrim(Str(oChamp.CurrentValue.Month)), 2)
stJour = Right("0" & Ltrim(Str(oChamp.CurrentValue.Day)), 2)
ValeurDate = CDateFromIso(stAn & stMois & stJour)
Else
    ValeurDate = CDateFromIso(oChamp.CurrentValue)
End If
End Function
```

Depuis LibreOffice 4.1.2, les dates sont stockées sous forme de tableaux dans les contrôles de formulaire. Cela signifie que la valeur actuelle du contrôle ne peut pas être utilisée pour accéder à la date elle-même. La date doit être recrée à partir du jour, du mois et de l'année si elle doit être utilisée ultérieurement dans les macros⁸.

Recherche d'enregistrements de données

Dans une base de données vous pouvez rechercher tous les enregistrements qui contiennent, dans un de leurs champs, une donnée (numérique, chaîne de caractère) sans utiliser de macro. Cependant, la conception d'une requête SQL sera très complexe. Une macro peut résoudre ce problème grâce aux boucles **While... Wend** et **For... Next**⁹.

Lancée par une procédure principale, qui lui transmet le nom de la table à parcourir en tant que variable, la sous-procédure suivante réalise la recherche.

Pour ce faire elle :

- lit le critère de recherche dans le formulaire ;
- vérifie que la connexion à la base de donnée est bien établie ;
- exécute en interne une requête SQL ;
- parcourt tous les champs de la table ;
- écrit finalement une liste de numéros, correspondant aux clés primaires des enregistrements de la table qui sont extraits par le terme de recherche, dans une table temporaire.

Cette table Cherchetmp, se compose d'une clé primaire à incrémentation automatique (ID) et d'un champ appelé (Nr). Il contient toutes les clés primaires extraites de la table dans laquelle la recherche est effectuée.

Pour obtenir un résultat correct, la table doit contenir le contenu que vous recherchez sous forme de texte et non de clé étrangère.

S'il n'y a pas qu'une seule table, dans laquelle effectuer la recherche, le contenu des tables doit être joint dans une VUE que la macro utilisera.

```
Sub Rechercher(stTable As String)
    Dim oSourceDonnees As Object
    Dim oConnexion As Object
    Dim oSQL_Commande As Object
    Dim stSql As String
    Dim oResultat As Object
```

⁸ Voir la base de données Exemple_DateFormulaire_en_Variable.odt

⁹ Voir la base de données Exemple_Cherche_et_Filtre.odt, Formulaire RequeteCherche, associée à ce manuel.

```

Dim oDoc As Object
Dim oDrawpage As Object
Dim oFormulaire As Object
Dim oFormulaire2 As Object
Dim oChamp As Object
Dim stContenu As String
Dim arContenu() As String
Dim inI As Integer
Dim inK As Integer
oDoc = thisComponent
oDrawpage = oDoc.drawpage
oFormulaire = oDrawpage.forms.getByNamed("FormulaireRecherche")
oChamp = oFormulaire.getByNamed("TexteCherche")
stContenu = oChamp.getCurrentValue()
stContenu = LCase(stContenu)

```

Le contenu (`stContenu`) du champ de texte de recherche ("`TexteCherche`") est converti en minuscules (`LCase`) de sorte que la fonction de recherche suivante n'a besoin que de comparer les écritures en minuscules. La connexion à la source de données est vérifiée et si besoin établie.

```

oSourceDonnees = ThisComponent.Parent.DataSource
oConnexion = oSourceDonnees.GetConnection("", "")
oSQL_Commande = oConnexion.createStatement()

```

Il faut d'abord déterminer si un terme de recherche a été saisi. Si le champ est vide, on supposera qu'aucune recherche n'est requise. Tous les enregistrements seront affichés.

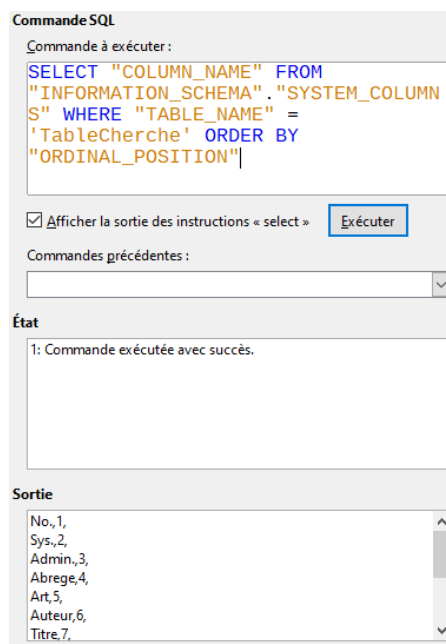
Si un terme de recherche a été saisi, les noms de colonne sont lus à partir de la table recherchée, afin que la requête puisse accéder aux champs.

```

If stContenu <> "" Then
    stSql = "SELECT ""COLUMN_NAME"" FROM ""INFORMATION_SCHEMA"". ""SYSTEM_COLUMNS""
    WHERE ""TABLE_NAME"" = ' " + stTable + "' ORDER BY ""ORDINAL_POSITION"" "
    oResultat = oSQL_Commande.executeQuery(stSql)

```

Dans les bases de données relationnelles, le schéma d'information (`INFORMATION_SCHEMA`) est un ensemble de vues, en lecture seule, également appelées tables système. Cet ensemble fournit des informations sur toutes les tables, vues, colonnes et procédures d'une base de données. Ici la vue utilisée est `SYSTEM_COLUMNS`, qui permet d'obtenir tous les noms des colonnes de la table `stTable`. Il est possible de visualiser le résultat de cette requête par **OUTILS > SQL...** comme montré sur l'illustration suivante :





Note

Les formules SQL dans les macros doivent d'abord être placées entre guillemets comme les chaînes de caractères normales. Les noms de champ et les noms de table sont déjà entre guillemets dans la formule SQL. Pour créer un code final qui transmet correctement les guillemets doubles, les noms de champ et les noms de table doivent recevoir deux ensembles de ces guillemets.

`stSql = "SELECT ""Nom"" FROM ""Table""";`, devient lorsqu'il est affiché avec la commande `MsgBox stSql : SELECT "Nom" FROM "Table";`

L'index du tableau dans lequel les noms de champs sont écrits est initialement défini sur 0. Ensuite, la requête commence à être lue. La taille du tableau étant inconnue, elle doit être ajustée en continu. C'est pourquoi la boucle commence par **ReDim Preserve arContenu (inI)** pour définir la taille du tableau et en même temps conserver son contenu existant. Ensuite, les champs sont lus et l'index du tableau est incrémenté de 1. Puis, le tableau est de nouveau dimensionné et une autre valeur peut être stockée.

```
InI = 0
While oResultat.next
    ReDim Preserve arContenu(inI)
    arContenu(inI) = oResultat.getString(1)
    inI = inI + 1
Wend
stSql = "DROP TABLE ""Cherchetmp"" IF EXISTS"
oSQL_Commande.executeUpdate (stSql)
```

La requête est maintenant regroupée dans une boucle et ensuite appliquée à la table définie au début. Toutes les combinaisons de casse sont autorisées, car le contenu du champ de la requête est converti en minuscules.

La requête est construite de telle sorte que les résultats aboutissent dans la table « Cherchetmp ». On suppose que la clé primaire est le premier champ de la table (arContenu (0)).

```
stSql = "SELECT """+arContenu(0)+""" INTO ""Cherchetmp"" FROM "" + stTable _
+ "" WHERE "
For inK = 0 To (inI - 1)
    stSql = stSql+"LCase('"+arContenu(inK)+"') LIKE '%" + stContenu+"%'"
    If inK < (inI - 1) Then
        stSql = stSql+" OR "
    End If
Next
oSQL_Commande.executeQuery(stSql)
Else
    stSql = "DELETE FROM ""Cherchetmp""
    oSQL_Commande.executeUpdate (stSql)
End If
```

Le formulaire d'affichage doit être rechargé. Sa source de données est une requête (dans cet exemple RequeteCherche).

```
oFormulaire2 = oDrawpage.forms.getByName("Affiche")
oFormulaire2.reload()
End Sub
```

Dans la mesure du possible, cette requête doit être construite de manière à pouvoir être modifiée ultérieurement.

```
SELECT * FROM "TableCherche" WHERE "No." IN (SELECT "No." FROM "Cherchetmp") OR
"No." = CASE WHEN (SELECT COUNT("No.") FROM "Cherchetmp") > 0 THEN '0' ELSE "No."
END
```

Tous les éléments de la **TableCherche** sont inclus, y compris la clé primaire. Aucune autre table n'apparaît dans la requête directe. Par conséquent, aucune clé primaire d'une autre table n'est nécessaire et le résultat de la requête reste modifiable.

La clé primaire est enregistrée dans cet exemple sous le nom « **No.** ». La macro lit précisément ce champ. Il y a une vérification initiale pour voir si le contenu du champ « **No.** » apparaît dans la table Cherchetmp. L'opérateur **IN** est compatible avec plusieurs valeurs. La sous-requête peut également générer plusieurs enregistrements.

Pour de plus grandes quantités de données, la correspondance des valeurs à l'aide de l'opérateur **IN** ralentit rapidement. Par conséquent, il n'est pas judicieux d'utiliser un champ de recherche vide simplement pour transférer tous les champs de clé primaire de la table de recherche dans la table **Cherchetmp**, puis d'afficher les données de la même manière. Au lieu de cela, un champ de recherche vide crée une table **Cherchetmp** vide, de sorte qu'aucun enregistrement n'est disponible. C'est le but de la seconde moitié de la condition :

```
OR "No." = CASE WHEN (SELECT COUNT("No.") FROM "Cherchetmp") > 0 THEN '-1'
ELSE "No." END
```

Si un enregistrement est trouvé dans la table Cherchetmp, cela signifie que le résultat de la première requête est supérieur à 0. Dans ce cas : **"No." = '-1'** (ici nous avons besoin d'un nombre qui ne peut pas apparaître comme clé primaire, donc **'-1'** est une bonne valeur). Si la requête renvoie précisément 0 (ce qui sera le cas si aucun enregistrement n'est présent), alors **"No." = "No."**. Cela listera tous les enregistrements qui ont un **"No."**. Comme **No.** est la clé primaire, cela signifie tous les enregistrements.

Recherche dans les formulaires et mise en évidence des résultats

Avec un grand champ de texte, il est souvent difficile de savoir où les correspondances à un terme de recherche se produisent. Ce serait bien si le formulaire pouvait mettre en évidence les correspondances. Cela devrait ressembler à ce que montre l'illustration suivante :

Pour qu'un formulaire fonctionne de cette manière, nous avons besoin de quelques éléments supplémentaires dans notre boîte d'astuces.¹⁰

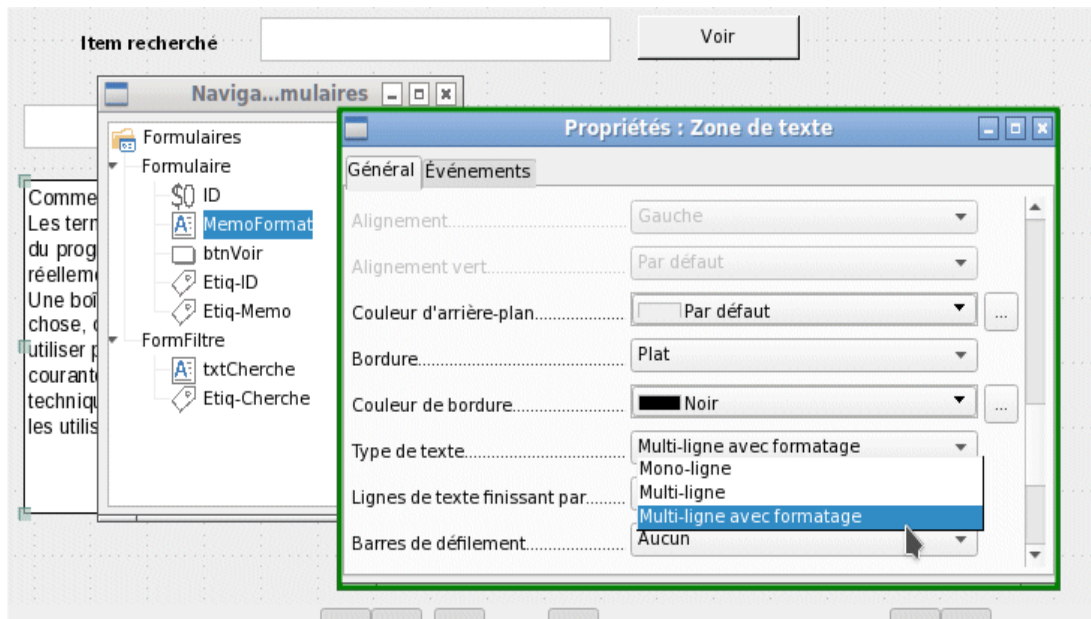
Le fonctionnement d'un champ de recherche comme celui-ci a déjà été expliqué. Une table de filtres est créée et un formulaire est utilisé pour écrire les valeurs actuelles d'un seul enregistrement dans cette table. Le formulaire principal est fourni avec son contenu à l'aide d'une requête qui ressemble à ceci :

```
SELECT "ID", "Memo" FROM "Table" WHERE LOWER("Memo") LIKE '%' || LOWER (
(SELECT "TexteCherche" FROM "Filtre" WHERE "ID" = TRUE)) || '%'
```

¹⁰ Voir la base de données Exemple_Autotexte_Recherche_Orthographe.odb associée à ce manuel.

Lorsque le texte de recherche est entré, tous les enregistrements de la table « Table » qui ont le texte de recherche dans le champ « Memo » sont affichés. La recherche n'est pas sensible à la casse.

Si aucun texte de recherche n'est entré, tous les enregistrements du tableau sont affichés. La clé primaire de cette table étant incluse dans la requête, cette dernière peut être éditée.



Dans le formulaire, en plus du champ ID de la clé primaire, il y a un champ appelé MemoFormat qui a été configuré (en utilisant **Propriétés> Général> Type de texte> Multi-lignes avec formatage**) pour afficher le texte en couleur et en noir. Un examen attentif des propriétés du champ de texte révèle que l'onglet Données a maintenant disparu. En effet, les données ne peuvent pas être saisies dans un champ qui a une mise en forme supplémentaire que la base de données elle-même ne peut pas stocker. Néanmoins, il est toujours possible de mettre du texte dans ce champ, de le baliser, et de le transférer après une mise à jour en utilisant une macro.

La procédure *LireContenu* sert à transférer le contenu du champ de base de données « Memo » dans le champ de texte formaté MemoFormat, et à le formater de sorte que tout texte correspondant à celui du champ de recherche soit mis en évidence.

La procédure est liée à **Formulaire> Événements> Après le changement d'enregistrement**.

```
Sub LireContenu(oEvent As Object)
    Dim inMemo As Integer
    Dim oChamp As Object
    Dim stChercheTexte As String
    Dim oCurseur As Object
    Dim inCherche As Integer
    Dim inChercheVieux As Integer
    Dim inLongueur As Integer
    oFormulaire = oEvent.Source
    inMemo = oFormulaire.findColumn("Memo")
    oChamp = oFormulaire.getByName("MemoFormat")
    oChamp.Text = oFormulaire.getString(inMemo)
```

Les variables sont d'abord définies. Ensuite, le champ de table « Memo » est recherché dans le formulaire et la fonction **getString()** est utilisée pour lire le texte de la colonne numérotée. Celui-ci est transféré dans le champ qui peut être formaté mais qui n'a pas de lien vers la base de données : MemoFormat.

Les tests initiaux ont montré que le formulaire s'ouvrait mais que la barre d'outils du formulaire en bas n'était plus créée. Par conséquent, une très courte attente de 5/1000 secondes a été intégrée. Après cela, le contenu affiché est lu à partir du FormFiltre (qui est parallèle au formulaire dans la hiérarchie des formulaires).

```
Wait 5
stChercheTexte = oFormulaire.Parent.getByName("FormFiltre")._
               getByName("txCherche").Text
```

Pour pouvoir mettre en forme du texte, un **TextCursor** (invisible) doit être créé dans le champ qui contient le texte. L'affichage par défaut du texte utilise une police serif de 12 points qui peut ne pas apparaître dans d'autres parties du formulaire et ne peut pas être directement personnalisée à l'aide des propriétés de contrôle de formulaire. Dans cette procédure, le texte est défini sur l'apparence souhaitée dès le début. Si cela n'est pas fait, les différences de formatage peuvent entraîner la coupure de la limite supérieure du texte dans le champ. Lors des premiers tests, seuls 2/3 de la première ligne étaient lisibles.

Pour que le curseur invisible marque le texte, il est défini initialement au début du champ, puis à la fin. L'argument dans les deux cas est **True**. Viennent ensuite les spécifications concernant la taille de la police, le type de la police, la couleur et la taille. Ensuite, le curseur revient au début.

```
oCurseur = oChamp.createTextCursor()
oCurseur.gotoStart(true)
oCurseur.gotoEnd(true)
oCurseur.CharHeight = 10
oCurseur.CharFontName = "Arial, Helvetica, Tahoma"
oCurseur.CharColor = RGB(0,0,0)
oCurseur.CharWeight = 100.000000 'com:: sun:: star:: awt:: FontWeight
oCurseur.gotoStart(false)
```

S'il y a du texte dans le champ et qu'une entrée a été faite demandant une recherche, ce texte est maintenant recherché pour trouver la chaîne de recherche. La boucle externe demande d'abord si ces conditions sont remplies ; celle de l'intérieur établit si la chaîne de recherche est vraiment dans le texte du champ MemoFormat. Ces paramètres peuvent en fait être omis, car la requête sur laquelle le formulaire est basé n'affiche que le texte qui remplit ces conditions.

```
If oChamp.Text <> "" And stChercheTexte <> "" Then
  If InStr(oChamp.Text, stChercheTexte) Then
    inCherche = 1
    inChercheVieux = 0
    inLen = Len(stChercheTexte)
```

Le texte est parcouru pour la chaîne de recherche. Cela se déroule dans une boucle qui se termine lorsqu'aucune autre correspondance n'est affichée. **InStr()** renvoie l'emplacement du premier caractère de la chaîne de recherche dans le format d'affichage spécifié, indépendamment de la casse. La boucle est contrôlée par l'exigence qu'à la fin de chaque cycle, le début de **inCherche** soit incrémenté de 1 (-1 dans la première ligne de la boucle et +2 dans la dernière ligne). Pour chaque cycle, le curseur est déplacé vers la position initiale sans marquage à l'aide de **oCurseur.goRight (Position, false)**, puis vers la droite avec un marquage par la longueur de la chaîne de recherche. Ensuite, le formatage souhaité (bleu et un peu plus gros) est appliqué et le curseur est ramené à son point de départ suivant pour la prochaine exécution.

```
Do While InStr(inCherche, oChamp.Text, stChercheTexte) > 0
  inCherche = InStr(inCherche, oChamp.Text, stChercheTexte) - 1
  oCurseur.goRight(inCherche-inChercheVieux, false)
  oCurseur.goRight(inLen, true)
  oCurseur.CharColor = RGB(102,102,255)
  oCurseur.CharWeight = 110.000000
  oCurseur.goLeft(inLen, false)
  inChercheVieux = inCherche
  inCherche = inCherche + 2
Loop
End If
End If
```

End Sub

La procédure **EcrireContenu** sert à transférer le contenu du champ de texte formatable MemoFormat dans la base de données. Cela se déroule indépendamment du fait qu'une modification ait lieu ou non.

La procédure est liée à **Formulaire> Événements> Avant le changement d'enregistrement**.

```
Sub EcrireContenu(oEvent As Object)
    Dim oFormulaire As Object
    Dim inMemo As Integer
    Dim loID As Long
    Dim oChamp As Object
    Dim stMemo As String
    oFormulaire = oEvent.Source
    If InStr(oFormulaire.ImplementationName, "ODatabaseForm") Then
```

L'événement déclencheur est implémenté deux fois. Seul le nom d'implémentation qui se termine par ODatabaseForm donne l'accès correct à l'enregistrement (les implémentations sont expliquées à la page 76).

```
        If Not oFormulaire.isBeforeFirst() And Not oFormulaire.isAfterLast() Then
```

Lorsque le formulaire est lu ou rechargé, le curseur se place avant l'enregistrement en cours. Ensuite, si une tentative est effectuée, vous obtenez le message "État du curseur non valide".

```
            inMemo = oFormulaire.findColumn("Memo")
            loID = oFormulaire.findColumn("ID")
            oChamp = oFormulaire.getByName("MemoFormat")
            stMemo = oChamp.Text
            If stMemo <> "" Then
                oFormulaire.updateString(inMemo, stMemo)
            End If
            If stMemo <> "" And oFormulaire.getString(loID) <> "" Then
                oFormulaire.UpdateRow()
            End If
        End If
    End If
End Sub
```

Le champ de la table « Memo » est localisé à partir de la source de données du formulaire, avec celui de « ID ». Si le champ MemoFormat contient du texte, il est transféré dans le champ Memo de la source de données à l'aide de **oFormulaire.updateString ()**. Une mise à jour est effectuée uniquement s'il y a une entrée dans le champ ID (en d'autres termes, une clé primaire a été définie). Sinon, un nouvel enregistrement est inséré dans le cadre du fonctionnement normal du formulaire, le formulaire reconnaît le changement et le stocke indépendamment.

Un bouton situé dans le formulaire principal permet de rafraîchir le formulaire en cas de modification du champ texte recherché, et donc de mettre à jour. L'affichage du champ mémo du compteur d'enregistrements. **Propriétés> Général> Action> Rafraîchir le formulaire**.

Vérification orthographique lors de la saisie des données

Cette macro peut être utilisée pour les **champs de texte multilignes avec formatage**. Comme dans le chapitre précédent, le contenu de chaque enregistrement doit d'abord être écrit, puis le nouvel enregistrement peut être chargé dans le contrôle de formulaire. Les procédures LireContenu et EcrireContenu ne diffèrent que par le point auquel la fonction de recherche peut être mise entre guillemets.

introduction

Dans les opérations quotidiennes de bureau, les feuilles de calcul sont régulièrement utilisées pour agréger des ensembles de données et pour effectuer des analyses sur ces derniers. Comme les données d'une feuille de calcul sont présentées dans une vue de tableau, clairement visibles et peuvent être modifiées ou ajoutées, de nombreux utilisateurs se demandent pourquoi ils devraient utiliser une base de données au lieu d'une feuille de calcul. Ce manuel explique les différences entre les deux, en commençant par une courte section sur ce qui peut être fait avec une base de données. Ce chapitre présente deux exemples de bases de données et l'ensemble du manuel est construit autour de ceux-ci. Une base de données est nommée Media_sans_macros.odt et l'autre, étendue avec l'inclusion de macros, est nommée Media_avec_macros.odt.

Le vérificateur d'orthographe de LibreOffice est lancé dans le formulaire ci-dessus chaque fois qu'un espace ou un retour est frappé dans le contrôle de formulaire. En d'autres termes, il s'exécute à la fin de chaque mot. Il pourrait également être lié au contrôle perdant le focus pour s'assurer que le dernier mot est vérifié.

La procédure est liée à **Formulaire> Événements> Touche relâchée**.

```
Sub MarqueMauvaisMotDirect(oEvent As Object)
    GlobalScope.BasicLibraries.LoadLibrary("Tools")
```

La fonction **RTrimStr** est utilisée pour supprimer tout signe de ponctuation à la fin de la chaîne. Sinon, tous les mots qui se terminent par une virgule, un point ou un autre signe de ponctuation apparaîtront comme des fautes d'orthographe. De plus, **LTrimChar** est utilisé pour supprimer les guillemets au début des mots¹¹.

```
Dim aProp() As New com.sun.star.beans.PropertyValue
Dim oLinuSvcMgr As Object
Dim oSpellChk As Object
Dim oChamp As Object
Dim arTexte()
Dim stMot As String
Dim inLongueurMot As Integer
Dim ink As Integer
Dim i As Integer
Dim oCurseur As Object
Dim stTexte As String
oLinguSvcMgr = createUnoService("com.sun.star.linguistic2.LinguServiceManager")
If Not IsNull(oLinguSvcMgr) Then
    oSpellChk = oLinguSvcMgr.getSpellChecker()
End If
```

Tout d'abord, toutes les variables sont déclarées. Ensuite, le module de vérification orthographique de base **SpellChecker** est accédé. Ce sera ce module qui vérifiera l'exactitude des mots individuels.

```
oChamp = oEvent.Source.Model
ink = 0
If oEvent.KeyCode = 1280 Or oEvent.KeyCode = 1284 Then
```

L'événement qui lance la macro est une frappe. Cet événement comprend un code, le **KeyCode**, pour chaque touche individuelle. Le **KeyCode** de la touche « Entrée » est 1280, celui de l'espace est 1284. Comme beaucoup d'autres informations, ces éléments ont été récupérés via l'outil Xray.

11 Pour l'utilisation de cette macro, voir l'exemple Exemple_Autotexte_Recherche_Orthographe.odt, associée à ce manuel.

Si vous appuyez sur espace ou entrée, l'orthographe est vérifiée. Il est lancé, en d'autres termes, à la fin de chaque mot. Seul le test du dernier mot ne se produit pas automatiquement.

Chaque fois que la macro s'exécute, tous les mots du texte sont vérifiés. La vérification de mots individuels pourrait également être possible, mais demanderait beaucoup plus de travail.

Le texte est divisé en mots simples. Le délimiteur est le caractère d'espace. Avant cela, les mots séparés par des sauts de ligne doivent être réunis à nouveau, sinon les morceaux peuvent être confondus avec des mots complets.

```
stTexte = Join(Split(oChamp.Text, CHR(10)), " ")
stTexte = Join(Split(stTexte, CHR(13)), " ")
arTexte = Split(RTrim(stTexte), " ")
For i = LBound(arText) To Ubound(arText)
    stMot = arTexte(i)
    inLongueurMot = Len(stMot)
    stMot = Trim(RTrimStr(RTrimStr(RTrimStr(RTrimStr(RTrimStr( _
        RTrimStr(stMot, ","), "."), "?"), "!"), "."), ")))
    stMot = LTrimChar(stMot, "(")
```

Les mots individuels sont lus. Leur longueur non tronquée est nécessaire pour l'étape d'édition suivante. Ce n'est qu'ainsi que la position du mot dans tout le texte (qui est nécessaire pour la mise en évidence spécifique des fautes d'orthographe) peut être déterminée.

Trim est utilisé pour supprimer les espaces, tandis que **RTrimStr** supprime les virgules et les points à la fin du texte et **LTrimChar** tous les signes de ponctuation au début.

```
If stMot <> "" Then
    oCurseur = oChamp.createTextCursor()
    oCurseur.gotoStart(False)
    oCurseur.goRight(ink, False)
    oCurseur.goRight(inLongueurMot, True)
    If Not oSpellChk.IsValid(stMot, "fr", aProp()) Then
        oCurseur.CharUnderline = 9
        oCurseur.CharUnderlineHasColor = True
        oCurseur.CharUnderlineColor = RGB(255, 51, 51)
    Else
        oCurseur.CharUnderline = 0
    End If
End If
ink = ink + inLongueurMot + 1
Next
End If
End Sub
```

Si le mot n'est pas nul, un curseur de texte est créé. Ce curseur est déplacé sans surligner au début du texte dans le champ de saisie. Ensuite, il saute vers la droite, toujours sans mise en évidence, au terme stocké dans la variable **ink**. Cette variable débute par 0, mais après l'exécution de la première boucle, elle est égale à la longueur du mot (+1 pour l'espace suivant). Ensuite, le curseur est déplacé vers la droite de la longueur du mot actuel. Les propriétés de police sont modifiées pour créer la région en surbrillance.

Le correcteur orthographique est lancé. Il requiert le mot et le code du pays comme arguments, sans code de pays tout compte comme correct. L'argument tableau est généralement vide.

Si le mot ne figure pas dans le dictionnaire, il est marqué d'une ligne ondulée rouge. Ce type de soulignement est ici représenté par « 9 ». Si le mot est trouvé, il n'y a pas de soulignement (0). Cette étape est nécessaire car sinon, un mot reconnu comme faux puis corrigé continuerait à être affiché avec la ligne ondulée rouge. Il ne serait jamais supprimé car aucun format conflictuel n'a été indiqué.

Boîtes combinées sous forme de Zone de liste avec une option d'entrée

Une table avec un seul enregistrement peut être directement créée en utilisant des Zones combinées et des champs numériques invisibles et la clé primaire correspondante entrée dans une autre table.¹²

Le contrôle Zone combinée traite les champs de formulaire en combinant l'entrée et le choix des valeurs (comboboxes) comme des Zones de liste avec une option de saisie. À cet effet, en plus des combobox du formulaire, les valeurs de champ clé qui doivent être transférées à la table sous-jacente sont stockées dans des champs numériques séparés. Les champs peuvent être déclarés comme invisibles. Les clés de ces champs sont lues lorsque le formulaire est chargé et la zone de liste déroulante est définie pour afficher le contenu correspondant. Si le contenu de la liste déroulante est modifié, il est enregistré et la nouvelle clé primaire est transférée dans le champ numérique correspondant pour être stockée dans la table principale.

Si des requêtes modifiables sont utilisées à la place des tables, le texte à afficher dans les combinaisons de champs peut être directement déterminé à partir de la requête. Une macro n'est alors pas requise pour cette étape.

Une hypothèse pour le fonctionnement de la macro est que la clé primaire de la table qui est la source de données pour le champ de combinaison est un entier à incrémentation automatique. On suppose également que le nom de champ de la clé primaire est ID.

Affichage du texte dans les zones combinées

Ce sous-programme consiste à afficher le texte dans la zone de liste déroulante en fonction de la valeur des champs de clé étrangère invisibles du formulaire principal. Il peut également être utilisé pour les listes de sélection qui renvoient à deux tables différentes. Cela peut se produire si, par exemple, le code postal d'une adresse postale est stocké séparément de la ville. Dans ce cas, le code postal peut être lu à partir d'une table contenant uniquement une clé étrangère pour la ville. La zone de liste doit afficher le code postal et la ville ensemble¹³.

```
Sub AfficheTexte(oEvent As Object)
```

Cette macro doit être liée à l'événement de formulaire suivant : "Après le changement d'enregistrement".

La macro est appelée directement depuis le formulaire. L'événement déclencheur est la source de toutes les variables dont la macro a besoin. Certaines variables ont déjà été déclarées globalement dans un module séparé et ne sont plus déclarées ici.

```
Dim oFormulaire As Object
Dim oChampliste As Object
Dim stValeurChamp As String
Dim inZones As Integer
Dim stRequete As String
oFormulaire = oEvent. Source
```

Dans le formulaire, il y a un contrôle caché à partir duquel les noms de toutes les différentes listes déroulantes peuvent être obtenus. Une par une, ces zones de liste sont traitées par la macro.

```
aZonesCombinees() = Split(oFormulaire.getByName("ZonesCombinees").Tag, ",")
For inZones = LBound(aZonesCombinees) TO UBound(aZonesCombinees)
...
Next inZone
```

12 Pour l'utilisation de zones combinées au lieu de zones de liste, consultez la base de données Exemple_ZoneCombinee_ListeChamps.odt associée à ce manuel.

13 Des applications de cette procédure se trouvent dans l'exemple Media_avec_Macros

Le complément d'information (Tag) attaché au contrôle masqué contient cette liste de noms de zones de liste, séparés par des virgules. Les noms sont écrits dans un tableau puis traités dans une boucle. La boucle se termine par l'instruction **NEXT**.

La zone combinée, qui a remplacé une zone de liste, s'appelle oListeChamps. Pour obtenir la clé étrangère, nous avons besoin de la colonne correcte dans la table qui sous-tend le formulaire. Ceci est accessible en utilisant le nom du champ de table, qui est stocké dans les informations supplémentaires de la liste déroulante.

```
oChampListe = oFormulaire.getByName(Trim(aZonesCombinees(inZone)))
oChamp = oFormulaire.getString(oFormulaire.findColumn(oChampListe.Tag))
oChampListe.Refresh()
```

La zone combinée est lue à nouveau en utilisant **Refresh()** au cas où le contenu du champ a été modifié par l'entrée de nouvelles données.

La requête nécessaire pour fournir le contenu visible de la zone combinée est basée sur le champ sous-jacent du contrôle et la valeur déterminée pour la clé étrangère. Pour rendre le code SQL utilisable, toute opération de tri qui pourrait être présente est supprimée. Ensuite, une vérification est effectuée pour toutes les définitions de relation (qui commenceront par le mot **WHERE**). Par défaut, la fonction **InStr()** ne fait pas la distinction entre les majuscules et les minuscules, donc toutes les combinaisons de casse sont couvertes. S'il existe une relation, cela signifie que la requête contient des champs de deux tables différentes. Nous devons trouver la table qui fournit la clé étrangère pour le lien. La macro dépend ici du fait que la clé primaire de chaque table s'appelle ID.

Si aucune relation n'est définie, la requête accède à une seule table. Les informations de table peuvent être ignorées et la condition formulée directement à l'aide de la valeur de clé étrangère.

```
If NOT IsEmpty(oChamp.getCurrentValue()) Then
    stRequete = oChampListe.ListSource
    If InStr(stRequete,"order by") > 0 Then
        stSql = Left(stRequete, InStr(stRequete,"order by")-1)
    Else
        stSql = stRequete
    End If
    If InStr(stSql,"where") Then
        st = Right(stSql, Len(stSql)-InStr(stSql,"where")-4)
        If InStr(Left(st, InStr(st,"=")), ""ID"") Then
            a() = Split(Right(st, Len(st)-InStr(st,"=")-1), ".")
        Else
            a() = Split(Left(st, InStr(st,"=")-1), ".")
        End If
        stSql = stSql + "AND " + a(0) + ""ID" = " + oChamp.getCurrentValue
    Else
        stSql = stSql + "WHERE ""ID" = " + oChamp.getCurrentValue
    End If
```

Chaque nom de champ et de table doit être entré dans la commande SQL avec deux ensembles de guillemets. Les guillemets sont normalement interprétés par Basic comme des délimiteurs de chaîne de texte, ils n'apparaissent donc plus lorsque le code est transmis à SQL. Le fait de doubler les guillemets garantit qu'un jeu est transmis. ""ID"" signifie que le champ "ID" sera accessible dans la requête, avec l'ensemble unique de guillemets requis par SQL.

La requête stockée dans la variable **stSql** est maintenant exécutée et son résultat enregistré dans **oResultat**.

```
oSourceDonnees = ThisComponent.Parent.CurrentController
If Not (oSourceDonnees.isConnected()) Then
    oSourceDonnees.connect()
End If
oConnexion = oSourceDonnees.ActiveConnection()
oSQL_Commande = oConnexion.createStatement()
oResultat = oSQL_Commande.executeQuery(stSql)
```

Le résultat de la requête est lu dans une boucle. Comme pour une requête dans l'interface graphique, plusieurs champs et enregistrements peuvent être affichés. Mais la construction de cette requête ne nécessite qu'un seul résultat, qui se trouvera dans la première colonne (1) du jeu de résultats de la requête. C'est l'enregistrement qui fournit le contenu affiché de la zone combinée. Le contenu est du texte (**getString()**), d'où la commande **oResultat.getString(1)**.

```
While oResultat.next
    stValeurChamp = oResultat.getString(1)
Wend
```

La zone combinée doit maintenant être définie sur la valeur de texte récupérée par la requête.

```
oChampListe.Text = stValeurChamp
Else
```

S'il n'y a pas de valeur dans le champ de la clé étrangère **oChamp**, la requête a échoué et la zone de liste déroulante est définie sur une chaîne vide.

```
    oChampListe.Text = ""
End If
Next inZone
End Sub
```

Cette procédure gère le contact entre la zone combinée et la clé étrangère disponible dans un champ de la source de données du formulaire. Cela devrait être suffisant pour afficher les valeurs correctes dans les zones combinées. Le stockage de nouvelles valeurs nécessiterait une procédure supplémentaire.

Transférer une valeur de clé étrangère d'une zone combinée vers un champ numérique

Si une nouvelle valeur est entrée dans la zone de liste déroulante (et c'est après tout le but pour lequel cette macro a été construite), la clé primaire correspondante doit être entrée dans la table sous-jacente du formulaire en tant que clé étrangère.

```
Sub EnregistrerValeurTexteSelection(oEvent As Object)
```

Cette macro doit être liée à l'événement de formulaire suivant : "Avant l'action d'enregistrement".

Une fois que les variables ont été déclarées (non affichées ici), nous devons d'abord déterminer exactement quel événement doit lancer la macro. Avant l'action d'enregistrement, deux implémentations sont appelées successivement. Il est important que la macro elle-même récupère l'objet de formulaire. Cela peut être fait dans les deux implémentations mais de différentes manières. Ici, l'implémentation appelée *ODatabaseForm* est filtrée.

```
    If InStr(oEvent.Source.ImplementationName,"ODatabaseForm") Then
    ...
End If
End Sub
```

Cette boucle se construit avec le même début que la procédure **AfficheTexte** :

```
oFormulaire = oEvent.Source
aZonesListe() = Split(oFormulaire.getByName("ZonesCombinees").Tag, ",")
For inZone = LBound(aZonesListe) To Ubound(aZonesListe)
    ...
Next inZone
```

Le champ **oChampListe** affiche le texte. Il peut se trouver à l'intérieur d'un champ table, auquel cas il n'est pas possible d'y accéder directement depuis le formulaire. Dans de tels cas, les informations supplémentaires pour les zones combinées de contrôle masqué doivent contenir le chemin d'accès au champ à l'aide de la zone de liste déroulante du "ControleTable". Le fractionnement de cette entrée révélera comment accéder à la zone de liste déroulante.

```
a() = Split(Trim(aZonesListe(inZone)), ">")
```

```

If Ubound(a) > 0 Then
    oChampListe = oFormulaire.getByName(a(0)).getByName(a(1))
Else
    oChampListe = oFormulaire.getByName(a(0))
End If

```

Ensuite, la requête est lue à partir de la zone de liste déroulante et divisée en ses parties individuelles. Pour les combobox simples, les informations nécessaires sont le nom du champ et le nom de la table :

```
SELECT "Champ" FROM "Table"
```

Cela pourrait dans certains cas être complété par une instruction de tri. Chaque fois que deux champs doivent être réunis dans la zone de liste déroulante, plus de travail sera nécessaire pour les séparer.

```
SELECT "Champ1" || ' ' || "Champ2" FROM "Table"
```

Cette requête associe deux champs avec un espace entre eux. Comme le séparateur est un espace, la macro le recherchera et divisera le texte en deux parties en conséquence. Naturellement, cela ne fonctionnera de manière fiable que si Champ1 ne contient pas déjà du texte dans lequel les espaces sont autorisés. Sinon, si le prénom est "Anne Marie" et le nom de famille "Müller", la macro traitera "Anne" comme prénom et "Marie Müller" comme nom de famille. Dans de tels cas, un séparateur plus approprié doit être utilisé, qui peut ensuite être trouvé par la macro. Dans le cas des noms, cela peut être "Nom, Prénom".

Les choses se compliquent encore si les deux champs proviennent de tables différentes :

```

SELECT "Table1"."Champ1" || ' > ' || "Table2"."Champ2"
FROM "Table1", "Table2"
WHERE "Table1"."ID" = "Table2"."ID_Etrangère"
ORDER BY "Table1"."Champ1" || ' > ' || "Table2"."Champ2" ASC

```

Ici, les champs doivent être séparés les uns des autres, la table à laquelle appartient chaque champ doit être établie et les clés étrangères correspondantes déterminées.

```

stRequete = oChampListe.ListSource
aChamps() = Split(stRequete, " ")
stContenu = ""
For i=LBound(aChamps)+1 To UBound(aChamps)

```

Le contenu de la requête est dépourvu de scories inutile. Les pièces sont réassemblées dans un tableau avec une combinaison de caractères inhabituelle comme séparateur. FROM sépare l'affichage du champ visible des noms de table. WHERE sépare la condition des noms de table. Les jointures ne sont pas prises en charge.

```

If Trim(UCASE(aChamps(i))) = "ORDER BY" Then
    Exit For
ElseIf Trim(UCASE(aChamps(i))) = "FROM" Then
    stContenu = stContenu + " §§ "
ElseIf Trim(UCASE(aChamps(i))) = "WHERE" Then
    stContenu = stContenu + " §§ "
Else
    stContenu = stContenu + Trim(aChamps(i))
End If
Next i
acontenu() = Split(stContenu, " §§ ")

```

Dans certains cas, le contenu de l'affichage du champ visible provient de différents champs :

```

aPremier() = Split(aContenu(0), "||")
If Ubound(aPremier) > 0 Then
    If Ubound(aContenu) > 1 Then

```

La première partie contient au moins deux champs. Les champs commencent par un nom de table. La deuxième partie contient deux noms de table, qui peuvent être déterminés à partir de la première partie. La troisième partie contient une relation avec une clé étrangère, séparée par = :

```

aTest() = Split(aFirst(0), ".")
NomTable1 = aTest(0)
NomChampTable1 = aTest(1)
Erase aTest
stFieldSeparator = Join(Split(aFirst(1), "'"), ",")
aTest() = Split(aFirst(2), ".")
NomTable2 = aTest(0)
NomChampTable2 = aTest(1)
Erase aTest
aTest() = Split(acontenu(2), "=")
aTest1() = Split(aTest(0), ".")
If aTest1(1) <> "ID" Then
    NomIDTab2 = aTest1(1)
    IF aTest1(0) = NomTable1 Then
        Position = 2
    Else
        Position = 1
    End If
Else
    Erase aTest1
    aTest1() = Split(aTest(1), ".")
    NameTab12ID = aTest1(1)
    If aTest1(0) = NomTable1 Then
        Position = 2
    Else
        Position = 1
    End If
End If
Else

```

La première partie contient deux noms de champs sans noms de table, éventuellement avec des séparateurs. La deuxième partie contient les noms de table. Il n'y a pas de troisième partie :

```

If UBound(aFirst) > 1 Then
    NomChampTable1 = aFirst(0)
    stFieldSeparator = Join(Split(aFirst(1), "'"), ",")
    NomChampTable2 = aFirst(2)
Else
    NomChampTable1 = aFirst(0)
    NomChampTable2 = aFirst(1)
End If
NomTable1 = acontenu(1)
End If
Else

```

Il n'y a qu'un seul champ dans une table :

```

NomChampTable1 = aFirst(0)
NomTable1 = acontenu(1)
End If

```

La longueur de caractère maximale qu'une entrée peut avoir est donnée par la fonction ColumnSize. La zone combinée ne peut pas être utilisée pour limiter la taille, car elle peut avoir besoin de contenir deux champs en même temps.

```

LongueurChamp1 = TailleColonne(NomTable1, NomChampTable1)
If NomChampTable2 <> "" Then
    If NomTable2 <> "" Then
        LongueurChamp2 = TailleColonne(NomTable2, NomChampTable2)
    Else
        LongueurChamp2 = TailleColonne(NomTable1, NomChampTable2)
    End If
Else
    LongueurChamp2 = 0

```

```
End If
```

Le contenu de la combobox est lu :

```
stContenu = oChampListe.getCurrentValue()
```

Les espaces de début et de fin et les caractères non imprimables sont supprimés si nécessaire.

```
stContenu = Trim(stContenu)
If stContenu <> "" Then
    If NomChampTable2 <> "" Then
```

Si un deuxième champ de table existe, le contenu de la zone de liste déroulante doit être fractionné. Pour déterminer où le fractionnement doit se produire, nous utilisons le séparateur de champ fourni à la fonction comme argument.

```
a_stParts = Split(stContenu, FieldSeparator, 2)
```

Le dernier paramètre signifie que le nombre maximum de parties est de 2.

En fonction de l'entrée correspondant au champ 1 et de celle du champ 2, le contenu de la zone de liste déroulante est maintenant alloué aux variables individuelles. "Position = 2" sert ici d'indicateur que la deuxième partie du contenu correspond au champ 2.

```
If Position = 2 Then
    stContenu = Trim(a_stParts(0))
    If UBound(a_stParts()) > 0 Then
        stContenuChamp2 = Trim(a_stParts(1))
    Else
        stContenuChamp2 = ""
    End If
    stContenuChamp2 = Trim(a_stParts(1))
Else
    stContenuChamp2 = Trim(a_stParts(0))
    If UBound(a_stParts()) > 0 Then
        stContenu = Trim(a_stParts(1))
    Else
        stContenu = ""
    End If
    stContenu = Trim(a_stParts(1))
End If
End If
```

Il peut arriver qu'avec deux contenus séparables, la taille installée de la zone de liste déroulante (longueur du texte) ne corresponde pas aux champs du tableau à enregistrer. Pour les zones combinées qui représentent un seul champ, cela est normalement géré en configurant correctement le contrôle de formulaire. Ici, en revanche, nous avons besoin d'un moyen de détecter ces erreurs. La longueur maximale autorisée du champ concerné est vérifiée.

```
If (LongueurChamp1 > 0 And Len(stContenu) > LongueurChamp1) Or _
(LongueurChamp2 > 0 And Len(stContenuChamp2) > LongueurChamp2) Then
```

Si la longueur du champ de la première ou de la deuxième partie est trop grande, une chaîne par défaut est stockée dans l'une des variables. Le caractère **Chr(13)** est utilisé pour insérer un saut de ligne.

```
stmsgbox1 = "Le champ " + NomChampTable1 + " ne peut dépasser " + _
LongueurChamp + "caractères en longueur." + Chr(13)
stmsgbox2 = "Le champ " + NomChampTable2 + " ne peut dépasser " + _
LongueurChamp2 + "caractères en longueur." + Chr(13)
```

Si le contenu des deux champs est trop long, les deux textes sont affichés.

```
If ( LongueurChamp1 > 0 And Len(stContenu) > LongueurChamp1) And _
( LongueurChamp2 > 0 And Len(stContenuChamp2) > LongueurChamp2) Then
    MsgBox("Le texte saisi est trop long." + Chr(13) + stmsgbox1 + _
stmsgbox2 + "Raccourcir SVP.", 64, "Entrée invalide")
```

L'affichage utilise la fonction **MsgBox()**. Cela attend comme premier argument une chaîne de texte, puis éventuellement un nombre (qui détermine le type de boîte de message affiché), et enfin une chaîne de texte optionnelle comme titre de la fenêtre. La fenêtre aura donc le titre "Entrée invalide" et le numéro "64" fournit une boîte contenant le symbole d'information.

Le code suivant couvre tout autre cas de texte excessivement long qui pourrait survenir.

```

ElseIf (LongueurChamp1 > 0 And Len(stContenu) > LongueurChamp) Then
    MsgBox("Le texte saisi est trop long." + Chr(13) + stmsgbox1 + "Raccourcir SVP.", 64, "Entrée invalide")
Else
    MsgBox("Le texte saisi est trop long." + Chr(13) + stmsgbox2 + "Raccourcir SVP.", 64, "Entrée invalide")
End If
Else

```

S'il n'y a pas de texte trop long, la fonction peut continuer. Sinon, elle sort ici.

Désormais, les entrées sont masquées afin que les guillemets éventuellement présents ne génèrent pas d'erreur.

```

stContenu = String_to_SQL(stContenu)
If stContenuChamp2 <> "" Then
    stContenuChamp2 = String_to_SQL(stContenuChamp2)
End If

```

Les premières variables sont préallouées et peuvent ensuite être modifiées par la requête. Les variables inID1 et inID2 stockent le contenu des champs de clé primaire des deux tables. Si une requête ne donne aucun résultat, Basic attribue à cette variable entière une valeur de 0. Cependant, cette valeur peut également indiquer une requête réussie renvoyant une valeur de clé primaire de 0 ; donc la variable est préreglée à -1. HSQLDB ne peut pas définir cette valeur pour un champ de valeur automatique.

Ensuite, la connexion à la base de données est établie, si elle n'existe pas déjà.

```

inID1 = -1
inID2 = -1
oSourceDonnees = ThisComponent.Parent.CurrentController
If Not (oSourceDonnees.isConnected()) Then
    oSourceDonnees.connect()
End If
oConnexion = oSourceDonnees.ActiveConnection()
oSQL_Commande = oConnexion.createStatement()
If NomChampTable2 <> "" And Not IsEmpty(stContenuChamp2) And _
NomTable2 <> "" Then

```

Si un deuxième champ de table existe, une deuxième dépendance doit d'abord être déclarée.

```

stSql = "SELECT ""ID"" FROM "" + NomTable2 + "" WHERE "" + _ NomChampTable2
+ ""="" + stContenuChamp2 + ""
oResultat = oSQL_Commande.executeQuery(stSql)
While oResultat.next
    inID2 = oResultat.getInt(1)
Wend
If inID2 = -1 Then
    stSql = "INSERT INTO "" + NomTable2 + "" ("" + NomChampTable2 + "")
VALUES ('" + stContenuChamp2 + "')"
    oSQL_Commande.executeUpdate(stSql)
    stSql = "CALL IDENTITY()"

```

Si le contenu de la zone de liste n'est pas présent dans le tableau correspondant, il y est inséré. La valeur de la clé primaire qui en résulte est ensuite lue. S'il est présent, la clé primaire existante est lue de la même manière. La fonction utilise les champs de clé primaire générés automatiquement (**IDENTITY**).

```

oResultat = oSQL_Commande.executeQuery(stSql)
While oResultat.next

```

```

        inID2 = oResultat.getInt(1)
    Wend
End If

```

La clé primaire de la deuxième valeur est temporairement stockée dans la variable **inID2**, puis écrite en tant que clé étrangère dans la table correspondant à la première valeur. Selon que l'enregistrement de la première table était déjà disponible, le contenu est nouvellement enregistré (**INSERT**) ou modifié (**UPDATE**) :

```

If inID1 = -1 Then
    stSql = "INSERT INTO " + NomTable1 + " (" + NomChampTable1 +
    ", " + NomIDTab2 + ") VALUES ('" + stContent + "', '" + inID2 + "')"
    oSQL_Commande.executeUpdate(stSql)

```

Et l'ID correspondant directement lu :

```

    stSql = "CALL IDENTITY()"
    oResultat = oSQL_Commande.executeQuery(stSql)
    While oResultat.next
        inID1 = oResultat.getInt(1)
    Wend

```

La clé primaire de la première table doit enfin être lue à nouveau pour pouvoir être transférée vers la table sous-jacente du formulaire.

```

Else
    stSql = "UPDATE " + NomTable1 + " SET " + NameTab12ID + _
    "='" + inID2 + "' WHERE " + NomChampTable1 + " = '" + stContent + "'"
    oSQL_Commande.executeUpdate(stSql)
End If
End If

```

Dans le cas où les deux champs sous-jacents à la zone combinée sont dans la même table (par exemple Nom et Prénom dans la table Noms), une requête différente est nécessaire :

```

If NomChampTable2 <> "" And NomTable2 = "" Then
    stSql = "SELECT "ID" FROM " + NomTable1 + " WHERE " + _
    NomChampTable1 + "='" + stContent + "' AND " + NomChampTable2 + _
    "='" + stContentField2 + "'"
    oResultat = oSQL_Commande.executeQuery(stSql)
    While oResultat.next
        inID1 = oResultat.getInt(1)
    Wend
    If inID1 = -1 Then

```

... et une deuxième table n'existe pas :

```

        stSql = "INSERT INTO " + NomTable1 + " (" + NomChampTable1 + ", " +
        NomChampTable2 + ") VALUES ('" + stContent + "', '" + stContentField2 + "')"
        oSQL_Commande.executeUpdate(stSql)

```

Ensuite, la clé primaire est à nouveau lue.

```

        stSql = "CALL IDENTITY()"
        oResultat = oSQL_Commande.executeQuery(stSql)
        While oResultat.next
            inID1 = oResultat.getInt(1)
        Wend
    End If
End If
If NomChampTable2 = "" Then

```

Considérons maintenant le cas le plus simple : le deuxième champ de table n'existe pas et l'entrée n'est pas encore présente dans la table. En d'autres termes, une seule nouvelle valeur a été entrée dans la zone de liste déroulante.

```

    stSql = "SELECT "ID" FROM " + NomTable1 + " WHERE " + NomChampTable1 +
    "='" + stContent + "'"
    oResultat = oSQL_Commande.executeQuery(stSql)

```

```

While oResultat.next
    inID1 = oResultat.getInt(1)
Wend
If inID1 = -1 Then

```

S'il n'y a pas de deuxième champ, le contenu de la zone est inséré en tant que nouvel enregistrement.

```

    stSql = "INSERT INTO "" + NomTable1 + "" ("" + NomChampTable1 + "")
VALUES ('" + stContenu + "') "
    oSQL_Commande.executeUpdate(stSql)

```

... Et l'ID qui en résulte est lu directement.

```

    stSql = "CALL IDENTITY()"
    oResultat = oSQL_Commande.executeQuery(stSql)
    While oResultat.next
        inID1 = oResultat.getInt(1)
    Wend
End If
End If

```

La valeur du champ de clé primaire doit être déterminée afin de pouvoir être transférée vers la partie principale du formulaire.

Ensuite, la valeur de clé primaire résultant de toutes ces boucles est transférée dans le champ invisible de la table principale et de la base de données sous-jacente. Le champ de table lié au champ de formulaire est atteint en utilisant "BoundField". **"updateInt"** place un entier (voir sous les définitions de type numérique) dans ce champ.

```

    oFormulaire.updateLong(oFormulaire.findColumn(oChampListe.Tag), inID1)
End If
ELSE

```

Si aucune clé primaire ne doit être saisie, parce qu'il n'y avait aucune entrée dans la zone de liste déroulante ou que cette entrée a été supprimée, le contenu du champ invisible doit également être supprimé. `updateNull()` est utilisé pour remplir le champ avec l'expression spécifique à la base de données pour un champ vide, NULL.

```

    oFormulaire.updateNULL(oFormulaire.findColumn(oChampListe.Tag), NULL)
End If
NEXT inZone
End If
End Sub

```

Fonction pour mesurer la longueur de l'entrée zone combinée

La fonction suivante donne le nombre de caractères dans la colonne de table respective, de sorte que les entrées trop longues ne soient pas simplement tronquées. Une **Function** est choisie ici pour fournir des valeurs de retour. Un **SUB** n'a pas de valeur de retour qui peut être transmise et traitée ailleurs.

```

Function TailleColonne(Tablename As String, NomChamp As String) As Integer
    oSourceDonnees = ThisComponent.Parent.CurrentController
    If Not (oSourceDonnees.isConnected()) Then
        oSourceDonnees.connect()
    End If
    oConnexion = oSourceDonnees.ActiveConnection()
    oSQL_Commande = oConnexion.createStatement()
    stSql = "SELECT ""COLUMN_SIZE"" FROM ""INFORMATION_SCHEMA"". ""SYSTEM_COLUMNS""
WHERE ""TABLE_NAME"" = '" + Tablename + "' AND ""COLUMN_NAME"" = '" + NomChamp + "'"
    oResultat = oSQL_Commande.executeQuery(stSql)
    While oResultat.next
        i = oResultat.getInt(1)
    Wend
    TailleColonne = i
End Function

```

Générer des actions de base de données

```
Sub GenereActionEnregistrement(oEvent As Object)
```

Cette macro doit être liée à l'événement *Texte modifié* de la zone de liste. Il est nécessaire que dans tous les cas où la zone de liste est modifiée, la modification soit enregistrée. Sans cette macro, il n'y aurait aucun changement dans la table réelle que Base pourrait reconnaître, car la zone de liste déroulante n'est pas liée au formulaire.

Cette macro modifie directement les propriétés du formulaire :

```
Dim oFormulaire As Object
oFormulaire = oEvent.Source.Model.Parent
oFormulaire.IsModified = TRUE
End Sub
```

Cette macro n'est pas nécessaire pour les formulaires qui utilisent des requêtes pour le contenu des zones combinées. Les changements dans les zones combinées sont enregistrés directement.

Navigation d'un formulaire à un autre

Dans un formulaire lorsqu'un événement particulier se produit, un second formulaire peut être ouvert¹⁴. Dans les propriétés du contrôle de formulaire, sur la ligne « Complément d'information » (balise), entrez le nom du second formulaire suivi du nom du premier, séparés par une virgule (ex : Form_2, Form_1). Dans l'exemple utilisé ceci est réalisé dans les propriétés de deux boutons.

De plus amples informations peuvent également être saisies ici, puis séparées à l'aide de la fonction **Split()**.

```
Sub De_formulaire_a_formulaire(oEvent As Object)
Dim stTag As String
stTag = oEvent.Source.Model.Tag
aFormulaire() = Split(stTag, ",")
```

Le tableau est déclaré et rempli avec les noms des formulaires, d'abord celui du formulaire à ouvrir et deuxièmement celui du formulaire actuel, qui sera fermé après l'ouverture de l'autre.

```
ThisDatabaseDocument.FormDocuments.getByName(Trim(aFormulaire(0))).open
ThisDatabaseDocument.FormDocuments.getByName(Trim(aFormulaire(1))).close
End Sub
```

Si au contraire, le formulaire ne doit être ouvert que lorsque le formulaire actuel est fermé, par exemple lorsqu'un formulaire principal existe et que tous les autres formulaires sont contrôlés à partir de celui-ci à l'aide de boutons, la macro suivante doit être liée au formulaire avec **Outils> Personnaliser> Événements> Document fermé** :

```
Sub Ouvre_Formulaire_Principal
ThisDatabaseDocument.FormDocuments.getByName("Formulaire").open
End Sub
```

Si les documents de formulaire sont triés dans le fichier ODB dans des dossiers, la macro de changement de formulaire doit être plus étendue :

```
Sub De_Formulaire_a_Formulaire_avec_dossiers(oEvent As Object)
REM Le formulaire à ouvrir est donné en premier.
REM Si un formulaire se trouve dans un dossier, utilisez "/" pour définir
REM la relation afin que le sous-dossier puisse être trouvé.
Dim stTag As String
stTag = oEvent.Source.Model.Tag 'La balise est entrée dans les informations
complémentaires
aFormulaires() = Split(stTag, ",") 'Ici, le nom du nouveau formulaire vient en
premier, puis celui de l'ancien formulaire
aFormulaires1() = Split(aFormulaires(0), "/")
```

14 Voir la base Exemple_Formulaire_a_formulaire.odb, associée à ce manuel. Des usages sont visibles également dans la base Media_avec_Macros.

```

aFormulaires2() = Split(aFormulaires(1), "/")
If UBound(aFormulaires1()) = 0 Then
    ThisDatabaseDocument.FormDocuments.getByName(Trim(aFormulaires1(0))).open
Else
    ThisDatabaseDocument.FormDocuments.getByName(Trim(aFormulaires1(0)))._
    getByName(Trim(aFormulaires1(1))).open
End If
If UBound(aFormulaires2()) = 0 Then
    ThisDatabaseDocument.FormDocuments.getByName _
    (Trim(aFormulaires2(0))).close
Else
    ThisDatabaseDocument.FormDocuments.getByName _
    (Trim(aFormulaires2(0))).getByName(Trim(aFormulaires2(1))).close
End If
End Sub

```

Les documents de formulaire qui se trouvent dans un dossier sont saisis dans le champ Informations supplémentaires en tant que dossier/formulaire. Cela doit être converti en :

```
...getByName("Dossier").getByName("Formulaire").
```

Ouvrir des rapports, tableaux, requêtes depuis un formulaire

Comme dans le chapitre précédent, les rapports, eux aussi, peuvent être ouverts depuis un formulaire. Tout comme les formulaires, les rapports sont des documents distincts inclus dans le dossier de base. Au lieu de **FormDocuments**, **ReportDocuments** doit être utilisé ici. En outre, il faut noter que les formulaires et les rapports peuvent être localisés dans des sous-répertoires. Cependant, il est plus difficile d'accéder aux tableaux, requêtes et vues, car ils ne sont pas disponibles sous forme de documents séparés¹⁵.

```

Subouvre_Element(oEvent As Object)
    Dim stTag As String
    Dim inType As Integer
    Dim aOuvrir()
    Dim oDoc As Object
    stTag = oEvent.Source.Model.Tag
    aOuvrir() = Split(stTag, ",")
    Select Case Trim(aOuvrir(0))
        Case "Formulaire", "Rapport"
            Rem Formulaires et rapports peuvent être
            Rem localisés dans des sous-répertoires.
            aForms1() = Split(Trim(aOuvrir(1)), "/")
            If Trim(aOuvrir(0)) = "Formulaire" Then
                oDoc = ThisDatabaseDocument.FormDocuments
            Else
                oDoc = ThisDatabaseDocument.ReportDocuments
            End If
            If UBound(aForms1()) > 0 Then
                oDoc.getByName(Trim(aForms1(0))).getByName(Trim(aForms1(1))).open
            Else
                oDoc.getByName(Trim(aForms1(0))).open
            End If
            If Trim(aOuvrir(0)) = "Formulaire" And UBound(aOuvrir()) > 1 Then
                Rem Le formulaire qui lance la macro pourrait également être fermé
                aForms2() = Split(Trim(aOuvrir(2)), "/")
                If UBound(aForms2()) > 0 Then
                    ThisDatabaseDocument.FormDocuments.getByName(Trim(aForms2(0))). _
                    getByName(Trim(aForms2(1))).close
                Else
                    ThisDatabaseDocument.FormDocuments.getByName(Trim(aForms2(0))).close
                End If
            End If
            Exit Sub
        Case "Requête"
            inType = 1
    End Select
End Sub

```

15 Voir la base de données Exemple_Formulaire_a_formulaire.odt associée à ce manuel.

```

        Open_Table_Query_View(Trim(aOuvrir(1)), inType)
    Case "Table"
        inType = 0
        Open_Table_Query_View(Trim(aOuvrir(1)), inType)
    End Select
End Sub

```

Dans le formulaire de base (dans l'exemple Formulaire_Menu) les informations nécessaires sont lues dans les **propriétés** du bouton utilisé, onglet **Général**.

Deux termes sont utilisés, le premier pour indiquer que l'on désire ouvrir une Table (ou un Formulaire, une Requête, un Rapport), le second pour donner le nom de l'objet à ouvrir (Exemple : Table, Adresses).

Ces deux informations sont récupérées dans **oTag**, puis placées dans un tableau **aOuvrir**. **Trim** est utilisé pour supprimer tous les espaces superflus devant et après les expressions.

Si le premier terme est « **Formulaire** » ou « **Rapport** », le formulaire ou le rapport nommé dans le second terme est ouvert.

Pour les requêtes et les tables (ou vues), il faut adopter une approche différente.

Une variable (**inType**) détermine s'il s'agit d'une **Requête** (1), d'une **Table** ou d'une **Vue** (0), puis la sous-procédure suivante est lancée, avec pour paramètres : **aOuvrir(1)** – le nom de l'élément à ouvrir, et **inType**.

```

SUB Ouvre_Form_Rapp_Tab_Vue_Reg(stNom AS STRING, inType AS INTEGER)
    Rem Form pour Formulaire - Rapp pour Rapport
    Rem Tabl pour Table - Vue - Req pour Requête
    DIM oContrôleur AS OBJECT
    DIM oConnexion AS OBJECT
    oContrôleur = ThisDatabasedocument.CurrentController
    IF NOT oContrôleur.isconnected THEN
        oContrôleur.connect
    END IF
    oConnexion = oContrôleur.ActiveConnection
    DIM URL AS NEW com.sun.star.util.URL
    DIM Args(5) AS NEW com.sun.star.beans.PropertyValue
    URL.Complete = ".component:DB/DataSourceBrowser"
    Dispatch = StarDesktop.queryDispatch(URL, "_Blank", 8)
    Args(0).Name = "ActiveConnection"
    Args(0).Value = oConnexion
    Args(1).Name = "CommandType"
    Args(1).Value = inType '0=Table 1=Requête_SQL 2=Commande
    Args(2).Name = "Command"
    Args(2).Value = stNom
    Args(3).Name = "ShowMenu"
    Args(3).Value = True
    Args(4).Name = "ShowTreeView"
    Args(4).Value = False
    Args(5).Name = "ShowTreeViewButton"
    Args(5).Value = False
    Dispatch.dispatch(URL, Args)
END SUB

```

Tout d'abord, la connexion à la base de données est établie, si elle n'existe pas encore. Cette connexion est transmise dans un tableau avec quelques informations supplémentaires, notamment le type d'élément à ouvrir (table, requête, ...) et le nom de l'élément.

L'élément est finalement ouvert via le **StarDesktop.queryDispatch** avec la commande **dispatch**.

Zones de liste hiérarchiques

Les paramètres d'un champ de liste sont destinés à influencer directement les paramètres d'un autre. Pour les cas simples, cela a déjà été décrit ci-dessus dans la section sur le filtrage des enregistrements. Mais en supposant que la première zone de liste est censée affecter le contenu de la deuxième, qui affecte alors le contenu d'une troisième zone de liste, et ainsi de suite¹⁶.

Niveau	Classe	Noms
1	a	Anna Logis
2	b	Alain Dissoir
3	c	Adele Scott
4	d	Ivan Tilelapièce
5	e	Vladimir Pourlavessel
6	f	
7	g	
8		
9		
10		
11		
12		
13		

Exemples de zones de liste pour une sélection hiérarchique

Dans cet exemple, la première zone de liste (Niveau) contient toutes les années scolaires. Les Classes de chaque niveau sont représentées par des lettres. Les noms sont ceux des membres de la classe.

Dans des circonstances normales, la liste des niveaux affiche les 13 années, la liste des classes toutes les lettres des classes et la liste des noms tous les élèves de l'école.

S'il s'agit de listes hiérarchiques, le choix des classes est restreint une fois que le niveau est choisi. Seules les lettres de classe qui sont effectivement présentes à ce niveau-là sont affichées. Cela peut varier car, si le nombre d'élèves augmente, le nombre de classes par niveau peut également augmenter. La dernière zone de liste, Noms, est très restreinte. Au lieu de plus de 1000 élèves, il n'en afficherait que 30.

Au début, seul le niveau peut être sélectionné. Une fois cela fait, la liste (restreinte) des classes est mise à disposition. Ce n'est qu'à la fin que la liste des noms est donnée.

Si la zone de liste Niveaux est modifiée, la séquence doit recommencer. Si seule la zone de liste Classes est modifiée, le numéro de niveau de la dernière zone de liste reste valide

Pour créer une telle fonction, le formulaire doit pouvoir stocker une variable intermédiaire. Cela a lieu dans un contrôle caché.

¹⁶ Un exemple de contrôles hiérarchiques se trouve dans la base Exemple_Cherche_et_Filtre.odt

La macro est liée à une modification du contenu d'une zone de liste : **Propriétés Zone de liste> Événements> Modifiée**. Les variables nécessaires sont stockées dans le complément d'information de la zone de liste.

Voici un exemple d'informations complémentaires fournies:
Formulaire, **Niveau**, Contrôle_masqué, **ZoneListe_2**

Le formulaire s'appelle Formulaire. La Zone de liste actuelle est appelée ZoneListe_1. Cette zone de liste affiche le contenu du champ de table Niveau et les zones de liste suivantes doivent être filtrées en fonction de cette entrée. Le contrôle masqué est désigné par Controle_Masque et l'existence d'une deuxième zone de liste (ZoneListe_2) est transmise à la procédure de filtrage.

```
Sub Controles_Hierarchiques(oEvent As Object)
    Dim oDoc As Object
    Dim oDrawpage As Object
    Dim oFormulaire As Object
    Dim oChampCache As Object
    Dim oChamp As Object
    Dim oChamp1 As Object
    Dim stSql As String
    Dim acontenu()
    Dim stTag As String
    oChamp = oEvent.Source.Model ' la zone de liste origine de l'événement
    stTag = oChamp.Tag
    oFormulaire = oChamp.Parent
    REM La balise entre dans le champ Complément d'informations
    REM Elle contient :
    REM 0. Nom du champ du champ à filtrer dans la table
    REM 1. Nom de champ du contrôle masqué qui stockera la valeur filtrée
    REM 2. Zone de liste supplémentaire éventuelle
    REM La balise est lue à partir de l'élément qui lance la macro. La variable est
    REM passée à la procédure, et si nécessaire à toutes les autres zones de liste.
    aFiltre() = Split(stTag, ",")
    stFiltre = ""
```

Une fois que les variables ont été déclarées, le contenu de la balise est passé (découpée) à un tableau, afin que les éléments individuels soient accessibles. Ensuite, l'accès aux différents champs du formulaire est déclaré.

La zone de liste qui a appelé la macro est déterminée et sa valeur lue. Ce n'est que si cette valeur n'est pas NULL qu'elle sera combinée avec le nom du champ à filtrer, dans notre exemple Niveau, pour créer une commande SQL. Sinon, le filtre restera vide. Si les zones de liste sont destinées à filtrer un formulaire, un contrôle masqué n'est pas utile. Dans ce cas, la valeur du filtre est stockée directement dans le formulaire.

```
        If Trim(aFiltre(1)) = "" Then

                                If oChamp.GetCurrentValue <> "" Then

                                        stFiltre = """"+Trim(aFiltre(0))

+""""=""+oChamp.GetCurrentValue()+"""
```

Si un filtre existe déjà (par exemple celui traitant de ZoneListe_2, qui est en cours d'accès), le nouveau contenu est attaché au contenu précédent stocké dans le champ masqué.

```
        If oFormulaire.Filter <> ""
```

Cela ne doit se produire que lorsque le même champ n'a pas encore été filtré. Par exemple, si nous filtrons par Niveau, une répétition du filtre ne trouvera aucun enregistrement supplémentaire pour la zone de liste Nom. Une personne ne peut être trouvée que dans un niveau. Il faut donc exclure la possibilité que le nom du filtre ait déjà été utilisé.

```
        And InStr(oFormulaire.Filter, """"+Trim(aFiltre(0))+""""="") = 0 Then
            stFiltre = oFormulaire.Filter + " AND " + stFiltre
```

Si un filtre existe et que le champ qui sera utilisé pour le filtrage est déjà présent dans le filtre, le filtrage précédent sur ce nom de champ doit être supprimé et un nouveau filtre créé.

```

ElseIf oForm.Filter <> "" Then
    stFiltre = Left(oFormulaire.Filter,
        InStr(oFormulaire.Filter, """"+Trim(aFiltre(0))+""""=')-1) + _
        stFiltre
End If
End If

```

Ensuite, le filtre est entré dans le formulaire. Ce filtre peut également être vide si la première zone de liste a été sélectionnée et n'a pas de contenu.

```

oFormulaire.Filter = stFiltre
oFormulaire.reload()

```

La même procédure s'exécutera si le formulaire n'a pas besoin d'être filtré immédiatement. Dans ce cas, la valeur du filtre est stockée dans l'intervalle de temps dans un champ masqué.

```

Else
    oChampCache = oFormulaire.getByName(Trim(aFiltre(1)))
    If oChamp.getCurrentValue <> "" Then
        stFiltre = """"+Trim(aFiltre(0))+""""=')+oChamp.getCurrentValue()+""
        If oChampCache.HiddenValue <> ""
            And InStr(oChampCache.HiddenValue, """"+Trim(aFiltre(0))+""""=') = 0 Then
                stFiltre = oChampCache.HiddenValue + " AND " + stFiltre
            ElseIf oChampCache.HiddenValue <> "" Then
                stFiltre = Left(oChampCache.HiddenValue,
                    InStr(oChampCache.HiddenValue, """"+Trim(aFiltre(0))+""""=')-1) + _
                    stFiltre
            End If
        End If
        oChampCache.HiddenValue = stFiltre
    End If
End If

```

Si les informations supplémentaires ont une entrée numérotée 4 (la numérotation commence à 0), la zone de liste suivante doit être réglée sur l'entrée correspondante de la zone de liste origine de l'appel.

```

If UBound(aFiltre()) > 1 Then
    oChamp1 = oFormulaire.getByName(Trim(aFiltre(2)))
    aFiltre1() = Split(oChamp1.Tag, ",")

```

Les données nécessaires au filtrage sont lues à partir des informations supplémentaires (étiquette) dans la zone de liste correspondante. Malheureusement, il n'est pas possible d'écrire uniquement le nouveau code SQL dans la zone de liste, puis de lire les valeurs de la zone de liste. Au lieu de cela, les valeurs correspondant à la requête doivent être écrites directement dans la zone de liste.

La création du code part du fait que la table à laquelle le formulaire se réfère est la même que celle à laquelle se réfèrent les zones de liste. Une telle zone de liste n'est pas conçue pour transférer des clés étrangères vers la table.

```

If oChamp.getCurrentValue <> "" Then
    stSql = "SELECT DISTINCT """"+Trim(aFiltre1(0))+"""" FROM """"+ _
        oFormulaire.Command+ """" WHERE "+stFiltre+" ORDER BY """"+ _
        Trim(aFiltre1(0))+""""
    oSourceDonnees = ThisComponent.Parent.CurrentController
    If Not (oSourceDonnees.isConnected()) Then
        oSourceDonnees.connect()
    End If
    oConnexion = oSourceDonnees.ActiveConnection()
    oSQL_Instruction = oConnexion.createStatement()
    oResultat_Requete = oSQL_Instruction.executeQuery(stSql)

```

Les valeurs sont lues dans un tableau. Le tableau est transféré directement dans la zone de liste. Les indices correspondants pour le tableau sont incrémentés dans une boucle.

```

inIndex = 0
While oResultat_Requete.next
    ReDim Preserve acontenu(inIndex)
    acontenu(inIndex) = oResultat_Requete.getString(1)
    inIndex = inIndex+1
Wend
Else
    acontenu(0) = ""
End If
oChamp1.StringItemList = acontenu()

```

Le contenu de la zone de liste a été recréé. Maintenant, il doit être relu. Ensuite, à l'aide de la propriété "*Complément d'information*" de la zone de liste qui a été actualisée, chacune des zones de liste dépendantes qui suivent est vidée, lançant une boucle pour toutes les zones de liste suivantes jusqu'à ce qu'une zone soit atteinte qui n'a pas de quatrième terme dans son Complément d'information.

```

oChamp1.refresh()
While UBound(aFiltre1()) > 1
    Dim aVide()
    oChamp2 = oFormulaire.getByName(Trim(aFiltre1(2)))
    Dim aFiltre1()
    aFiltre1() = Split(oChamp2.Tag, ",")
    oChamp2.StringItemList = aVide()
    oChamp2.refresh()
Wend
End If
End Sub

```

Le contenu visible des zones de liste est stocké dans `oChamp1.StringItemList`. Si une valeur supplémentaire doit être stockée pour être transmise à la table sous-jacente en tant que clé étrangère, comme d'habitude pour les zones de liste dans les formulaires, cette valeur doit être transmise à la requête séparément, puis stockée avec `oChamp1.ValueItemList`.

Une telle extension nécessite des variables supplémentaires telles que, en plus de la table dans laquelle les valeurs du formulaire doivent être stockées, la table à partir de laquelle est tiré le contenu de la zone de liste.

Une attention particulière doit être portée à la formulation du filtre.

```
stFiltre = """"+Trim(aFiltre(1))+""""=''+oChamp.getCurrentValue()+"""
```

Ceci ne fonctionnera que si la version sous-jacente de LibreOffice est 4.1 ou ultérieure, car c'est la valeur à stocker qui est donnée comme `CurrentValue()`, et non la valeur affichée. Pour vous assurer qu'il fonctionne dans différentes versions, définissez Propriétés : Zone de liste> Données> Champ lié> '0'.

Saisie des temps en millisecondes

Afin de stocker les temps avec une précision de l'ordre de la milliseconde, il faut un champ d'horodatage dans la table, adapté séparément par une requête SQL à cet effet (voir « Création de table » – « Clés primaires [trois derniers paragraphes] au chapitre 3). Un tel champ peut être représenté sur un formulaire par un champ formaté au format **MM:SS,00**. Cependant, lors de la première tentative d'écriture, l'entrée d'enregistrement échouera. Cela peut être corrigé avec la macro suivante, qui doit être liée à la propriété « Avant l'action d'enregistrement » du formulaire :

```

SUB Horodatage
    Dim unoStmp As New com.sun.star.util.DateTime
    Dim oDoc As Object
    Dim oDrawpage As Object
    Dim oFormulaire As Object
    Dim oChamp As Object
    Dim stTemps As String
    Dim arMinEtSec()

```

```

Dim arMandS()
Dim loNano As Long
Dim inSecond As Integer
Dim inMinute As Integer
oDoc = thisComponent
oDrawpage = oDoc.Drawpage
oFormulaire = oDrawpage.Forms.getByName("Formulaire")
oChamp = oFormulaire.getByName("Temps")
stTemps = oChamp.Text

```

Les variables sont déclarées en premier. Le reste du code est exécuté uniquement lorsque le champ de temps contient quelque chose. Sinon, le mécanisme interne du formulaire agira pour définir le champ sur **NULL**.

```

If stTemps <> "" Then
    arMinEtSec() = Split(stTemps, ".")
    loNano = CLng(ar(1) & "00000000")
    arMinEtSec() = Split(ar(0), ":")
    inSeconde = CInt(arMinEtSec(1))
    inMinute = CInt(arMinEtSec(0))

```

L'entrée dans le champ temps est décomposée en ses éléments.

Tout d'abord, la partie décimale est séparée et complétée à droite avec des caractères nuls jusqu'à un total de neuf chiffres. Un nombre aussi élevé ne peut être stocké que dans une variable Long.

Ensuite, le reste du temps est divisé en minutes et secondes, en utilisant les deux points comme séparateur, et ceux-ci sont convertis en nombres entiers.

```

With unoStmp
    .NanoSeconds = loNano
    .Seconds = inSecond
    .Minutes = inMinute
    .Hours = 0
    .Day = 30 ' jour du mois
    .Month = 12
    .Year = 1899
End With

```

Les valeurs d'horodatage sont attribuées à la date standard de LibreOffice du 30/12/1899. Bien sûr, la date actuelle réelle peut être stockée à côté.

```

    oChamp.BindField.updateTimestamp(unoStmp)
End If
End Sub

```

Maintenant, l'horodatage que nous avons créé est transféré dans le champ à l'aide de **updateTimestamp** et stocké dans le formulaire.



Note

Obtenir et stocker la date actuelle :

```

Dim Maintenant As Date
Maintenant = Now()
With unoStmp
    .NanoSeconds = loNano
    .Seconds = inSecond
    .Minutes = inMinute
    .Hours = Hour(now)
    .Day = Day(now)
    .Month = Month(now)
    .Year = Year(now)
End With

```

Dans les didacticiels précédents (notamment Apache Open Office), les **NanoSeconds** étaient appelées **HundrethSeconds**. Cela ne correspond pas à l'API LibreOffice et provoquera un message d'erreur.

Un événement – plusieurs implémentations

Lors de l'utilisation de formulaires, une macro liée à un seul événement peut être exécutée deux fois. Cela se produit parce que plusieurs processus sont liés simultanément, par exemple, au stockage d'un enregistrement modifié. Les différentes causes d'un tel événement peuvent être déterminées de la manière suivante :

```
Sub Determine_CauseEvenement(oEvent As Object)
    Dim oFormulaire As Object
    oFormulaire = oEvent.Source
    MsgBox oFormulaire.ImplementationName
End Sub
```

Lorsqu'un enregistrement modifié est stocké, deux implémentations sont impliquées, nommées **org.openoffice.comp.svx. FormController** et **com.sun.star.comp.forms. ODatabaseForm**. En utilisant ces noms, nous pouvons nous assurer qu'une macro n'exécute son code qu'une seule fois. Une exécution en double provoque généralement juste une (petite) pause dans l'exécution du programme, mais cela peut conduire à ce qu'un curseur soit reculé de deux enregistrements au lieu d'un. Chaque implémentation n'autorise que des commandes spécifiques, il peut donc être important de connaître le nom de l'implémentation.

Enregistrer avec confirmation

Pour les modifications d'enregistrements complexes, il est logique de demander à l'utilisateur avant l'exécution si la modification doit réellement être effectuée. Si la réponse dans la boîte de dialogue est « Non », la sauvegarde est abandonnée, la modification est annulée et le curseur reste sur l'enregistrement en cours.

```
Sub Confirmation_Enregistre(oEvent As Object)
    Dim oFonctionFormulaire As Object
    Dim oOperationsFormulaire As Object
    Dim inReponse As Integer
    oFonctionFormulaire = com.sun.star.form.runtime.FormFeature
    Select Case oEvent.Source.ImplementationName
        Case "org.openoffice.comp.svx.FormController"
            inReponse = MsgBox("Faut-il modifier l'enregistrement ?", 4, "Modification _
enregistrement")
            Select Case inReponse
                Case 6 ' Oui, aucune autre action
                Case 7 ' Non, interrompre l'enregistrement
                    oOperationsFormulaire = oEvent.Source.FormOperations
                    oOperationsFormulaire.execute(oFonctionFormulaire.UndoRecordChanges)
            Case Else
            End Select
        Case "com.sun.star.comp.forms.ODatabaseForm"
    End Select
End Sub
```

Il existe deux moments de déclenchement avec des noms d'implémentation différents. Ces deux implémentations se distinguent dans **SELECT CASE**. Le code sera exécuté uniquement pour l'implémentation **FormController**. Cela est dû au fait que seul **FormController** a la variable **FormOperations**.

Outre Oui et Non, l'utilisateur peut également cliquer sur le bouton Fermer. Cela donne cependant la même valeur que Non, à savoir 7.

Si le formulaire est parcouru avec la touche de tabulation, l'utilisateur ne voit que la boîte de dialogue avec l'invite de confirmation. Cependant, les utilisateurs qui utilisent la barre de navigation verront également un message indiquant que l'enregistrement ne sera pas modifié.

Clé primaire à partir du numéro courant et de l'année

Lorsque les factures sont préparées, les soldes annuels sont affectés. Cela conduit souvent à une volonté de séparer les tableaux de facturation d'une base de données par année et de commencer une nouvelle table chaque année.

La solution de macro suivante utilise une méthode différente. Il écrit automatiquement la valeur du champ ID dans la table mais prend également en compte le champ « Année » qui existe dans la table comme clé primaire secondaire. Ainsi, les clés primaires suivantes peuvent apparaître dans le tableau :

Annee	ID
2014	1
2014	2
2014	3
2015	1
2015	2

De cette manière, un aperçu de l'année est plus facilement obtenu pour les documents.

```
Sub ID_et_Date_Courante
    Dim oSourceDonnees As Object
    Dim oConnexion As Object
    Dim oSQL_Commande As Object
    Dim stSql As String
    Dim oResultat As Object
    Dim oDoc As Object
    Dim oDrawpage As Object
    Dim oFormulaire As Object
    Dim oChamp1 As Object
    Dim oChamp2 As Object
    Dim oChamp3 As Object
    Dim inIDnouveau As Integer
    Dim inAnnee As Integer
    Dim unoDate
    oDoc = thisComponent
    oDrawpage = oDoc.drawpage
    oFormulaire = oDrawpage.forms.getByName("Formulaire")
    oChamp1 = oFormulaire.getByName("fmt_Annee")
    oChamp2 = oFormulaire.getByName("fmtID")
    oChamp3 = oFormulaire.getByName("dat_Date")
    If IsEmpty(oChamp2.getCurrentValue()) Then
        If IsEmpty(oChamp3.getCurrentValue()) Then
            unoDate = createUnoStruct("com.sun.star.util.Date")
            unoDate.Year = Year(Date)
            unoDate.Month = Month(Date)
            unoDate.Day = Day(Date)
            inAnnee = Year(Date)
        Else
            inAnnee = oChamp3.CurrentValue.Year
        End If
    End If
    oSourceDonnees = ThisComponent.Parent.CurrentController
    If Not (oSourceDonnees.isConnected()) Then
        oSourceDonnees.connect()
    End If
```

```

oConnexion = oSourceDonnees.ActiveConnection()
oSQL_Commande = oConnexion.createStatement()
stSql = "SELECT MAX(" & ""ID"" & ") + 1 FROM ""orders"" WHERE ""year"" = '" +
        inAnnee + "'"
oResultat = oSQL_Commande.executeQuery(stSql)
While oResultat.next
    inIDnouveau = oResultat.getInt(1)
Wend
If inIDnouveau = 0 Then
    inIDnouveau = 1
End If
oChamp1.BoundField.updateInt(inAnnee)
oChamp2.BoundField.updateInt(inIDnouveau)
If IsEmpty(oChamp3.getCurrentValue()) Then
    oChamp3.BoundField.updateDate(unoDate)
End If
End If
End Sub

```

Toutes les variables sont déclarées. Les contrôles de formulaire dans le formulaire principal sont accessibles. Le reste du code s'exécute uniquement si l'entrée du champ fmtID est toujours vide. Ensuite, si aucune date n'a été saisie, une structure de date est créée afin que la date et l'année en cours puissent être reportées dans les champs appropriés. Ensuite, une connexion est établie à la base de données, si elle n'existe pas déjà. La valeur la plus élevée du champ ID pour l'année en cours est incrémentée de 1. Si l'ensemble de résultats est vide, cela signifie qu'il n'y a aucune entrée dans le champ ID. À ce stade, 0 peut être entré dans le contrôle fmtID, mais la numérotation des commandes doit commencer à 1 afin que la variable inIDnouveau reçoive la valeur 1.

La valeur retournée pour l'année, l'ID et la date du jour (si aucune date n'a été saisie) sont transférés vers le formulaire.

Dans le formulaire, les champs de l'ID et de l'année des clés primaires sont protégés en écriture. Par conséquent, ils ne peuvent recevoir des valeurs qu'à l'aide de cette macro.

Cette macro est utilisée dans la base exemple Exemple_Numero_Serie_Annee.odt

Tâches de base de données développées à l'aide de macros

Établir une connexion à une base de données

```

oSourceDonnees = ThisComponent.Parent.DataSource
If Not oSourceDonnees.IsPasswordRequired Then
    oConnexion = oSourceDonnees.GetConnection("", "")

```

Ici, il serait possible de fournir un nom d'utilisateur et un mot de passe, si nécessaire. Dans ce cas, les guillemets contiendraient ("Utilisateur", "MotdePasse"). Au lieu d'inclure le nom d'utilisateur et un mot de passe en texte clair, la boîte de dialogue de protection par mot de passe est appelée :

```

Else
    oAuthentication = createUnoService("com.sun.star.sdb.InteractionHandler")
    oConnexion = oSourceDonnees.ConnectWithCompletion(oAuthentication)
End If

```

Si toutefois un formulaire dans le même fichier de base accède à la base de données, il vous suffit de :

```

oSourceDonnees = ThisComponent.Parent.CurrentController
If Not (oSourceDonnees.isConnected()) Then
    oSourceDonnees.connect()
End If
oConnexion = oSourceDonnees.ActiveConnection()

```

Ici, la base de données est connue, donc un nom d'utilisateur et un mot de passe ne sont pas nécessaires, car ils sont déjà désactivés dans la configuration de base HSQLDB pour la version interne.

Pour les formulaires en dehors de Base, la connexion se fait via le premier formulaire :

```
oSourceDonnees = ThisComponent.Drawpage.Forms(0)
oConnexion = oSourceDonnees.activeConnection
```

Copie de données d'une base de données à une autre

La base de données interne est une base de données mono-utilisateur. Les enregistrements sont stockés dans un fichier *.odb. L'échange de données entre différents fichiers de base de données n'était pas autorisé mais est néanmoins possible à l'aide de l'exportation et de l'importation.

Mais souvent, les fichiers *.odb sont configurés pour permettre l'échange automatique de données entre les bases de données. La procédure suivante peut être utile ici.

Une fois les variables déclarées, le chemin d'accès à la base de données actuelle est lu à partir d'un bouton du formulaire. Le nom de la base de données est séparé du reste du chemin. Le fichier cible des enregistrements est également présent dans ce dossier. Le nom de ce fichier est attaché au chemin pour permettre une connexion à la base de données cible.

La connexion à la base de données source est déterminée par rapport au formulaire qui contient le bouton : **ThisComponent.Parent.CurrentController**. La connexion à la base de données externe est établie à l'aide de **DatabaseContext** et du chemin.

```
Sub CopieDonnees
    Dim oContexteBase As Object
    Dim oSourceDonnees As Object
    Dim oSourceDonneesCible As Object
    Dim oConnexion As Object
    Dim oConnexionCible As Object
    Dim oDB As Object
    Dim oSQL_Commande As Object
    Dim oSQL_CommandeCible As Object
    Dim oResultat As Object
    Dim oResultatCible As Object
    Dim stSql As String
    Dim stSqlCible As String
    Dim inID As Integer
    Dim inIDCible As Integer
    Dim stNom As String
    Dim stVille As String
    Dim Drapeau As Boolean
    Drapeau = False
    oDB = ThisComponent.Parent
    stDir = Left(oDB.Location, Len(oDB.Location)-Len(oDB.Title))
    stDir = ConvertToUrl(stDir & "BDCible.odb")
    oSourceDonnees = ThisComponent.Parent.CurrentController
    If Not (oSourceDonnees.isConnected()) Then
        oSourceDonnees.connect()
    End If
    oConnexion = oSourceDonnees.ActiveConnection()
    oContexteBase = createUnoService("com.sun.star.sdb.DatabaseContext")
    oSourceDonneesTarget = oContexteBase.getByName(stDir)
    oConnexionTarget = oSourceDonneesTarget.GetConnection("", "")
    oSQL_Commande = oConnexion.createStatement()
    stSql = "SELECT * FROM ""Table""
    oResultat = oSQL_Commande.executeQuery(stSql)
    While oResultat.next
        inID = oResultat.getInt(1)
        stNom = oResultat.getString(2)
        stVille = oResultat.getString(3)
        oSQL_CommandeCible = oConnexionTarget.createStatement()
```

```

stSqlCible = "SELECT ""ID"" FROM ""Table"" WHERE ""ID"" = '"+inID+"'"
oResultatCible = oSQL_CommandeCible.executeQuery(stSqlCible)
inIDCible = - 1
While oResultatCible.next
    inIDCible = oResultatCible.getInt(1)
Wend
If inIDCible = - 1 Then
    stSqlCible = "INSERT INTO ""Table"" (""ID"", ""Nom"", ""Ville"") _
        VALUES ('"+inID+"', '"+stNom+"', '"+stVille+"')"
    oSQL_CommandeCible.executeUpdate(stSqlCible)
    Drapeau = True
End If
Wend
If Drapeau Then MsgBox("Données copiées") Else MsgBox ("Données non copiées")
End Sub

```

Les tables complètes de la base de données source sont lues et insérées, ligne par ligne, dans la table de la base de données cible à l'aide de la connexion qui a été établie. Avant l'insertion, une vérification est effectuée pour voir si une valeur a déjà été définie pour la clé primaire dans la table cible. Si tel est le cas, l'enregistrement n'est pas copié.

Il est également possible de faire en sorte qu'au lieu de copier un nouvel enregistrement, un enregistrement existant soit mis à jour. Dans tous les cas, cela garantit que la base de données cible contient des enregistrements avec la clé primaire correcte de la base de données source.

Cette macro est utilisée dans la base Exemple_CopieDonnees_Source_Cible.odt.

Importation directe des données de Calc

Il arrive souvent que Calc soit utilisé à la place de Base pour entrer des données dans un tableau. Ces données peuvent ensuite être copiées dans une table de base de données via le presse-papiers ou par glisser-déposer. Il est également possible d'exporter dans un fichier texte *.csv déjà inclus dans Base.

Mais, l'utilisateur peut aussi vouloir transférer simplement les données saisies sous Calc dans une base de données, afin de profiter des fonctionnalités d'un SGDB. C'est là qu'entre en jeu la macro suivante, qui est lancée à partir d'un formulaire via un bouton.¹⁷

La macro suppose les conditions suivantes :

- Les données doivent se trouver sur la « **feuille1** » du document Calc ;
- Sur cette feuille de travail les données sont disposées en colonnes ;
- La première ligne contient les noms de champs, qui doivent correspondre exactement aux noms des champs de la table dans la base de données. Ils peuvent être classés dans un ordre différent ;
- Les colonnes dont les en-têtes ne correspondent pas aux champs du tableau sont ignorées.

Les noms des variables ont volontairement été laissés en langue anglaise. Les commentaires et noms des éléments utilisés ont été traduits en français.

```

Sub ImportDonneesCalc(oEvent As Object)
    Dim oSourceDonnees As Object
    Dim oConnexion As Object
    Dim oSQL_Commande As Object
    Dim oResultat As Object
    Dim oDB As Object
    Dim oDoc As Object
    Dim oDocView As Object
    Dim oEtendue As Object

```

¹⁷ Voir la base de données "Exemple_Import_Données_Calc" associée à ce manuel.

```

Dim Arg()
Dim aColonne()
Dim aColonnes()
Dim aType()
Dim stSql As String
Dim stLigne As String
Dim stDir As String
Dim stColonnes As String
Dim stColonneDebut As String
Dim stColonneFin As String
Dim stLigneDebut As String
Dim stLigneFin As String
Dim stEtendue As String
Dim inCompteur As Integer
Dim loColonnes As Long
Dim loLignes As Long
Dim loID As Long
oSourceDonnees = ThisComponent.Parent.CurrentController
If Not (oSourceDonnees.isConnected()) Then
    oSourceDonnees.connect()
End If
oConnexion = oSourceDonnees.ActiveConnection()
oSQL_Commande = oConnexion.createStatement()

```

Après avoir déclaré les variables et vérifié la connexion à la source de données, lit les noms et les types de colonnes de la table dans la base de données. Sauf "ID" qui est une clé primaire qui n'existe que dans cette table en tant que clé primaire.

```

stSql = "SELECT COLUMN_NAME, TYPE_NAME FROM INFORMATION_SCHEMA. _
        SYSTEM_COLUMNS WHERE TABLE_NAME = 'Table' AND NOT COLUMN_NAME = 'ID'"
oResultat = oSQL_Commande.executeQuery(stSql)
inCompteur = 0
stColonnes = ""

```



Note

Pour Firebird, le code SQL doit être adapté :

```

stSql = "SELECT A.RDB$FIELD_NAME, C.RDB$TYPE_NAME
        FROM RDB$RELATION_FIELDS AS A, RDB$FIELDS AS B, RDB$TYPES AS C
        WHERE A.RDB$FIELD_SOURCE = B.RDB$FIELD_NAME
              AND B.RDB$FIELD_TYPE = C.RDB$TYPE
              AND C.RDB$FIELD_NAME = 'RDB$FIELD_TYPE'
              AND A.RDB$RELATION_NAME = 'Table'
              AND NOT A.RDB$FIELD_NAME = 'ID'"

```

Les noms et les types de colonnes sont lus et stockés dans des tableaux séparés. Cela fonctionnerait également dans un tableau bidimensionnel.

```

WHILE oResultat.next
    ReDim Preserve aColonne(inCompteur)
    ReDim Preserve aType(inCompteur)
    aColonne(inCompteur) = oResultat.getString(1)
    aType(inCompteur) = oResultat.getString(2)
    inCompteur = inCompteur+1
WEND

```

La valeur actuelle, la plus élevée, de la clé primaire « ID » est déterminée et incrémentée de 1. Dans l'exemple, la valeur de la clé n'est pas automatiquement incrémentée par HSQLDB, mais gérée par la macro.

```

stSql = "SELECT MAX('ID') FROM 'Table'"
oResultat = oSQL_Commande.executeQuery(stSql)
WHILE oResultat.next
    loID = oResultat.getInt(1) + 1
WEND

```

Le chemin d'accès au fichier Calc est déterminé, ici dans le même répertoire que le fichier de base de données exemple. Ce fichier est ouvert, mais rendu invisible afin qu'il ne passe pas au premier plan.

```
oDB = ThisComponent.Parent
globalscope.basicLibraries.loadLibrary("Tools") ' bibliothèque de fonctions
stDir = DirectoryNameoutofPath(ConvertFromURL(oDB.location), _ GetPathSeparator())
stDir = Left(oDB.Location, Len(oDB.Location)-Len(oDB.Title))
stDir = ConvertToUrl(stDir & "Donnees_Calc.ods")
oDoc = StarDesktop.loadComponentFromURL(stDir, "_blank", 0, Arg() )
oDocView = oDoc.CurrentController.Frame.ContainerWindow
oDocView.Visible = False
```

La zone utile de cellules de la feuille de calcul est déterminée. Cela se fait via **ColumnDescriptions** et **RowDescriptions**.

```
loColonnes = uBound(oDoc.Sheets(0).ColumnDescriptions) ' Nombre de colonnes
stColonneDebut = split(oDoc.Sheets(0).ColumnDescriptions(0))(1)
stColonneFin = split(oDoc.Sheets(0).ColumnDescriptions(loColonnes))(1)
loLignes = uBound(oDoc.Sheets(0).RowDescriptions) ' Nombre de lignes
stLigneDebut = split(oDoc.Sheets(0).RowDescriptions(0))(1)
stLigneFin = split(oDoc.Sheets(0).RowDescriptions(loLignes))(1)
stEtendue = stColonneDebut & stLigneDebut & ":" & stColonneFin & stLigneFin
```

La zone (**stEtendue**) est composée comme une chaîne. Cela se fait de la même manière que dans Calc, par exemple "A1:C7". Les données d'une telle plage peuvent être lues avec la fonction **GetDataArray()**.

```
oEtendue = oDoc.Sheets(0).getCellRangeByName(stEtendue)
aDonnees = oEtendue.getDataArray()
```

Les noms des colonnes se trouvent sur la première ligne. Ils peuvent avoir un ordre différent de celui prévu dans le tableau de la base. Ces noms sont donc stockés dans un tableau séparé pour une comparaison ultérieure. Elles ne sont pas interrogées dans la boucle suivante en tant que valeurs à lire dans le tableau. Par conséquent, la boucle commence avec **i=1**.

```
aColonnes = aDonnees(0)
FOR i = 1 TO uBound(aDonnees)
    aLigne = aDonnees(i)
    stColonnes = "" & "ID" & ""
    stLigne = loID
    FOR k = 0 TO loColonnes
        FOR n = 0 TO uBound(aColonne)
```

Dans ce qui suit, les noms des colonnes de Calc sont comparés avec ceux du tableau de base. En plus des colonnes de texte, le tableau de base contient ici aussi une colonne pour un montant en devise, qui est défini comme **DECIMAL**, et une date.

Pour le montant en devises, il peut être nécessaire de convertir le point décimal. C'est pourquoi ici la fonction **Cdbl** est couplée à la fonction **Str**.

Dans Calc une cellule au format date est enregistrée sous la forme d'un de numéro ISO. Ce code doit d'abord être vérifié pour voir si une date peut être formée à partir de celui-ci. Si cela est possible, une date conforme à SQL est compilée dans le format **AAAA-MM-JJ**, même avec des valeurs de jour et de mois à un chiffre.

```
IF aColonnes(k) = aColonne(n) THEN
    IF aType(n) = "DECIMAL" THEN
        IF aLigne(k) <> "" THEN
            aLigne(k) = Str(Cdbl(aLigne(k)))
        END IF
    END IF
    IF aType(n) = "DATE" THEN
        IF isDate(CDate(aLigne(k))) AND aLigne(k) <> "" THEN
            aLigne(k) = Year(CDate(aLigne(k))) & "-" &
                &Right("0"&Month(CDate(aLigne(k))),2)
```

```

        &"-"&Right("0"&Day(CDate(aLigne(k))),2)
    ELSE
        aLigne(k) = ""
    END IF
END IF

```

Si le contenu de la cellule est vide ou non valable pour un champ décimal ou un champ de date, selon le cas, NULL est transmis à la table de la base de données. Sinon, les contenus sont placés entre guillemets. Les contenus sont ensuite reliés entre eux à l'aide de virgules. Les colonnes sont également nommées dans l'ordre dans lequel elles ont été lues à partir de la table Calc.

```

    IF aLigne(k) = "" THEN
        aLigne(k) = "NULL"
    ELSE
        aLigne(k) = "'" & aLigne(k) & "'"
    END IF
    stLigne = stLigne & "," & aLigne(k)
    stColonnes = stColonnes & "," & aColonnes(k) & "'''"
END IF
NEXT
NEXT
stSql = "INSERT INTO ""Tabelle"" (" & stColonnes & ") VALUES (" & stLigne & ")"
oSQL_Commande.executeUpdate(stSql)

```

La valeur de la clé primaire est augmentée de 1 dans la macro.

```

        loID = loID + 1
    NEXT
    oDoc.close(True)
    oEvent.Source.Model.parent.reload()
End Sub

```

Une fois que le contenu de la feuille de calcul Calc a été entièrement lu, le document Calc (invisible) est fermé. Le formulaire est actualisé (rechargé – reload) à partir de l'événement déclencheur initial (oEvent) par oEvent.Source.Model.parent.reload().

Accès aux requêtes

Il est plus facile de créer des requêtes dans l'interface utilisateur graphique que de transférer leur texte dans des macros, avec la complication supplémentaire que des doubles guillemets sont obligatoires pour tous les noms de table et de champ.

```

Sub aContenuRequete
    Dim oFichierDonnees As Object
    Dim oRequete As Object
    Dim stRequete As String
    oFichierDonnees = ThisComponent.Parent.CurrentController.DataSource
    oRequete = oFichierDonnees.getQueryDefinitions()
    stRequete = oRequete.getByName("Requete").Command
    MsgBox stRequete
End Sub

```

Ici, le contenu d'un fichier *.odb est accessible à partir d'un formulaire. La requête est atteinte en utilisant **getQueryDefinitions()**. Le code SQL de la requête se trouve dans son champ **Command**. Cela peut ensuite être utilisé pour utiliser la commande plus loin dans une macro.

Lorsque vous utilisez le code SQL de la requête, vous devez veiller à ce que le code ne fasse pas référence à une autre requête. Cela conduit inévitablement au message que la table (apparente) de la base de données est inconnue. Pour cette raison, il est plus simple de créer des vues à partir de requêtes, puis d'accéder aux vues dans la macro.

Sécuriser votre base de données

Il peut parfois arriver, en particulier lors de la création d'une base de données, que le fichier ODB soit tronqué de manière inattendue. Une sauvegarde fréquente après édition est donc utile, en particulier lors de l'utilisation du module Rapports.

Lorsque la base de données est en cours d'utilisation peut être endommagée par une défaillance du système d'exploitation, notamment si cela se produit juste au moment où le fichier de base est fermé. C'est à ce moment que le contenu de la base de données est écrit dans le fichier.

De plus, il existe les soupçons habituels de panne pour les fichiers qui refusent soudainement de s'ouvrir, comme une panne de disque dur. Il ne fait donc aucun mal d'avoir une copie de sauvegarde la plus à jour possible. L'état des données ne change pas tant que le fichier ODB reste ouvert. Pour cette raison, les sous-programmes de sécurité peuvent être directement liés à l'ouverture du fichier. Copier simplement le fichier en utilisant le chemin de sauvegarde indiqué dans **Outils> Options> LibreOffice> Chemins**. La macro commence par écraser la version la plus ancienne après un nombre spécifique de copies (**inMax**).

```
Sub SauveBase(inMax As Integer)
    Dim oChemin As Object
    Dim oDoc As Object
    Dim sTitre As String
    Dim sUrl_Destination As String
    Dim sUrl_Source As String
    Dim i As Integer
    Dim k As Integer
    oDoc = ThisComponent
    sTitre = oDoc.Title
    sUrl_Source = oDoc.URL
    Do While sUrl_Source = ""
        oDoc = oDoc.Parent
        sTitre = oDoc.Title
        sUrl_Source = oDoc.URL
    Loop
```

Si la macro est exécutée lorsque vous lancez le fichier ODB, sTitre et sUrl_Source seront corrects. Cependant, si la macro est réalisée par un formulaire, il doit d'abord déterminer si une URL est disponible. Si l'URL est vide, un niveau supérieur (oDoc.Parent) est recherché.

```
oChemin = createUnoService("com.sun.star.util.PathSettings")
For i = 1 To inMax + 1
    If Not FileExists(oChemin.Backup & "/" & i & "_" & sTitre) Then
        If i > inMax Then
            For k = 1 To inMax - 1 To 1 Step -1
                If FileDateTime(oChemin.Backup & "/" & k & "_" & sTitre) <= _
                    FileDateTime(oChemin.Backup & "/" & k+1 & "_" & sTitre) Then
                    If k = 1 Then
                        i = k
                        Exit For
                    End If
                Else
                    i = k + 1
                    Exit For
                End If
            Next
        End If
        Exit For
    End If
Next
sUrl_Destination = oChemin.Backup & "/" & i & "_" & sTitre
FileCopy(sUrl_Source, sUrl_Destination)
End Sub
```

Vous pouvez également effectuer une sauvegarde pendant l'exécution de Base, à condition que les données puissent être réécrites du cache dans le fichier avant l'exécution du sous-programme

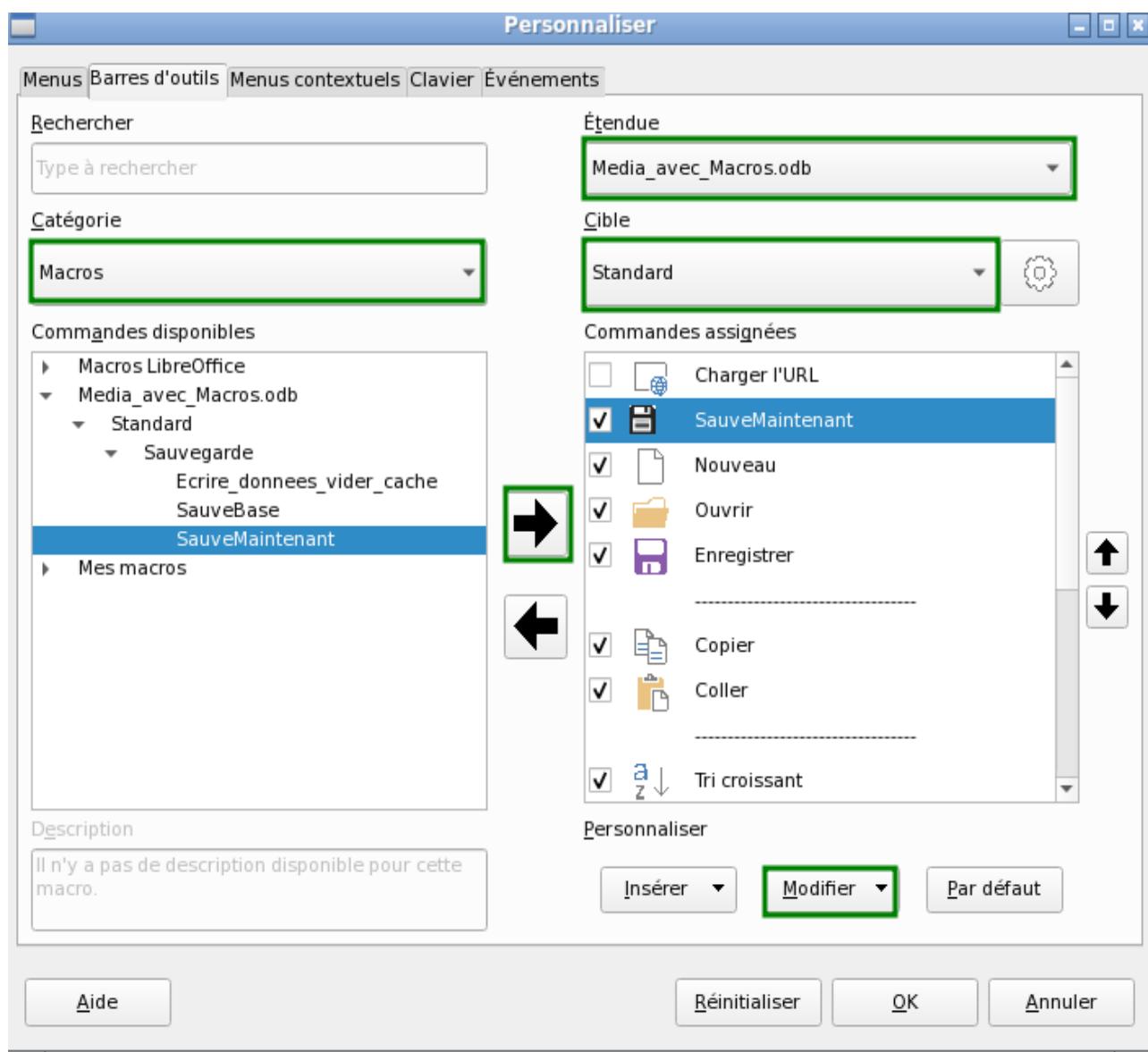
SauveBase. Il peut être utile de le faire, peut-être après un certain temps ou lorsqu'un bouton à l'écran est enfoncé. Cet effacement du cache est géré par le sous-programme suivant :

```
Sub Ecrire_donnees_vider_cache
    Dim oDonnees As Object
    Dim oSourceDonnees As Object
    oDonnees = ThisDatabaseDocument.CurrentController
    If Not (oDonnees.isConnected()) Then oDonnees.connect()
    oSourceDonnees = oDonnees.DataSource
    oSourceDonnees.flush
End Sub
```

Si tout cela doit être lancé à partir d'un seul bouton sur un formulaire, les deux procédures doivent être appelées par une autre procédure :

```
Sub SauveMaintenant
    Ecrire_donnees_vider_cache
    SauveBase(10)
End Sub
```

Surtout pour une macro de sécurité, il peut être judicieux de rendre la macro accessible via la barre d'outils de la base de données. Cela se fait dans la fenêtre principale du fichier de base en utilisant **Outils> Personnaliser> Barres d'outils**.



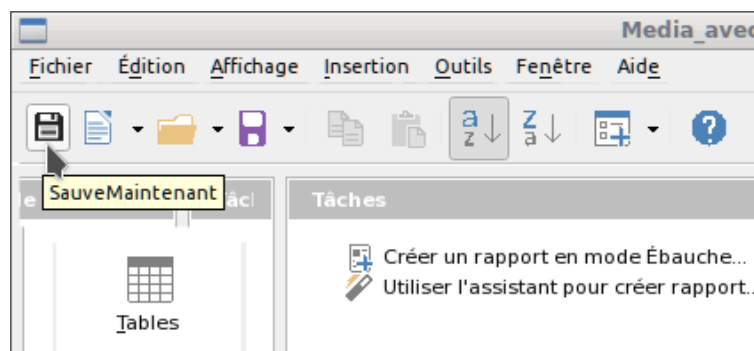
Dans la boîte de dialogue affichée sélectionner :

- dans la liste déroulante « Étendue », le nom du fichier contenant les macros, dans l'exemple « Media_avec_Macros.odt » ;
- dans « Catégorie », la macro « SauveMaintenant » ;
- dans « Cible », la barre d'outil « Standard », qui est toujours affichée dans toutes les parties de Base ;
- l'endroit où insérer la nouvelle commande en choisissant par un clic, dans « Commandes assignées », sur celle (existante) qui devra se trouver juste avant « SauveMaintenant ».
- Sans cela l'insertion se fera à la fin de la liste ;
- enfin, cliquez sur le bouton indiqué sur l'illustration pour transférer cette nouvelle fonction.

La commande est désormais disponible pour une utilisation dans la barre d'outils « Standard ». Pour affecter une icône à la commande, choisissez **Modifier > Modifier l'icône** pour ouvrir la boîte de dialogue suivante.



Sélectionnez une icône appropriée. Vous pouvez également créer et ajouter votre propre icône.



L'icône apparaît maintenant à la place du nom de la procédure. Le nom devient une info-bulle.

Pour exécuter la procédure, cliquez simplement sur l'icône dans la barre d'outils.



Note

Jusqu'à la version 6.4.7, les icônes personnalisées disparaissent après la fermeture de LibO et sont remplacées par le titre de la macro. Ce bug est corrigé dans LO 7.0.3

Compacter les bases de données

Il s'agit simplement d'une commande SQL (**SHUTDOWN COMPACT**), qui doit être exécutée de temps en temps, surtout après la suppression de nombreuses données. La base de données stocke les nouvelles données, mais réserve toujours l'espace pour les données supprimées. Dans les cas où les données ont été substantiellement modifiées, vous devez donc compacter la base de données.



Note

Depuis LibreOffice version 3.6, cette commande est automatiquement exécutée pour HSQLDB interne lorsque la base de données est fermée. Par conséquent, cette macro n'est plus nécessaire pour la base de données interne.

Une fois le compactage effectué, les tables ne sont plus accessibles. Le fichier doit être rouvert. Par conséquent, cette macro ferme le formulaire à partir duquel elle est appelée. Malheureusement, vous ne pouvez pas fermer le document lui-même sans provoquer de récupération lors de sa réouverture. Par conséquent, cette fonction fermeture est commentée.

```
Sub Compactage_BaseDeDonnees
    Dim stMessage As String
    oSourceDonnees = ThisComponent.Parent.CurrentController ' Accessible à partir du
formulaire
    If Not (oSourceDonnees.isConnected()) Then
        oSourceDonnees.connect()
    End If
    oConnexion = oSourceDonnees.ActiveConnection()
    oSQL_Statement = oConnexion.createStatement()
    stSql = "SHUTDOWN COMPACT" ' La base de données sera compactée et fermée
    oSQL_Statement.executeQuery(stSql)
    stMessage = "La base de données est en cours de compactage." + Chr(13) + "Le
formulaire va maintenant se fermer."
    stMessage = stMessage + Chr(13) + "Ensuite, le fichier de base de données doit
être fermé."
    stMessage = stMessage + Chr(13) + "La base de données n'est accessible qu'après la
réouverture du fichier de base de données."
    MsgBox stMessage
    ThisDatabaseDocument.FormDocuments.getByName("Maintenance").close
    REM La fermeture du fichier de base de données entraîne une opération
    REM de récupération lorsque vous l'ouvrez à nouveau.
    ' ThisDatabaseDocument.close(True)
End Sub
```

Diminution de l'index de table pour les champs à valeur automatique

Si beaucoup de données sont supprimées d'une table, les utilisateurs craignent souvent que la séquence de clés primaires générées automatiquement continue simplement vers le haut au lieu de recommencer à la valeur actuelle la plus élevée de la clé. Le sous-programme suivant lit la valeur actuellement la plus élevée du champ ID dans une table et définit la valeur de clé initiale à 1 de plus que ce maximum¹⁸.

Si le champ de clé primaire n'est pas appelé ID, la macro doit être modifiée en conséquence.

```
Sub Diminue_index_Table(stTable As String)
    REM Ce sous-programme définit le champ de clé primaire à incrémentation
    REM automatique (AutoChamp) avec le nom prédéfini "ID" sur la valeur
    REM la plus basse possible.
    Dim inNombre As Integer
    Dim inValeur_Sequence As Integer
    oSourceDonnees = ThisComponent.Parent.CurrentController ' Accessible via le
```

18 Cette procédure est utilisée dans le formulaire Maintenance de l'exemple Media_avecMacros.

```

formulaire
    If Not (oSourceDonnees.isConnected()) Then
        oSourceDonnees.connect()
    End If
    oConnexion = oSourceDonnees.ActiveConnection()
    oSQL_Statement = oConnexion.createStatement()
    ' La valeur la plus élevée de "ID" est déterminée
    stSql = "SELECT MAX(""ID"") FROM ""+stTable+""""
    Requete = oSQL_Instruction.executeQuery(stSql)
    ' La requête est lancée et la valeur de retour stockée dans la variable
    ' oResultat_Requete
    If Not IsNull(oResultat_Requete) Then
        While oResultat_Requete.next
            ' Le premier champ de données est lu
            inNombre = oResultat_Requete.getInt(1)
            'enregistrement suivant, dans ce cas aucun car un seul
            ' enregistrement existe
        Wend
        ' Si la valeur la plus élevée n'est pas une valeur, ce qui signifie
        ' que la table est vide, la valeur la plus élevée est définie sur -1
        If inNombre = "" Then
            inNombre = -1
        End If
    End If
    ' La valeur la plus élevée est augmentée de 1
    inValeur_Sequence = inNombre+1
    REM Une nouvelle commande est préparée pour la base de données.
    REM L'ID recommencera à partir de inNombre + 1.
    REM Cette instruction n'a pas de valeur de retour,
    REM car aucun enregistrement n'est en cours de lecture
    oSQL_statement = oConnexion.createStatement()
    oSQL_statement.executeQuery("ALTER TABLE "" + stTable + "" _
ALTER COLUMN ""ID"" RESTART WITH " + inValeur_Sequence + "")
    End If
End Sub

```

Impression depuis Base

La méthode standard pour obtenir un document imprimable depuis Base est d'utiliser un rapport. Alternativement, les tables et les requêtes peuvent être copiées dans Calc et préparées pour être imprimées. Bien entendu, l'impression directe d'un formulaire à partir de l'écran est également possible.

Impression d'un rapport à partir d'un formulaire interne

Normalement, la génération des rapports se fait à partir de l'interface utilisateur de base. Un clic sur le nom du rapport lance la préparation du rapport. Ce serait bien sûr plus facile si le rapport pouvait être lancé directement à partir d'un formulaire.

```

Sub LanceRapport
    ThisDatabaseDocument.ReportDocuments.getByName("Rapport").open
End Sub

```

Tous les rapports sont accessibles par nom depuis leur conteneur **ReportDocuments**. Ils sont ouverts avec l'instruction **open**. Si un rapport est lié à une requête filtrée via le formulaire, cette méthode permet à l'enregistrement en cours d'être imprimé.

Lancement, formatage, impression directe et fermeture d'un rapport

Ce serait encore plus agréable si le rapport pouvait être envoyé directement à l'imprimante. La combinaison de procédures suivante ajoute quelques petites fonctionnalités. Il sélectionne d'abord l'enregistrement actif dans le formulaire, reformate le rapport afin que les champs de texte soient définis automatiquement à la bonne hauteur, puis lance le rapport. Enfin, le rapport est imprimé et éventuellement stocké sous forme de pdf. Et tout cela se produit presque complètement en arrière-

plan, car le rapport passe à invisible directement après l'ouverture du formulaire et se referme après l'impression. Des suggestions pour les différentes procédures ont été faites par Andrew Pitonyak, Thomas Krumbein et Lionel Elie Mamane.

```
Sub DemarreRapport(oEvent As Object)
    Dim oFormulaire As Object
    Dim stSql As String
    Dim oSourceDonnees As Object
    Dim oConnexion As Object
    Dim oSQL_commande As Object
    Dim oRapport As Object
    Dim oRapportVue As Object
    oFormulaire = oEvent.Source.model.parent
    stSql = "UPDATE ""Filter"" SET ""Integer"" = ' " +
        oFormulaire.getInt(oFormulaire.findColumn("ID")) + "' WHERE ""ID"" = TRUE"
    oSourceDonnees = ThisComponent.Parent.CurrentController
    If Not (oSourceDonnees.isConnected()) Then
        oSourceDonnees.connect()
    End If
    oConnexion = oSourceDonnees.ActiveConnection()
    oSQL_commande = oConnexion.createStatement()
    oSQL_commande.executeUpdate(stSql)
    oRapport = ThisDatabaseDocument.ReportDocuments.getByName("Reportname").open
    oRapportVue = oRapport.CurrentController.Frame.ContainerWindow
    oRapportVue.Visible = False
    HauteurLigneRapportAuto(oRapport)
End Sub
```

La procédure *DemarreRapport* est liée à un bouton du formulaire. À l'aide de ce bouton, la clé primaire de l'enregistrement actuel peut être lue. À partir de l'événement qui lance la macro, nous pouvons accéder au formulaire (*oFormulaire*). Le nom du champ de clé primaire est donné ici sous la forme « ID ». En utilisant ***oFormulaire.getInt (oFormulaire.findColumn ("ID"))***, la clé est lue dans le champ sous la forme d'un entier. Cette valeur est stockée dans une table de filtres. La table de filtrage contrôle une requête pour garantir que seul l'enregistrement actuel sera utilisé pour le rapport.

Le rapport peut être ouvert sans référence au formulaire. Il est alors accessible en tant qu'objet (*oRapport*). La fenêtre de rapport est rendue invisible. Malheureusement, il ne peut pas être invisible lorsqu'il est appelé, il apparaît donc brièvement, puis est rempli avec le contenu approprié en arrière-plan.

Ensuite, la procédure *HauteurLigneRapportAuto* est lancée. Cette procédure reçoit une référence au rapport ouvert en tant qu'argument.

La hauteur de la ligne d'enregistrement peut être réglée automatiquement au moment de l'impression. S'il y a probablement trop de texte dans un champ particulier, le texte est tronqué et le reste est indiqué par un triangle rouge. Lorsque cela ne fonctionne pas, la procédure suivante garantira que dans toutes les tables portant le nom *Détail*, le contrôle automatique de la hauteur sera activé.

```
Sub HauteurLigneRapportAuto(oRapport As Object)
    Dim oTables As Object
    Dim oTable As Object
    Dim inT As Integer
    Dim inI As Integer
    Dim oLignes As Object
    Dim oLigne As Object
    oTables = oRapport.getTextTables()
    For inT = 0 To oTables.count() - 1
        oTable = oTables.getByIndex(inT)
        If Left$(oTable.name, 6) = "Detail" Then
            oLignes = oTable.Rows
            For inI = 0 To oLignes.count - 1
                oLigne = oLignes.getByIndex(inI)
            Next inI
        End If
    Next inT
End Sub
```

```

        oLigne.IsAutoHeight = True
    Next inI
End If
Next inT
ImprimeFermeRapport(oRapport)
End Sub

```

Lors de la création du rapport, il faut veiller à ce que tous les champs de la même ligne de la section « Détail » aient la même hauteur. Sinon, le contrôle automatique de la hauteur peut soudainement définir une ligne pour doubler la hauteur.

Une fois que toutes les tables portant le nom « Detail » ont été réglées automatiquement en hauteur, le rapport est envoyé à l'imprimante par la procédure ImprimeFermeRapport.

Le tableau « Props » contient les valeurs associées à une imprimante dans un document. Pour la commande d'impression, le nom de l'imprimante par défaut est important. Le rapport doit rester ouvert jusqu'à ce que l'impression soit réellement terminée. Ceci est garanti en donnant le nom de l'imprimante et la commande « Attendre que j'aie fini » (**Wait**) comme arguments.

```

Sub ImprimeFermeRapport(oRapport As Object)
    Dim Props
    Dim stImprimante As String
    Props = oRapport.getPrinter()
    stImprimante = Props(0).value
    Dim arg(1) As New com.sun.star.beans.PropertyValue
    arg(0).name = "Name"
    arg(0).value = "<" & stImprimante & ">"
    arg(1).name = "Wait"
    arg(1).value = True
    oRapport.print(arg())
    oRapport.close(true)
End Sub

```

Ce n'est que lorsque l'impression a été complètement envoyée à l'imprimante que le document est fermé.

Pour les paramètres de l'imprimante, consultez la section Imprimante et paramètres d'impression du wiki.

Si, à la place (ou en plus) d'une impression, vous voulez un pdf du document comme copie de sécurité, la méthode **storeToURL()** peut être utilisée :

```

Sub EnregistreRapportPDF(oRapport As Object)
    Dim stUrl As String
    Dim arg(0) As New com.sun.star.beans.PropertyValue
    arg(0).name = "FilterName"
    arg(0).value = "writer_pdf_Export"
    stUrl = "file:///..."
    oRapport.storeToURL(stUrl, arg())
End Sub

```

L'URL doit bien sûr être une adresse URL complète. Mieux encore, cette adresse doit être liée à un enregistrement permanent du document imprimé tel qu'un numéro de facture. Sinon, il se peut qu'un fichier important soit simplement écrasé par l'impression suivante.

Impression de rapports à partir d'un formulaire externe

Il y a des problèmes lorsque des formulaires externes sont utilisés. Les rapports se trouvent dans le fichier *.odb et ne sont pas disponibles à l'aide du navigateur de sources de données.

```

Sub LanceRapportExterne(oEvent As Object)
    Dim oChamp As Object
    Dim oFormulaire As Object
    Dim oDocument As Object
    Dim oDocVue As Object
    Dim Arg()

```

```

oChamp = oEvent.Source.Model
oFormulaire = oChamp.Parent
sURL = oFormulaire.DataSourceName
oDocument = StarDesktop.loadComponentFromURL(sURL, "_blank", 0, Arg())
oDocVue = oDocument.CurrentController.Frame.ContainerWindow
oDocVue.Visible = False
oDocument.getCurrentController().connect
Wait(100)
oDocument.ReportDocuments.getByName("Rapport").open
oDocument.close(True)
End Sub

```

Le rapport est lancé à partir d'un bouton sur le formulaire externe. Le bouton indique au formulaire le chemin d'accès au fichier *.odb : **oForm.DataSourceName**. Ensuite, le fichier est ouvert à l'aide de **loadComponentFromURL**. Le fichier doit rester en arrière-plan afin que la vue du document soit accessible et que l'interface soit définie sur **Visible = False**. Idéalement, cela aurait dû être fait directement en utilisant la liste d'arguments **Arg()**, mais les tests montrent que cela ne donne pas le résultat correct.

Le rapport ne peut pas être appelé immédiatement à partir du document ouvert, car la connexion n'est pas encore prête. Le rapport apparaît avec un arrière-plan gris, puis LibreOffice plante. Une courte attente de 100 millisecondes résout ce problème. Des tests pratiques sont nécessaires pour déterminer le temps d'attente minimum. Le rapport est maintenant lancé. Comme le rapport sera dans un fichier texte séparé, le fichier *.odb ouvert peut être refermé. La méthode **oDocument.close (True)** transmet cette instruction au fichier *.odb. Le fichier ne sera fermé que lorsqu'il n'est plus actif, c'est-à-dire qu'aucun autre enregistrement ne doit être transmis au rapport.

Un accès similaire peut être lancé à partir de formulaires dans le fichier *.odb, mais dans ce cas, le document ne doit pas être fermé.

Vous pouvez obtenir des impressions de bonne qualité beaucoup plus rapidement qu'avec le Générateur de rapports en utilisant des macros associées à la fonction de publipostage ou à des champs de texte.

Faire un publipostage depuis Base

Parfois, un rapport est tout simplement insuffisant pour produire des lettres de bonne qualité aux destinataires. Les champs de texte d'un rapport sont d'une utilité très limitée dans la pratique. Au lieu de cela, une lettre de publipostage peut être créée dans Writer. Il n'est cependant pas nécessaire d'ouvrir d'abord Writer, de faire toute la saisie et de la personnalisation là-bas, puis de l'imprimer. Vous pouvez faire tout cela directement depuis Base, en utilisant une macro.

```

Sub ImprimePublipostage
Dim oPublipostage As Object
Dim aProps()
oPublipostage = CreateUnoService("com.sun.star.text.MailMerge")

```

Le nom donné à la source de données est celui sous lequel la base de données est enregistrée dans LibreOffice. Ce nom n'a pas besoin d'être identique au nom de fichier. Le nom enregistré dans cet exemple est Adresses.

```

oPublipostage.DataSourceName = "Adresses"

```

Le chemin vers le fichier publipostagedoit être formaté selon les conventions de votre système d'exploitation, dans cet exemple un chemin absolu dans un système Linux.

```

oPublipostage.DocumentURL = ConvertToUrl("home/utilisateur/Documents/ _
publipostage.odt")

```

Le type de commande est défini. 0 représente une table, 1 une requête et 2 une commande SQL directe.

```

oPublipostage.CommandType = 1

```

Ici, une requête a été choisie avec le nom *RequetePublipostage*.

```
oPublipostage.Command = "RequetePublipostage"
```

Un filtre est utilisé pour déterminer les enregistrements à utiliser pour l'impression de publipostage. Ce filtre peut, par exemple, être spécifié à l'aide d'un contrôle de formulaire et transmis de Base à la macro. L'utilisation de la clé primaire d'un enregistrement peut entraîner l'impression d'un seul document.

Dans cet exemple, le champ Sexe de la RequetePublipostage est sélectionné, puis sont recherchés les enregistrements contenant « m » dans ce champ.

```
oPublipostage.Filter = ""Sexe""="m""
```

Les types de sortie disponibles sont Imprimante (1), Fichier (2) et Courrier (3). Ici, à des fins de test, un fichier de sortie est choisi. Ce fichier est stocké sur le chemin donné. Pour chaque enregistrement de publipostage, il y aura une impression. Pour distinguer cette impression, le champ nom de famille est incorporé au nom de fichier.

```
oPublipostage.OutputType = 2
oPublipostage.OutputUrl = ConvertToUrl("home/utilisateur/Documents")
oPublipostage.FileNameFromColumn = True
oPublipostage.Filenameprefix = "NomFamille"
oPublipostage.execute(aProps())
End Sub
```

Si le filtre est fourni avec ses données via le formulaire, cela permet d'effectuer des publipostages sans ouvrir Writer.

Impression via des champs de texte

En utilisant **Insertion> Champ> Autres champs> Fonctions> Substituant**, un modèle peut être créé dans Writer pour un document qui sera imprimé ultérieurement. Les espaces réservés doivent porter les mêmes noms que les champs de la table de base de données ou de la requête sous-jacente au formulaire à partir duquel la macro est appelée.

Pour le cas simple, le type à choisir pour l'espace réservé est « Texte ».

Le chemin d'accès au modèle doit être fourni dans la macro. Un nouveau document Sans nom 1.odt est créé. La macro remplit les espaces réservés avec le contenu de l'enregistrement actuel de la requête. Le document ouvert peut ensuite être modifié selon les besoins.

L'exemple de base de données Exemple_BasedeDonnees_LettreType_directe.odt montre comment une facture complète peut être produite à l'aide de champs de texte et d'accéder à une table préparée dans le document modèle. Contrairement aux factures créées avec le Générateur de rapports, ce type de création de facture n'a pas de limites de hauteur pour les champs de la table. Tout le texte est affiché. Cet exemple permet également de créer des lettres-type depuis un formulaire.

Voici une partie du code, principalement fournie par Dpunch :

```
Sub Remplir_ChampsTexte
oFormulaire = thisComponent.Drawpage.Forms.MainForm
If oFormulaire.RowCount = 0 Then
    MsgBox "Aucun enregistrement disponible pour l'impression"
    Exit Sub
End If
```

Le formulaire principal est activé. Le bouton qui lance la macro peut également être utilisé pour trouver le formulaire. Ensuite, la macro établit que le formulaire contient réellement des données imprimables.

L'accès direct à l'URL depuis le formulaire n'est pas possible. Cela doit être fait en utilisant la référence de niveau supérieur à la base de données.

```
oColonnes = oFormulaire.Columns
oDB = ThisComponent.Parent
```

Le titre de la base de données est séparé de l'URL.

```
stDir = Left(oDB.Location, Len(oDB.Location)-Len(oDB.Title))
```

Le modèle est trouvé et ouvert

```
stDir = stDir & "Exemple_ChampsTexte.ott"
Dim args(0) As New com.sun.star.beans.PropertyValue
args(0). Name = "AsTemplate"
args(0). Value = True
oNouveauDocument = StarDesktop.loadComponentFromURL(stDir, "_blank", 0, args)
```

Les champs de texte sont écrits.

```
oChampsTexte = oNouveauDocument.Textfields.createEnumeration
Do While oChampsTexte.hasMoreElements
    oChampTexte = oChampsTexte.nextElement
    If oChampTexte.supportsService("com.sun.star.text.TextField.JumpEdit") Then
        stNomColonne = oChampTexte.PlaceHolder
```

Le substituant représente le champ de texte.

```
If oColonnes.hasByName(stNomColonne) Then
```

Si le nom du champ de texte est le même que le nom de la colonne dans l'ensemble de données sous-jacent, le contenu de la base de données est transféré dans le champ du document texte.

```
    inIndex = oFormulaire.findColumn(stNomColonne)
    oChampTexte.Anchor.String = oFormulaire.getString(inIndex)
End If
End If
Loop
End Sub
```

Appeler des applications externes pour ouvrir des fichiers

Cette procédure permet avec un simple clic dans un champ de texte d'appeler le programme qui est lié au suffixe du nom de fichier dans le système d'exploitation. De cette manière, des liens Internet peuvent être suivis ou un programme de messagerie lancé pour une adresse spécifique stockée dans la base de données.

Pour cette section, voir aussi l'exemple de base de données :
Exemple_Activer_Fichier_Courriel.odt

```
Sub Activer_SiteWeb_ou_Courriel
    Dim oDoc As Object
    Dim oDrawpage As Object
    Dim oFormulaire As Object
    Dim oChamp As Object
    Dim oShell As Object
    Dim stChamp As String
    oDoc = thisComponent
    oDrawpage = oDoc.Drawpage
    oFormulaire = oDrawpage.Forms.getByName("Formulaire")
    oChamp = oFormulaire.getByName("url_ou_courriel")
```

Le contenu du champ nommé est lu. Il peut s'agir d'une adresse Web commençant par « http :// », d'une adresse de courriel contenant « @ » ou d'un chemin vers un document (par exemple, une image ou un fichier PDF stocké en externe).

```
stChamp = oChamp.Text
If stChamp = "" Then
```

```
Exit Sub
End If
```

Si le champ est vide, la macro se ferme immédiatement. Lors de la saisie des données, il arrive souvent que les champs soient accessibles à l'aide de la souris, mais le fait de cliquer sur le champ dans le but d'y écrire pour la première fois ne doit pas conduire à l'exécution du code de macro.

Maintenant, le champ est scanné pour un caractère «@». Cela indiquerait une adresse de courriel. Le programme de messagerie doit être lancé pour envoyer du courrier à cette adresse.

```
If InStr(stChamp, "@") Then
    stChamp = "mailto:" + stChamp
```

S'il n'y a pas de «@», le terme est converti en URL. Si cela commence par 'http://', nous n'avons pas affaire à un fichier dans le système de fichiers local mais à une ressource Internet qui doit être recherchée avec un navigateur Web. Sinon, le chemin commencera par le terme 'file:///'.

```
Else
    stChamp = convertToUrl(stChamp)
End If
```

Maintenant, le programme attribué par le système d'exploitation à ces fichiers est recherché. Pour le mot-clé 'mailto:' c'est le programme de messagerie, pour 'http://' le navigateur, sinon le système doit décider d'utiliser le suffixe du nom de fichier.

```
oShell = createUnoService("com.sun.star.system. SystemShellExecute")
oShell.execute(stChamp,,0)
End Sub
```

Appeler un programme de messagerie avec un contenu prédéfini

L'exemple précédent peut être étendu pour lancer un programme de messagerie avec un sujet et un contenu prédéfinis.

Pour cette section, voir aussi l'exemple de base de données :
 Exemple_Activer_Courriel_Fichier.odt.

Le programme de messagerie est lancé en utilisant 'mailto:recipient?subject=&body= &cc=&bcc='. Les deux dernières entrées ne sont pas présentes dans le formulaire. Les pièces jointes ne sont pas prévues dans la définition de "mailto" mais parfois 'attachment=' fonctionne.

```
Sub Activer_Courriel
    Dim oDoc As Object
    Dim oDrawpage As Object
    Dim oFormulaire As Object
    Dim oChamp1 As Object
    Dim oChamp2 As Object
    Dim oChamp3 As Object
    Dim oChamp4 As Object
    Dim oShell As Object
    Dim stChamp1 As String
    Dim stChamp2 As String
    Dim stChamp3 As String
    Dim stChamp4 As String
    oDoc = thisComponent
    oDrawpage = oDoc.Drawpage
    oFormulaire = oDrawpage.Forms.getByName("Formulaire")
    oChamp1 = oFormulaire.getByName("Adresse_Courriel")
    oChamp2 = oFormulaire.getByName("Sujet_Courriel")
    oChamp3 = oFormulaire.getByName("Contenu_Courriel")
    stChamp1 = oChamp1.Text
    If stChamp1 = "" Then
        MsgBox "Adresse courriel manquante." & Chr(13) & _
            "Le programme de messagerie ne sera pas activé", 48, "Envoyer courriel"
```

```
Exit Sub
End If
```

La conversion en URL est nécessaire pour empêcher les caractères spéciaux et les sauts de ligne d'interférer avec l'appel. Cela préfixe cependant le chemin avec le terme 'file:///'. Ces 8 caractères au début ne sont pas transférés.

```
stChamp2 = Mid(ConvertToUrl(oChamp2.Text),9)
stChamp3 = Mid(ConvertToUrl(oChamp3.Text),9)
```

Contrairement à un simple lancement de programme, les détails de l'appel du courrier sont donnés ici dans le cadre de l'appel d'exécution.

```
oShell = createUnoService("com.sun.star.system.SystemShellExecute")
oShell.execute("mailto:" + stChamp1 + "?subject=" + stChamp2 + "&body=" +
stChamp3,,0)
End Sub
```



Note

L'envoi de courriels à l'aide d'un programme de messagerie peut également être effectué à l'aide du code suivant, mais le contenu réel du courriel ne peut pas être inséré de cette manière.

```
Dim attaches(0)
oCourrielleur = _ createUnoService("com.sun.star.system.SimpleSystemMail")
oProgrammeCourriel = oCourrielleur.querySimpleMailClient()
oNouveauMessage = oProgrammeCourriel.createSimpleMailMessage()
oNouveauMessage.setRecipient(stField1)
oNouveauMessage.setSubject(stField2)
attaches(0) = "file:///..."
oNouveauMessage.setAttachement(attaches())
oProgrammeCourriel.sendSimpleMailMessage(oNouveauMessage, 0)
```

Pour les paramètres utilisables, voir les détails de l'API :

http://api.libreoffice.org/docs/idl/ref/interfacecom_1_1sun_1_1star_1_1system_1_1XSimpleMailMessage.html

Changer le pointeur de la souris lors du parcours d'un contrôle

Ceci est normal sur Internet et Base le recrée : le pointeur de la souris traverse un lien et se transforme en main pointée. Le texte du lien peut également changer ses propriétés, devenant bleu et souligné. La ressemblance avec un lien Internet est parfaite. Tout utilisateur attendra un clic pour ouvrir un programme externe.

Pour cette section, voir l'exemple de base de données Exemple_Activer_Courriel_Fichier.odt

Cette courte procédure doit être liée à l'événement '**Souris à l'intérieur**'

```
Sub Pointeur_Souris(oEvent As Object)
REM Voir aussi Bibliothèques standard : Outils → ModuleControls → SwitchMousePointer
Dim oPointeur As Object
oPointeur = createUnoService("com.sun.star.awt.Pointer")
oPointeur.setType(27) 'pour les types voir com.sun.star.awt.SystemPointer
oEvent.Source.Peer.SetPointer(oPointeur)
End Sub
```

Affichage des formulaires sans barre d'outils

Les nouveaux utilisateurs de Base sont souvent irrités par le fait qu'une barre d'outils existe mais ne soit pas utilisable dans un formulaire. Ces barres d'outils peuvent être supprimées de

différentes manières. Les meilleurs moyens dans toutes les versions de LibreOffice sont les deux décrits ci-dessous.

Les tailles de fenêtre et les barres d'outils sont généralement contrôlées par une macro qui est lancée à partir d'un document de formulaire en utilisant **Outils > Personnaliser > Événements > Ouvrir le document**. Cela fait référence à l'ensemble du document, pas à un formulaire principal ou à un sous-formulaire individuel.

Formulaires sans barre d'outils dans la fenêtre

La taille d'une fenêtre peut être modifiée. En utilisant le bouton approprié, elle peut également être fermée. Ces tâches sont effectuées par le gestionnaire de fenêtres de votre système. La position et la taille d'une fenêtre à l'écran peuvent être fournies par une macro au démarrage du programme¹⁹.

```
Sub Cache_Barres_Outils
    Dim oCadre As Object
    Dim oFenetre As Object
    Dim oLayoutMng As Object
    Dim aElements()
    oCadre = StarDesktop.getCurrentFrame()
    ' Le titre du formulaire doit être affiché dans la barre de titre de la fenêtre.
    oCadre.setTitle "Mon Formulaire"
    oFenetre = oCadre.getContainerWindow()
```

La fenêtre est agrandie. Ce n'est pas la même chose que le mode plein écran, car la barre des tâches est toujours visible et la fenêtre a une barre de titre, qui peut être utilisée pour changer sa taille ou la fermer.

```
oFenetre.IsMaximized = true
```

Il est possible de créer une fenêtre avec une taille et une position spécifiques. Ceci est réalisé avec 'oFenetre.setPosSize(0,0,600,400,15)'. Ici, la fenêtre apparaît dans le coin supérieur gauche de l'écran avec une largeur de 600 pixels et une hauteur de 400. Le dernier chiffre indique que toutes les dimensions, en pixels, sont données. On l'appelle Flag. Flag est calculé à partir de la somme des valeurs suivantes : x = 1, y = 2, largeur = 4, hauteur = 8. Comme x, y, la largeur et la hauteur sont toutes données, Flag a la valeur 1+2+4+8=15.

```
oLayoutMng = oCadre.LayoutManager
aElements = oLayoutMng.getElements()
For i = LBound(aElements) To UBound(aElements)
    If aElements(i).ResourceURL = _
        "private:resource/toolbar/formsnavigationbar" Then
    Else
        oLayoutMng.hideElement(aElements(i).ResourceURL)
    End If
Next
ThisComponent.CurrentController.Sidebar.Visible = False
End Sub
```

Pour la barre de navigation de formulaire, rien n'est à faire, car il faut la conserver pour que le formulaire reste utilisable. Seules les barres d'outils autres que la barre de navigation doivent être masquées. Pour cette raison, il n'y a aucune action pour ce cas.

Si les barres d'outils ne sont pas restaurées directement après avoir quitté le formulaire, elles seront toujours masquées. Elles peuvent bien sûr être restaurées en utilisant **Affichage > Barres d'outils**. Mais ce serait plutôt ennuyeux si la barre d'outils standard (**Affichage > Barres d'outils > Standard**) ou la barre d'état (**Affichage > Barre d'état**) manquait.

¹⁹ Un exemple de cette macro est visible dans le fichier Exemple_Formulaire_a_formulaire.odt

Cette procédure restaure ("showElement") les barres d'outils de leur état caché ("hideElement"). Les commentaires contiennent les barres dont l'absence est la plus susceptible d'être remarquée.

```
Sub Affiche_Barre_Outils
    Dim oCadre As Object
    Dim oLayoutMng As Object
    Dim aElements()
    oCadre = StarDesktop.getCurrentFrame()
    oLayoutMng = oCadre.LayoutManager
    aElements = oLayoutMng.getElements()
    For i = LBound(aElements) To UBound(aElements)
        oLayoutMng.showElement(aElements(i).ResourceURL)
    Next
    ' éléments importants qui peuvent être absents :
    ' "private: resource/toolbar/standardbar"
    ' "private: resource/statusbar/statusbar"
End Sub
```

Les macros sont liées à : **Outils > Personnaliser > Événements > Ouvrir le document > Cache_Barre_Outils et Document fermé > Afficher_Barres_Outils.**

Malheureusement, souvent, les barres d'outils ne parviennent pas à revenir. Dans le pire des cas, il peut être utile de ne pas lire les éléments que le gestionnaire de mise en page connaît déjà, mais d'abord de créer des barres d'outils particulières, puis de les afficher :

```
Sub Affiche_Barres_Outils
    Dim oCadre As Object
    Dim oLayoutMng As Object
    Dim i As Integer
    Dim aElements(5) As String
    oCadre = StarDesktop.getCurrentFrame()
    oLayoutMng = oCadre.LayoutManager
    aElements(0) = "private:resource/menubar/menubar"
    aElements(1) = "private:resource/statusbar/statusbar"
    aElements(2) = "private:resource/toolbar/formsnavigationbar"
    aElements(3) = "private:resource/toolbar/standardbar"
    aElements(4) = "private:resource/toolbar/formdesign"
    aElements(5) = "private:resource/toolbar/formcontrols"
    For i = LBound(aElements) To UBound(aElements)
        If Not(oLayoutMng.requestElement(i)) Then
            oLayoutMng.createElement(aElements(i))
        End If
        oLayoutMng.showElement(aElements(i))
    Next i
    ThisComponent.CurrentController.Sidebar.Visible = True
End Sub
```

Les barres d'outils à créer sont nommées explicitement. Si une barre d'outils correspondante n'est pas disponible pour le gestionnaire de disposition, elle est créée à l'aide de createElement puis affichée en utilisant showElement.

Formulaires en mode plein écran

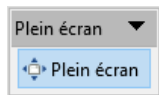
En mode plein écran, tout l'écran est couvert par le formulaire. Il n'y a pas de barre des tâches ou d'autres éléments qui pourraient indiquer si d'autres programmes sont en cours d'exécution²⁰.

```
Sub PleinEcran(boSwitch As Boolean)
    Dim oDispatcher As Object
    Dim Props(0) As New com.sun.star.beans.PropertyValue
    oDispatcher = createUnoService("com.sun.star.frame.DispatchHelper")
    Props(0).Name = "FullScreen"
    Props(0).Value = boSwitch
    oDispatcher.executeDispatch(ThisComponent.CurrentController.Frame, _
        ".uno:FullScreen", "", 0, Props())
End Sub
```

20 Un exemple de cette macro est visible dans le fichier Exemple_Formulaire_a_formulaire.odt

Cette fonction est lancée par les macros suivantes, qui déclenchent également soit la procédure de masquage des barres d'outils (**Cache_Barres_Outils**), soit celle d'affichage des barres d'outils (**Affiche_Barres_Outils**). Seule persiste la barre d'outils « Navigation pour formulaire ».

```
Sub PleinEcran_on
    PleinEcran(true)
    Cache_Barres_Outils
End Sub
```

La barre d'outils spécifique au mode plein écran « **CTRL + MAJ + J** »  n'apparaît pas, ce qui empêche la possibilité d'utiliser la touche « **ESC** » pour désactiver ce mode. Un bouton spécifique doit être utilisé pour la sortie du mode plein écran et le rétablissement des barres d'outils, cette commande figure à la ligne suivante :

```
Sub PleinEcran_off
    PleinEcran(false)
    Affiche_Barres_Outils
End Sub
```

Lancement des formulaires directement depuis l'ouverture de la base de données

Lorsque les barres d'outils ont disparu ou qu'un formulaire doit être affiché en mode plein écran, le fichier de base de données doit lancer le formulaire directement lors de son ouverture. Malheureusement, une simple commande pour ouvrir un formulaire ne fonctionnera pas, car la connexion à la base de données n'existe pas encore.

La macro suivante est lancée depuis **Outils> Personnaliser> Événements> Ouvrir le document**. Utilisez l'option **Enregistrer dans>** « Nom de la base.odt ».

```
Sub Formulaire_Direct
    Dim oSourceDonnees As Object
    oSourceDonnees = ThisDatabaseDocument.CurrentController
    If Not (oSourceDonnees.isConnected()) Then
        oSourceDonnees.connect()
    End If
    ThisDatabaseDocument.FormDocuments.getByName("MonFormulaire").open
End Sub
```

Tout d'abord, une connexion à la base de données doit être établie. Le contrôleur fait partie de ThisDatabaseDocument, tout comme le formulaire. Ensuite, le formulaire peut être lancé et lire ses données hors de la base de données.

Accéder à une base de données MySQL avec des macros

Toutes les macros présentées jusqu'à présent faisaient partie d'une base de données interne HSQLDB. Lorsque vous travaillez avec des bases de données externes, quelques modifications et extensions sont nécessaires.

Code MySQL dans les macros

Lors de l'accès à la base de données interne, les noms des tables et des champs doivent être placés entre guillemets doubles en double, par rapport au SQL :

```
SELECT "Champ" FROM "Table"
```

Comme ces commandes SQL doivent être préparées à l'intérieur de macros, les guillemets doivent être masqués :

```
stSQL = "SELECT ""Champ"" FROM ""Table"""
```

Les requêtes MySQL utilisent une forme différente de masquage :

```
SELECT `Champ` FROM `Database`.`Table`
```

À l'intérieur du code de macro, cette forme de masquage apparaît comme :

```
stSql = "SELECT `Champ` FROM `Database`.`Table`"
```

Tables temporaires comme stockage intermédiaire individuel

Dans le chapitre précédent, un tableau à une ligne était fréquemment utilisé pour rechercher ou filtrer les tables. Cela ne fonctionnera pas dans un système multi-utilisateur, car les autres utilisateurs seraient alors dépendants de la valeur de filtre de quelqu'un d'autre. Les tables temporaires dans MySQL ne sont accessibles qu'à l'utilisateur de la connexion active, donc ces tables sont accessibles pour la recherche et le filtrage.

Naturellement, de telles tables ne peuvent pas être créées à l'avance. Elles doivent être créées lors de l'ouverture du fichier Base. Par conséquent, la macro suivante doit être liée à l'ouverture du fichier *.odb.

```
Sub Creer_Table_Temporaire
    oSourceDonnees = thisDatabaseDocument.CurrentController
    If Not (oSourceDonnees.isConnected()) Then
        oSourceDonnees.connect()
    oConnexion = oSourceDonnees.ActiveConnection()
    oSQL_Instruction = oConnexion.createStatement()
    stSql = "CREATE TEMPORARY TABLE IF NOT EXISTS `tmpCherche` _
        (`ID` INT PRIMARY KEY, `Nom` VARCHAR(50))"
    oSQL_Instruction.executeUpdate(stSql)
End Sub
```

Lorsque le fichier *.odb est ouvert pour la première fois, il n'y a pas de connexion à une base de données MySQL externe. La connexion doit être créée. Ensuite, une table temporaire avec les champs nécessaires peut être mise en place.

Dialogues

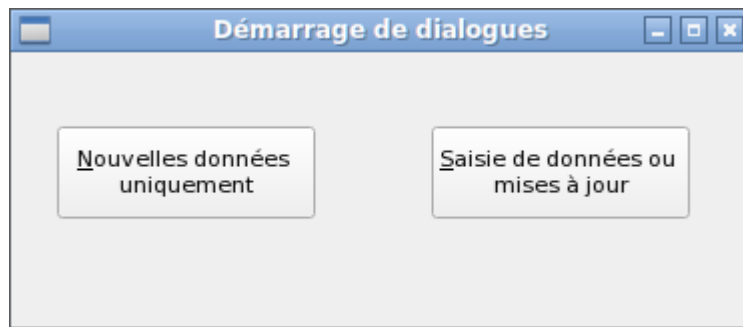
Dans Base (comme dans les autres modules), vous pouvez utiliser des boîtes de dialogue aussi bien que des formulaires pour la saisie de données, la modification de données ou la maintenance de la base de données. Les boîtes de dialogue peuvent être directement personnalisées pour l'environnement d'application actuel, mais elles ne sont naturellement pas définies à l'avance aussi facilement que les formulaires. Voici une brève introduction se terminant par un exemple assez complexe à utiliser dans la maintenance des bases de données²¹.

Lancement et fin des dialogues

La boîte de dialogue doit d'abord être créée. Pour ce faire, utilisez **Outils> Macros> Gérer les boîtes de dialogue> Nom de fichier de la base de données> Standard> Nouveau**. La boîte de dialogue apparaît avec une surface grise continue et une barre de titre avec une icône de fermeture. Cette boîte de dialogue vide peut maintenant être appelée puis refermée.

Il faut sélectionner la boîte de dialogue en cliquant sur le trait fin qui l'entoure. Il est alors possible sous les propriétés générales de définir une taille et une position. Son titre peut également être saisi.

21 Voir la base Exemple_Dialogues.odb, associée à ce manuel ainsi que le formulaire Maintenance de la base Media_avec_Macros.odb



La barre d'outils en bas de la fenêtre contient divers contrôles de formulaire. À partir de là, deux boutons ont été sélectionnés pour notre boîte de dialogue, lui permettant de lancer d'autres boîtes de dialogue. L'édition du contenu et la liaison des macros à des événements s'effectuent de la même manière que pour les boutons dans les formulaires.

Le positionnement des déclarations de variables pour les dialogues nécessite une attention particulière. La boîte de dialogue est déclarée comme une variable globale afin qu'elle soit accessible par différentes procédures. Dans ce cas, la boîte de dialogue s'appelle `oDialogue0`, car il y aura d'autres boîtes de dialogue avec des numéros de séquence plus élevés

```
Dim oDialogue0 As Object
```

La bibliothèque de la boîte de dialogue est d'abord chargée. Elle se trouve dans le catalogue Standard, si aucun autre endroit n'a été choisi lors de la création de la boîte de dialogue. Le dialogue lui-même peut être atteint dans cette bibliothèque en utilisant l'instruction **Dialogue0.Execute()**, qui lance la boîte de dialogue.

```
Sub DebutDialogue0
    DialogLibraries.LoadLibrary("Standard")
    oDialogue0 = createUnoDialog(DialogLibraries.Standard.Dialogue0)
    oDialogue0.Execute()
End Sub
```

En principe, une boîte de dialogue peut être fermée à l'aide du bouton Fermer sur le cadre. Cependant, si vous souhaitez un autre bouton spécifique pour cela, la commande **EndExecute()** doit être utilisée dans la procédure.

```
Sub FinDialogue0
    oDialogue0.EndExecute()
End Sub
```

Dans ce cadre, n'importe quel nombre de dialogues peuvent être lancés et refermés.

Boîte de dialogue simple pour saisir de nouveaux enregistrements

Cette boîte de dialogue est la première étape de la boîte de dialogue suivante pour la modification des enregistrements. Tout d'abord, l'approche de base de la gestion des tables est clarifiée. Nous

traitons ici du stockage des enregistrements avec de nouvelles clés primaires ou de la nouvelle entrée complète des enregistrements. Dans quelle mesure un petit dialogue comme celui-ci peut suffire pour entrer dans une base de données particulière dépend des exigences de l'utilisateur.

```
Dim oDialogue1 As Object
```

Crée directement une variable globale pour la boîte de dialogue au niveau supérieur du module avant toutes les procédures.

La boîte de dialogue s'ouvre et se ferme de la même manière que la boîte de dialogue précédente. Seul le nom passe de Dialogue0 à Dialogue1. La procédure de fermeture de la boîte de dialogue est liée au bouton **Quitter**.

Le bouton **Nouveau** permet d'effacer toutes les entrées précédentes des contrôles de la boîte de dialogue, à l'aide de la procédure **Efface_Champs_Donnees**.

```
Sub Efface_Champs_Donnees
    oDialogue1.getControl("ChampNumerique1").Text = ""
    oDialogue1.getControl("ChampTexte1").Text = ""
    oDialogue1.getControl("ChampTexte2").Text = ""
End Sub
```

Chaque champ inséré dans une boîte de dialogue est accessible par son nom. L'interface utilisateur garantira que les noms ne sont pas dupliqués, ce qui n'est pas le cas des contrôles dans un formulaire.

La méthode **getControl** est utilisée avec le nom du contrôle. Les champs numériques ont également une propriété Text qui peut être utilisée ici. C'est la seule façon de vider un champ numérique. Le texte vide existe, mais il n'y a pas de nombre vide. Au lieu de cela, un 0 doit être écrit dans le champ de clé primaire.

Le bouton **Enregistrer** lance la procédure **Enregistre_Donnees1** :

```
Sub Enregistre_Donnees1
    Dim oSourceDonnees As Object
    Dim oConnexion As Object
    Dim oSQL_Commande As Object
    Dim loID As Long
    Dim stPrenom As String
    Dim stNom As String
    loID = oDialogue1.getControl("ChampNumerique1").Value
    stPrenom = oDialogue1.getControl("ChampTexte1").Text
    stNom = oDialogue1.getControl("ChampTexte2").Text
    If loID > 0 And stNom <> "" Then
        oSourceDonnees = thisDatabaseDocument.CurrentController
        If Not (oSourceDonnees.isConnected()) Then
            oSourceDonnees.connect()
        End If
        oConnexion = oSourceDonnees.ActiveConnection()
        oSQL_Commande = oConnexion.createStatement()
        stSql = "SELECT ""ID"" FROM ""Noms"" WHERE ""ID"" = '"+loID+"'"
        oResultat = oSQL_Commande.executeQuery(stSql)
        While oResultat.next
            MsgBox ("la valeur pour le champ ""ID"" existe déjà",16, _
                "Valeur dupliquée")
            Exit Sub
        Wend
        stSql = "INSERT INTO ""Noms"" (""ID"", ""Prenom"", ""Nom"") _
            VALUES ('"+loID+"', '"+stPrenom+"', '"+stNom+"')"
        oSQL_Commande.executeUpdate(stSql)
        MsgBox "Enregistrement effectué"
        Efface_Champs_Donnees
    End If
End Sub
```

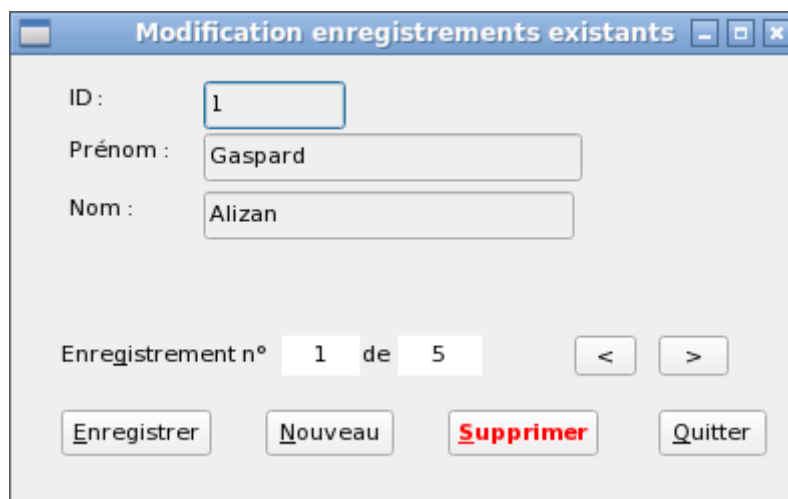
Comme dans la procédure `Efface_Champs_Donnees`, les champs de saisie sont accessibles. Cette fois, l'accès est en lecture uniquement. L'enregistrement ne sera transmis que si le champ ID a une entrée supérieure à 0 et que le champ Nom contient également du texte. Une valeur nulle pour l'ID peut être exclue car une variable numérique pour les nombres entiers est toujours initialisée à 0. Un champ vide est donc stocké avec une valeur nulle.

Si les deux champs ont été fournis avec du contenu, une connexion est établie avec la base de données. Comme les contrôles ne sont pas dans un formulaire, la connexion à la base de données doit être établie à l'aide de `thisDatabaseDocument.CurrentController`.

Tout d'abord, la base de données est interrogée pour voir si un enregistrement avec la clé primaire donnée existe déjà. Si cette requête produit un résultat, une boîte de message apparaît contenant un symbole d'arrêt (code : 16) et le message "Valeur dupliquée". Ensuite, la procédure se termine avec **Exit SUB**.

Si la requête ne trouve aucun enregistrement avec la même clé primaire, le nouvel enregistrement est inséré dans la base de données à l'aide de la commande SQL `INSERT`. Ensuite, la procédure `Efface_Champs_Donnees` est appelée pour fournir un nouveau formulaire vide.

Boîte de dialogue pour modifier les enregistrements dans une table



Ce dialogue offre clairement plus de possibilités que le précédent. Ici, tous les enregistrements peuvent être affichés et vous pouvez les parcourir, en créer de nouveaux ou supprimer des enregistrements. Naturellement, le code est beaucoup plus compliqué.

Le bouton **Quitter** est lié à la procédure, modifiée pour `Dialogue2`, qui a été décrite dans la boîte de dialogue précédente pour la saisie de nouveaux enregistrements. Ici, les boutons restants et leurs fonctions sont décrits.

La saisie des données dans la boîte de dialogue est limitée en ce que le champ ID doit avoir une valeur minimale de 1. Cette limitation concerne la gestion des variables en Basic : les variables numériques sont par définition initialisées à 0. Par conséquent, si les valeurs numériques des champs vides et ceux des champs contenant 0 sont lus, Basic ne peut détecter aucune différence entre eux. Cela signifie que si un 0 devait être utilisé dans le champ ID, il devrait d'abord être lu sous forme de texte et peut-être converti en nombre plus tard.

La boîte de dialogue est chargée dans les mêmes conditions qu'auparavant. Cependant, la procédure de chargement est rendue dépendante d'une valeur nulle pour la variable passée par la procédure `Charge_Donnees`.

```
Sub Charge_Donnees(loID As Long)
    Dim oSourceDonnees As Object
```

```

Dim oConnexion As Object
Dim oSQL_Commande As Object
Dim stPrenom As String
Dim stNom As String
Dim loLigne As Long
Dim loLigneMax As Long
Dim inDebut As Integer
oSourceDonnees = thisDatabaseDocument.CurrentController
If Not (oSourceDonnees.isConnected()) Then
    oSourceDonnees.connect()
End If
oConnexion = oSourceDonnees.ActiveConnection()
oSQL_Commande = oConnexion.createStatement()
If loID < 1 Then
    stSql = "SELECT MIN(""ID"") FROM ""Noms""
    oResultat = oSQL_Commande.executeQuery(stSql)
    While oResultat.next
        loID = oResultat.getInt(1)
    Wend
    inDebut = 1
End If

```

Les variables sont déclarées. La connexion à la base de données pour la boîte de dialogue est établie comme décrit ci-dessus. Au début, **loID** vaut **0**. Ce cas fournit la valeur de clé primaire la plus basse autorisée par SQL. L'enregistrement correspondant sera affiché ultérieurement dans la boîte de dialogue. Dans le même temps, la variable **inDebut** est définie sur **1**, de sorte que la boîte de dialogue puisse être lancée plus tard. Si la table ne contient aucun enregistrement, **loID** restera **0**. Dans ce cas, il ne sera pas nécessaire de rechercher le nombre et le contenu des enregistrements correspondants.

Ce n'est que si **loID** est supérieur à 0 que la requête testera les enregistrements disponibles dans la base de données. Ensuite, une deuxième requête comptera tous les enregistrements à afficher. La troisième requête donne la position de l'enregistrement actuel en comptant tous les enregistrements dont la clé primaire est inférieure ou égale à la clé primaire actuelle.

```

If loID > 0 Then
    stSql = "SELECT * FROM ""Noms"" WHERE ""ID"" = '"+loID+"'"
    oResultat = oSQL_Commande.executeQuery(stSql)
    While oResultat.next
        loID = oResultat.getInt(1)
        stPrenom = oResultat.getString(2)
        stNom = oResultat.getString(3)
    Wend
    stSql = "SELECT COUNT(""ID"") FROM ""Noms""
    oResultat = oSQL_Commande.executeQuery(stSql)
    While oResultat.next
        loLigneMax = oResultat.getInt(1)
    Wend
    stSql = "SELECT COUNT(""ID"") FROM ""Noms"" WHERE ""ID"" <= '"+loID+"'"
    oResultat = oSQL_Commande.executeQuery(stSql)
    While oResultat.next
        loLigne = oResultat.getInt(1)
    Wend
    oDialogue2.getControl("ChampNumerique1").Value = loID
    oDialogue2.getControl("ChampTexte1").Text = stPrenom
    oDialogue2.getControl("ChampTexte2").Text = stNom
End If
oDialogue2.getControl("ChampNumerique2").Value = loLigne
oDialogue2.getControl("ChampNumerique3").Value = loLigneMax
If loLigne = 1 Then
    ' ligne précédente
    oDialogue2.getControl("btnSupprimer").Model.enabled = False
Else
    oDialogue2.getControl("btnSupprimer").Model.enabled = True
End If
If loLigne <= loLigneMax Then

```

```

' ligne suivante | nouvelle ligne | effacer
oDialogue2.getControl("btnPrecedent").Model.enabled = True
oDialogue2.getControl("btnEnregistrer").Model.enabled = True
oDialogue2.getControl("btnSupprimer").Model.enabled = True
Else
oDialogue2.getControl("btnSuivant").Model.enabled = False
oDialogue2.getControl("btnEnregistrer").Model.enabled = False
oDialogue2.getControl("btnSupprimer").Model.enabled = False
End If
IF inDebut = 1 Then
oDialogue2.getControl("btnPrecedent").Model.enabled = false
oDialogue2.Execute()
End If
End Sub

```

Les valeurs récupérées sont transférées dans les champs de la boîte de dialogue. Les entrées pour le numéro d'enregistrement actuel et le nombre total d'enregistrements récupérés sont toujours écrites, remplaçant la valeur numérique par défaut de 0.

Les boutons de navigation (btnSuivant et btnPrecedent) ne sont utilisables que lorsqu'il est possible d'accéder à l'enregistrement correspondant. Sinon, ils sont temporairement désactivés avec **enabled = False**. Il en va de même pour les boutons btnNouveau et btnSupprimer. Ils ne doivent pas être disponibles lorsque le numéro de la ligne affichée est supérieur au nombre maximal de lignes déterminé. Il s'agit du paramètre par défaut de cette boîte de dialogue lors de la saisie d'enregistrements.

Si possible, la boîte de dialogue ne doit être lancée que lorsqu'elle doit être créée directement à partir d'un fichier de départ à l'aide de **Charge_Donnees (0)**. C'est pourquoi la variable spéciale **inDebut** reçoit la valeur 1 au début de la procédure.

Le bouton "<" est utilisé pour naviguer vers l'enregistrement précédent. Par conséquent, ce bouton n'est actif que lorsque l'enregistrement affiché n'est pas le premier de la liste. La navigation nécessite la lecture de la clé primaire de l'enregistrement actuel à partir du champ ChampNumerique1.

Ici, il y a deux cas possibles :

- 1) Vous passez à une nouvelle entrée, le champ correspondant n'a donc aucune valeur. Dans ce cas, **loID** a la valeur par défaut qui, selon la définition d'une variable entière, est 0.
- 2) Sinon, loID contiendra une valeur supérieure à 0. Une requête peut alors déterminer la valeur d'ID immédiatement inférieure à la valeur actuelle.

```

Sub Ligne_Precedente
Dim loID As Long
Dim loIDnouveau As Long
loID = oDialogue2.getControl("ChampNumerique1").Value
oSourceDonnees = thisDatabaseDocument.CurrentController
If Not (oSourceDonnees.isConnected()) Then
oSourceDonnees.connect()
End If
oConnexion = oSourceDonnees.ActiveConnection()
oSQL_Commande = oConnexion.createStatement()
If loID < 1 Then
stSql = "SELECT MAX(""ID"") FROM ""Noms"""
Else
stSql = "SELECT MAX(""ID"") FROM ""Noms"" WHERE ""ID"" < '"+loID+"'"
End If
oResultat = oSQL_Commande.executeQuery(stSql)
While oResultat.next
loIDnouveau = oResultat.getInt(1)
Wend
If loIDnouveau > 0 Then
Charge_Donnees(loIDnouveau)
End If

```

```

If loIDnouveau = 1 Then
    oDialogue2.getControl("btnPrecedent").Model.enabled = false
End If
oDialogue2.getControl("btnSuivant").Model.enabled = True
oDialogue2.getControl("btnNouveau").Model.enabled = True
End Sub

```

Si le champ ID est vide, l'affichage doit passer à la valeur la plus élevée du numéro de clé primaire. Si, en revanche, le champ ID fait référence à un enregistrement, la valeur précédente d'ID doit être renvoyée.

Le résultat de cette requête est utilisé pour exécuter à nouveau la procédure Charge_Donnees avec la valeur de clé correspondante.

Le bouton ">" est utilisé pour naviguer vers l'enregistrement suivant. Cette possibilité ne doit exister que lorsque la boîte de dialogue n'a pas été vidée pour la saisie d'un nouvel enregistrement. Ce sera naturellement le cas au lancement du dialogue et également avec une table vide.

Une valeur dans ChampNumerique1 est obligatoire. À partir de cette valeur, SQL peut déterminer quelle clé primaire est la suivante la plus élevée de la table. Si l'ensemble de résultats de la requête est vide, car il n'y a pas d'enregistrement correspondant, la valeur de **loIDnouveau = 0**. Sinon, le contenu de l'enregistrement suivant est lu à l'aide de Charge_Donnees.

```

Sub Ligne_Suivante
    Dim loID As Long
    Dim loIDnouveau As Long
    loID = oDialogue2.getControl("ChampNumerique1").Value
    oSourceDonnees = thisDatabaseDocument.CurrentController
    If Not (oSourceDonnees.isConnected()) Then
        oSourceDonnees.connect()
    End If
    oConnexion = oSourceDonnees.ActiveConnection()
    oSQL_Commande = oConnexion.createStatement()
    stSql = "SELECT MIN(""ID"") FROM ""Noms"" WHERE ""ID"" > '"+loID+'""
    oResultat = oSQL_Commande.executeQuery(stSql)
    While oResultat.next
        loIDnouveau = oResultat.getInt(1)
    Wend
    If loIDnouveau > 0 Then
        Charge_Donnees(loIDnouveau)
    Else
        Efface_Champs_Donnees2
    End If
End Sub

```

Si lors de la navigation vers l'enregistrement suivant, il n'y a pas d'autre enregistrement, la touche de navigation lance la procédure suivante Efface_Champs_Donnees2, qui sert à préparer l'entrée d'un nouvel enregistrement.

La procédure Efface_Champs_Donnees2 ne se contente pas de vider les champs de données eux-mêmes. La position de l'enregistrement en cours est définie sur une valeur supérieure au nombre maximum d'enregistrements, ce qui indique clairement que l'enregistrement en cours de traitement n'est pas encore inclus dans la base de données.

Dès qu'Efface_Champs_Donnees2 a été lancé, la possibilité de sauter à l'enregistrement précédent est activée, les sauts à l'enregistrement suivant et l'utilisation des procédures Nouveau et Suppression sont désactivées.

```

Sub Efface_Champs_Donnees2
    loLigneMax = oDialogue2.getControl("ChampNumerique3").Value
    oDialogue2.getControl("ChampNumerique1").Text = ""
    oDialogue2.getControl("ChampTexte1").Text = ""
    oDialogue2.getControl("ChampTexte2").Text = ""
    oDialogue2.getControl("ChampNumerique2").Value = loLigneMax + 1

```

```

oDialogue2.getControl("btnPrecedent").Model.enabled = True ' enreg. précédent
oDialogue2.getControl("btnSuivant").Model.enabled = False ' enreg. suivant
oDialogue2.getControl("btnNouveau").Model.enabled = False ' nouvel enreg.
oDialogue2.getControl("btnSupprimer").Model.enabled = False ' effacer
End Sub

```

L'enregistrement des données ne devrait être possible que lorsque les champs ID et Nom contiennent des entrées. Si cette condition est remplie, la procédure teste s'il s'agit d'un nouvel enregistrement. Cela utilise le pointeur d'enregistrement qui est défini pour que les nouveaux enregistrements soient un Nombre plus haut que le nombre maximum d'enregistrements

Pour les nouveaux enregistrements, des vérifications sont effectuées pour s'assurer que l'opération de sauvegarde réussira. Si le numéro utilisé pour la clé primaire a déjà été utilisé, un avertissement s'affiche. Si la réponse à la question associée est Oui, l'enregistrement existant avec ce numéro est écrasé. Sinon, la sauvegarde sera abandonnée. S'il n'y a pas d'entrées existantes dans la base de données (**loLigneMax = 0**), ce test n'est pas nécessaire et le nouvel enregistrement peut être sauvegardé directement. Pour un nouvel enregistrement, le nombre d'enregistrements est incrémenté de 1 et les entrées sont effacées pour l'enregistrement suivant.

Les enregistrements existants sont simplement écrasés par une commande de mise à jour.

```

Sub Enregistre_Donnees2(oEvent As Object)
    Dim oSourceDonnees As Object
    Dim oConnexion As Object
    Dim oSQL_Commande As Object
    Dim oDlg As Object
    Dim loID As Long
    Dim stPrenom As String
    Dim stNom As String
    Dim inMsg As Integer
    Dim loLigne As Long
    Dim loLigneMax As Long
    Dim stSql As String
    oDlg = oEvent. Source.getContext()
    loID = oDlg.getControl("ChampNumerique1").Value
    stPrenom = oDlg.getControl("ChampTexte1").Text
    stNom = oDlg.getControl("ChampTexte2").Text
    If loID > 0 And stNom <> "" Then
        oSourceDonnees = thisDatabaseDocument.CurrentController
        If Not (oSourceDonnees.isConnected()) Then
            oSourceDonnees.connect()
        End If
        oConnexion = oSourceDonnees.ActiveConnection()
        oSQL_Commande = oConnexion.createStatement()
        loLigne = oDlg.getControl("ChampNumerique2").Value
        loLigneMax = oDlg.getControl("ChampNumerique3").Value
        If loLigneMax < loLigne Then
            If loLigneMax > 0 Then
                stSql = "SELECT ""ID"" FROM ""Noms"" WHERE ""ID"" = '"+loID+"'"
                oResultat = oSQL_Commande.executeQuery(stSql)
                While oResultat.next
                    inMsg = MsgBox ("La valeur du champ "ID" existe déjà." & _
                        CHR(13) & "L'enregistrement doit-il être mis à jour?", 20, _
                        "Valeur dupliquée")
                    If inMsg = 6 Then
                        stSql = "UPDATE ""Noms"" SET ""Prenom""='"+stPrenom+"', _
                            ""Nom""='"+stNom+" WHERE ""ID"" = '"+loID+"'"
                        oSQL_Commande.executeUpdate(stSql)
                        ' Avec la mise à jour, une ligne a été réécrite
                        ' Le compte de lignes doit être réinitialisé
                        Charge_Donnees(loID)
                    End If
                End While
            End If
            Exit Sub
        End If
        stSql = "INSERT INTO ""Noms"" (""ID"", ""Prenom"", ""Nom"" ) VALUES

```

```

        ("'+loID+'','"+stPrenom+"','"+stNom+"')"
```

oSQL_Commande.executeUpdate(stSql)

oDlg.getControl("ChampNumerique3").Value = loLigneMax + 1

' Après l'insertion, une ligne est ajoutée

Efface_Champs_Donnees2

' Après l'insertion serait déplacé à l'insertion suivante

Else

stSql = "UPDATE ""Noms"" SET ""Prenom""='"+stPrenom+"', _

""Nom""='"+stNom+" WHERE ""ID"" = '"+loID+"'"

oSQL_Commande.executeUpdate(stSql)

End If

End If

End Sub

La procédure de suppression est fournie avec une question supplémentaire pour éviter les suppressions accidentelles. Puisque ce bouton est désactivé lorsque les champs de saisie sont vides, un ChampNumerique1 vide ne doit jamais se produire. Par conséquent, la condition de contrôle **IF loID > 0** peut être omise.

La suppression entraîne une décrémentation du nombre d'enregistrements de 1. Cela doit être corrigé à l'aide de **loLigneMax - 1**. Ensuite, l'enregistrement suivant celui en cours est affiché.

```

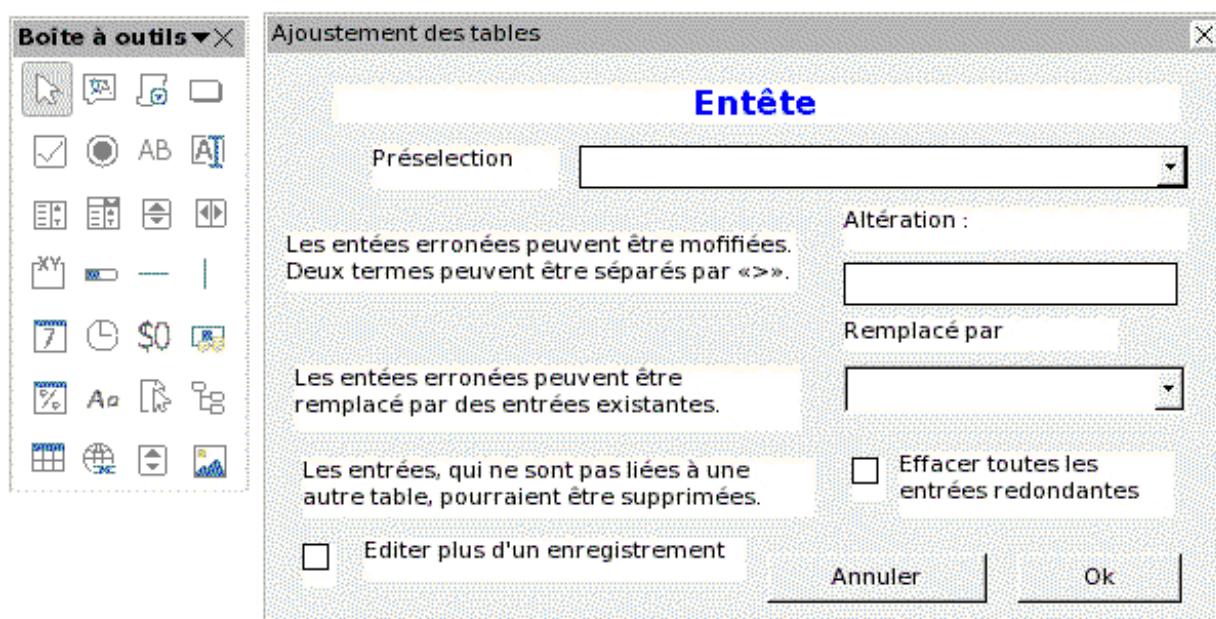
Sub Supprime_Ligne(oEvent As Object)
Dim oSourceDonnees As Object
Dim oConnexion As Object
Dim oSQL_Commande As Object
Dim oDlg As Object
Dim loID As Long
oDlg = oEvent.Source.getContext()
loID = oDlg.getControl("ChampNumerique1").Value
If loID > 0 Then
    inMsg = MsgBox ("Les données actuelles doivent-elles être supprimées ?",20, _
        "Supprimer la ligne actuelle")
    If inMsg = 6 Then
        oSourceDonnees = thisDatabaseDocument.CurrentController
        If Not (oSourceDonnees.isConnected()) Then
            oSourceDonnees.connect()
        End If
        oConnexion = oSourceDonnees.ActiveConnection()
        oSQL_Commande = oConnexion.createStatement()
        stSql = "DELETE FROM ""Noms"" WHERE ""ID"" = '"+loID+"'"
        oSQL_Commande.executeUpdate(stSql)
        loLigneMax = oDlg.getControl("ChampNumerique3").Value
        oDlg.getControl("ChampNumerique3").Value = loLigneMax - 1
        Ligne_Suivante
    End If
ELSE
    MsgBox ("Aucune ligne supprimée." & CHR(13) & _
        "Aucune donnée sélectionnée.",64,"Suppression impossible")
End If
End Sub
```

Cette petite boîte de dialogue a montré que l'utilisation du code macro peut fournir une base pour le traitement des enregistrements. L'accès via des formulaires est beaucoup plus facile, mais un dialogue peut être très flexible pour s'adapter aux exigences du programme. Cependant, il ne convient pas pour la création rapide d'une interface de base de données.

Utilisation d'une boîte de dialogue pour nettoyer les mauvaises données dans les tables

Les erreurs de saisie dans les champs ne sont souvent signalées que plus tard. Il est souvent nécessaire de modifier des entrées identiques dans plusieurs enregistrements en même temps. Il est gênant de devoir faire cela dans une vue de table normale, en particulier lorsque plusieurs enregistrements doivent être modifiés, car chaque enregistrement nécessite une entrée individuelle.

Les formulaires peuvent utiliser des macros pour faire ce genre de chose, mais pour le faire pour plusieurs tables, vous auriez besoin de formulaires construits de la même manière. Les dialogues peuvent faire le travail. Une boîte de dialogue peut être fournie au début avec les données nécessaires pour les tables appropriées et peut être appelée par plusieurs formulaires différents.



Les boîtes de dialogue sont enregistrées avec les modules pour les macros. Leur création s'apparente à celle d'un formulaire. Des champs de contrôle très similaires sont disponibles. Seul le contrôle de table des formulaires est absent en tant que possibilité d'entrée spéciale.

Ajoutement des tables

TypeMedia

Préselection: Livre > 21

Les entées erronées peuvent être modifiées. Deux termes peuvent être séparés par «>».

Altération, changement

Livre > 28

Remplacé par

Les entées erronées peuvent être remplacées par des entrées existantes.

Les entrées, qui ne sont pas liées à une autre table, pourraient être supprimées.

☐ Effacer toutes les entrées redondantes

☐ Editer plus d'un enregistrement

Annuler Ok

L'apparence des contrôles de boîte de dialogue est déterminée par les paramètres de l'interface utilisateur graphique.

La boîte de dialogue ci-dessus sert dans la base de données d'exemple pour éditer des tables qui ne sont pas utilisées directement comme base d'un formulaire. Ainsi, par exemple, le type de média n'est accessible que via une zone de liste (dans la version macro, il devient une zone combinée). Dans la version macro, le contenu du champ peut être étendu par un nouveau contenu mais une modification du contenu existant n'est pas possible. Dans la version sans macros, les modifications sont effectuées à l'aide d'un champ de table séparé.

Bien que les modifications dans ce cas soient faciles à effectuer sans macros, il est assez difficile de changer le type de support de nombreux supports à la fois. Supposons que les types suivants soient disponibles : « Livre, relié », « Livre, couverture rigide », « Livre broché » et « Reliure anneaux ». Maintenant, il s'avère, après que la base de données a été utilisée pendant une longue période, que des contemporains plus actifs prévoyaient des types de supports supplémentaires similaires pour les œuvres imprimées. La tâche de les différencier est devenue excessive. Nous souhaitons donc les réduire, de préférence à un seul terme. Sans macros, les enregistrements de la table des médias devraient être trouvés (à l'aide d'un filtre) et modifiés individuellement. Si vous connaissez SQL, vous pouvez le faire beaucoup mieux en utilisant une commande SQL. Vous pouvez modifier tous les enregistrements de la table Media avec une seule entrée. Une deuxième commande SQL supprime ensuite les types de média désormais excédentaires qui n'ont plus de lien vers la table Media. Cette méthode est précisément appliquée à l'aide de la boîte de dialogue Remplacer par – seule la commande SQL est d'abord adaptée à la table TypeMedia à l'aide d'une macro qui peut également modifier d'autres tables.

Souvent, les entrées se glissent dans une table qui, avec le recul, peut être modifiée dans le formulaire et ne sont donc plus nécessaires. Il ne fait aucun mal de simplement supprimer ces entrées orphelines, mais elles sont assez difficiles à trouver en utilisant l'interface utilisateur graphique. Là encore, une commande SQL appropriée est utile, associée à une instruction de suppression. Cette commande pour les tables concernées est incluse dans la boîte de dialogue sous *Supprimer toutes les entrées superflues*.

Si la boîte de dialogue doit être utilisée pour effectuer plusieurs modifications, cela est indiqué par la case à cocher Modifier plusieurs enregistrements. Ensuite, la boîte de dialogue ne se terminera pas simplement lorsque vous cliquerez sur le bouton OK.

Le code de macro associé à cette boîte de dialogue peut être consulté en entier dans la base de données d'exemple (. Seuls des extraits sont expliqués ci-dessous.

```
Sub AjustementTable(oEvent As Object)
```

La macro doit être lancée en entrant dans la section Informations complémentaires pour les boutons concernés :

```
0 : Formulaire, 1 : SousFormulaire, 2 : SousSousFormulaire, 3 : ZoneCombinée ou
contrôle de table, 4 : Champ de clé étrangère dans un formulaire, vide pour un champ
de table, 5 : Nom de table de la table auxiliaire, 6: Champ1 de la table auxiliaire,
7 : Champ de table 1 de la table auxiliaire, or 8 : Nom de table de la table
auxiliaire pour le champ2
```

Les entrées de cette zone sont répertoriées au début de la macro sous forme de commentaires. Les nombres qui leur sont liés sont transférés et l'entrée correspondante est lue à partir d'un tableau. La macro peut éditer des zones de liste, qui ont deux entrées, séparées par « > ». Ces deux entrées peuvent également provenir de tables différentes et être rassemblées à l'aide d'une requête, comme, par exemple, dans la table CodePostal, qui n'a que le champ de clé étrangère ID_Ville pour la ville, ce qui oblige la table Villes à afficher les noms des villes.

```
Dim aTablesEtrangeres(0, 0 to 1)
Dim aTablesEtrangeres2(0, 0 to 1)
```

Parmi les variables définies au début se trouvent deux tableaux. Alors que les tableaux normaux peuvent être créés par la commande **Split()** pendant l'exécution du sous-programme, les tableaux bidimensionnels doivent être définis à l'avance. Les tableaux à deux dimensions sont nécessaires pour stocker plusieurs enregistrements d'une requête lorsque la requête elle-même fait référence à plusieurs champs. Les deux tableaux déclarés ci-dessus doivent être capables d'interpréter les requêtes faisant référence à deux champs de table. Par conséquent, ils sont définis pour deux contenus différents en utilisant 0 à 1 pour la deuxième dimension.

```
stTag = oEvent.Source.Model.Tag
aTable() = Split(stTag, ", ")
For i = LBound(aTable()) To UBound(aTable())
    aTable(i) = trim(aTable(i))
Next
```

Les variables fournies sont lues. La séquence est celle mise en place dans le commentaire ci-dessus. Il y a un maximum de neuf entrées et vous devez déclarer s'il existe une huitième entrée pour le champ de table2 et une neuvième entrée pour une deuxième table.

Si des valeurs doivent être supprimées d'une table, il faut d'abord vérifier qu'elles n'existent pas en tant que clés étrangères dans une autre table. Dans les structures de table simples, une table donnée n'aura qu'une seule connexion de clé étrangère à une autre table. Cependant, dans l'exemple de base de données utilisé, il existe une table Villes qui est utilisée à la fois pour le lieu de publication des médias et la ville pour les adresses. Ainsi, la clé primaire de la table Villes est entrée deux fois dans différentes tables. Ces tables et noms de clés étrangères peuvent naturellement également être saisis à l'aide du champ Informations supplémentaires. Ce serait plus agréable si elles pouvaient être fournies universellement pour tous les cas. Cela peut être fait à l'aide de la requête suivante.

```
stSql = "SELECT ""FKTABLE_NAME"", ""FKCOLUMN_NAME"" FROM
""INFORMATION_SCHEMA"". ""SYSTEM_CROSSREFERENCE"" WHERE ""PKTABLE_NAME"" = ' ' +
aTable(5) + ' '"
```

Dans la base de données, la zone **INFORMATION_SCHEMA** contient toutes les informations sur les tables de la base de données, y compris des informations sur les clés étrangères. Les tables contenant ces informations sont accessibles à l'aide de **"INFORMATION_SCHEMA". "SYSTEM_CROSSREFERENCE". "PKTABLE_NAME"** donne la table qui fournit sa clé primaire

pour la connexion. **FKTABLE_NAME** donne la table qui utilise cette clé primaire comme clé étrangère. Enfin **FKCOLUMN_NAME** donne le nom du champ de clé étrangère.

La table qui fournit sa clé primaire à utiliser comme clé étrangère se trouve dans le tableau précédemment créé à la position 6. Comme le décompte commence par 0, la valeur est lue à partir du tableau à l'aide de **aTable(5)**.

```
inCompteur = 0
stIDEtrangerTab1Tab2 = "ID"
stIDEtrangerTab2Tab1 = "ID"
stTableSecondaire = aTable(5)
```

Avant que la lecture des tableaux ne commence, certaines valeurs par défaut doivent être définies. Il s'agit de l'index du tableau dans lequel les valeurs de la table auxiliaire seront écrites, de la clé primaire par défaut si nous n'avons pas besoin de la clé étrangère pour une deuxième table, et de la table auxiliaire par défaut, liée à la table principale, pour le code postal et ville, la table des codes postaux.

Lorsque deux champs sont liés pour être affichés dans une zone de liste, ils peuvent, comme décrit ci-dessus, provenir de deux tables différentes. Pour l'affichage du code postal et de la ville, la requête est :

```
SELECT "CodePostal"."CodePostal" || ' > ' || "Villes"."Ville" FROM
"CodePostal", "Villes" WHERE "CodePostal"."ID_Ville" = "Villes"."ID"
```

La table du premier champ (CodePostal), est liée à la deuxième table par une clé étrangère. Seules les informations de ces deux tables et des champs Code postal et Ville sont transmises à la macro. Toutes les clés primaires sont appelées par défaut ID dans l'exemple de base de données. La clé étrangère de Ville dans CodePostal doit donc être déterminée à l'aide de la macro.

De la même manière, la macro doit accéder à chaque table avec laquelle le contenu de la zone de liste est connecté par une clé étrangère.

```
oResultat_Requete = oSQL_Instruction.executeQuery(stSql)
If Not IsNull(oResultat_Requete) Then
    While oResultat_Requete.next
        ReDim Preserve aTablesEtrangeres(inCompteur, 0 to 1)
```

Le tableau doit être fraîchement dimensionné à chaque fois. Afin de préserver le contenu existant, il est sauvegardé à l'aide de (Preserve).

```
aTablesEtrangeres(inCompteur, 0) = oResultat_Requete.getString(1)
```

Lecture du premier champ avec le nom de la table qui contient la clé étrangère. Le résultat de la table CodePostal est la table Adresses.

```
aTablesEtrangeres(inCompteur, 1) = oResultat_Requete.getString(2)
```

Lecture du deuxième champ avec le nom du champ de clé étrangère. Le résultat de la table CodePostal est le champ ID_CodePostal dans la table Adresses.

Dans les cas où un appel au sous-programme inclut le nom d'une deuxième table, la boucle suivante est exécutée. L'entrée par défaut est modifiée uniquement lorsque le nom de la deuxième table apparaît comme table de clé étrangère pour la première table. Dans notre cas, cela ne se produit pas, car la table Villes n'a pas de clé étrangère de la table CodePostal. L'entrée par défaut de la table auxiliaire reste donc CodePostal. Enfin, la combinaison du code postal et de la ville sert de base à la table Adresses, qui contient une clé étrangère de la table CosePostal.

```
If UBound(aTable()) = 8 Then
    If aTable(8) = aTablesEtrangeres(inCount, 0) Then
        stIDEtrangerTab2Tab1 = aTablesEtrangeres(inCount, 1)
        stTableSecondaire = aTable(8)
    End If
End If
inCompteur = inCompteur + 1
```

Comme d'autres valeurs peuvent avoir besoin d'être lues, l'index est incrémenté pour redimensionner les tableaux. Puis la boucle se termine.

```
Wend  
End If
```

Si, lors de l'appel du sous-programme, un deuxième nom de table existe, la même requête est lancée pour cette table :

```
If UBound(aTable()) = 8 Then
```

Il s'exécute de la même manière, sauf que la boucle teste si le premier nom de table apparaît comme un nom de table de clé étrangère. C'est le cas ici : la table CodePostal contient la clé étrangère ID_Ville de la table Villes. Cette clé étrangère est maintenant affectée à la variable stForeignIDTab1Tab2, afin que la relation entre les tables puisse être définie.

```
If aTable(5) = aTablesEtrangeres2(inCount, 0) Then  
    stIDEtrangerTab1Tab2 = aTablesEtrangeres2(inCompteur, 1)  
End If
```

Après quelques réglages supplémentaires pour garantir un retour à la forme correcte après l'exécution de la boîte de dialogue (détermination du numéro d'enregistrement du formulaire, afin que nous puissions revenir à ce numéro de ligne après une nouvelle lecture), la boucle commence, ce qui recrée la boîte de dialogue lorsque la première action est terminée mais que la boîte de dialogue doit rester ouverte pour d'autres actions. Le réglage de la répétition s'effectue à l'aide de la case à cocher correspondante.

```
Do
```

Avant que la boîte de dialogue ne soit lancée, tout d'abord le contenu des zones de liste est déterminé. Des précautions doivent être prises si les zones de liste représentent deux champs de table et peuvent même être liés à deux tables différentes.

```
If UBound(aTable()) = 6 Then
```

La listbox ne concerne qu'une seule table et un seul champ, car le tableau d'arguments se termine à Tablefield1 de la table auxiliaire..

```
stSql = "SELECT "" + aTable(6) + "" FROM "" + aTable(5) + _  
"" ORDER BY "" + aTable(6) + ""  
ElseIf UBound(aTable()) = 7 Then
```

La zone de liste concerne deux champs de table mais une seule table, car le tableau d'arguments se termine à Tablefield2 de la table auxiliaire.

```
stSql = "SELECT "" + aTable(6) + "" || ' > ' || "" + aTable(7) + "" FROM "" _  
+ aTable(5) + "" ORDER BY "" + aTable(6) + ""  
Else
```

La zone de liste est basée sur deux champs de table provenant de deux tables. Cette requête correspond à l'exemple avec le code postal et la ville.

```
stSql = "SELECT "" + aTable(5) + ""." + aTable(6) + _  
"" || ' > ' || "" + aTable(8) + ""." + aTable(7) + "" FROM "" + _  
aTable(5) + "", "" + aTable(8) + "" WHERE "" + aTable(8) + _  
""." + stIDEtrangerTab2Tab1 + "" = "" + aTable(5) + ""." + _  
stIDEtrangerTab1Tab2 + "" ORDER BY "" + aTable(6) + ""  
End If
```

Ici, nous avons la première évaluation pour déterminer les clés étrangères. Les variables stIDEtrangerTab2Tab1 et stIDEtrangerTab1Tab2 commencent par l'ID de valeur. Pour stIDEtrangerTab1Tab2, l'évaluation de la requête précédente donne une valeur différente, à savoir la valeur de ID_Ville. De cette manière, la construction de la requête précédente donne exactement le contenu déjà formulé pour le code postal et la ville – uniquement amélioré par le tri.

Il faut maintenant prendre contact avec les zones de liste, pour leur fournir le contenu renvoyé par les requêtes. Ces zones de liste n'existent pas encore, car la boîte de dialogue elle-même n'a pas

encore été créée. Cette boîte de dialogue est d'abord créée en mémoire, en utilisant les lignes suivantes, avant d'être réellement dessinée à l'écran.

```
DialogLibraries.LoadLibrary("Standard")
oDlg = CreateUnoDialog(DialogLibraries.Standard.Dialogue_Ajustement_Tables)
```

Viennent ensuite les paramètres des champs de la boîte de dialogue. Voici, par exemple, la listbox qui doit être fournie avec les résultats de la requête ci-dessus :

```
oCtlListe1 = oDlg.GetControl("ZoneListe1")
oCtlListe1.addItem(acontenu(),0)
```

L'accès aux champs de la boîte de dialogue s'effectue à l'aide de GetControl avec le nom approprié. Dans les boîtes de dialogue, il n'est pas possible que deux champs utilisent le même nom car cela créerait des problèmes lors de l'évaluation de la boîte de dialogue.

La zone de liste est fournie avec le contenu de la requête, qui a été stocké dans le tableau acontenu(). La zone de liste contient uniquement le contenu à afficher sous forme de champ, donc seule la position 0 est remplie.

Une fois que tous les champs avec le contenu souhaité ont été remplis, la boîte de dialogue s'ouvre.

```
Select Case oDlg.Execute()
Case 1 ' Le cas 1 signifie que le bouton "OK" a été cliqué
Case 0 ' S'il s'agissait du bouton "Annuler"
    inRepete = 0
End Select
Loop While inRepete = 1
```

La boîte de dialogue s'exécute de manière répétée tant que la valeur de "inRepete" est 1. Ceci est défini par la case à cocher correspondante.

Voici, en bref, le contenu après avoir cliqué sur le bouton "OK":

```
Case 1
stContenu1 = String_to_SQL(oCtlListe.getSelectedItemAt())
'Lire la valeur de ZoneListe1... et déterminer la valeur ID correspondante.
```

La valeur ID de la première listbox est stockée dans la variable "inZC1".

```
stTexte = oCtlTexte.Text ' Lire la valeur du champ.
```

Si le champ de texte n'est pas vide, l'entrée du champ de texte est gérée. Ni la zone de liste pour une valeur de remplacement ni la case à cocher pour supprimer tous les enregistrements orphelins ne sont prises en compte. Ceci est rendu clair par le fait que la saisie de texte définit ces autres champs comme inactifs.

```
If stTexte <> "" Then
```

Si le champ de texte n'est pas vide, la nouvelle valeur est écrite à la place de l'ancienne en utilisant le champ ID précédemment lu dans la table. Il y a la possibilité de deux entrées, comme c'est aussi le cas dans la zone de liste. Le séparateur est ">". Pour deux entrées dans des tables différentes, deux commandes UPDATE doivent être lancées, qui sont créées ici simultanément et transmises, séparées par un point-virgule.

```
ElseIf oCtlListe2.getSelectedItemAt() <> "" Then
```

Si le champ de texte est vide et que la zone de liste 2 contient une valeur, la valeur de la listbox 1 doit être remplacée par la valeur de la zone de liste 2. Cela signifie que tous les enregistrements des tables pour lesquels les enregistrements des zones de liste sont des clés étrangères doivent être vérifiés et, si nécessaire, écrit avec une clé étrangère modifiée.

```
stContenu2 = oCtlListe2.getSelectedItemAt()
REM Lire la valeur de la liste.
REM Déterminer l'ID pour la valeur de la liste.
```

La valeur ID de la deuxième zone de liste est stockée dans la variable inZC2. Ici aussi, les choses évoluent différemment selon que un ou deux champs sont contenus dans la zone de liste, et aussi selon que une ou deux tables sont à la base du contenu de la zone de liste.

Le processus de remplacement dépend de la table définie comme table fournissant la clé étrangère pour la table principale. Pour l'exemple ci-dessus, il s'agit de la table CodePostal, car ID_CodePostal est la clé étrangère qui est transmise via Zone de liste 1 et Zone de liste 2.

```
If stTableSecondaire = aTable(5) Then
    For i = LBound(aTablesEtrangeres()) To UBound(aTablesEtrangeres())
```

Le remplacement de l'ancienne valeur ID par la nouvelle valeur ID devient problématique dans les relations n : m, car dans de tels cas, la même valeur peut être affectée deux fois. C'est peut-être ce que vous souhaitez, mais cela doit être évité lorsque la clé étrangère fait partie de la clé primaire. Ainsi, dans la table relation_Media_Auteur, un support ne peut pas avoir le même auteur deux fois, car la clé primaire est construite à partir de ID_Media et ID_Auteur. Dans la requête, tous les champs de clé sont recherchés qui ont collectivement la propriété UNIQUE ou ont été définis comme des clés étrangères avec la propriété UNIQUE à l'aide d'un index.

Donc, si la clé étrangère a la propriété UNIQUE et y est déjà représentée avec le futur souhaité inZC2, cette clé ne peut pas être remplacée.

```
stSql = "SELECT ""COLUMN_NAME"" FROM ""INFORMATION_SCHEMA"". _
""SYSTEM_INDEXINFO"" WHERE ""TABLE_NAME"" = ' " + aTablesEtrangeres(i, 0) + "' AND _
""NON_UNIQUE"" = False AND ""INDEX_NAME"" = (SELECT ""INDEX_NAME"" FROM _
""INFORMATION_SCHEMA"". ""SYSTEM_INDEXINFO"" WHERE ""TABLE_NAME"" = ' " + _
+ aTablesEtrangeres(i, 0) + "' AND ""COLUMN_NAME"" = ' " + aTablesEtrangeres(i, 1) + _
+ "')"
```

""NON_UNIQUE"" = False donne les noms des colonnes UNIQUE. Cependant, tous les noms de colonnes ne sont pas nécessaires, mais uniquement ceux qui forment un index avec le champ de clé étrangère. Ceci est géré par la sous-sélection avec les mêmes noms de table (qui contiennent la clé étrangère) et les noms des champs de clé étrangère.

Si maintenant la clé étrangère est présente dans l'ensemble, la valeur de clé ne peut être remplacée que si d'autres champs sont utilisés pour définir l'index correspondant comme UNIQUE. Lorsque vous effectuez des remplacements, vous devez veiller à ce que l'unicité de la combinaison d'indices ne soit pas compromise.

```
If aTablesEtrangeres(i, 1) = stNomChamp Then
    inUnique = 1
Else
    ReDim Preserve aColonnes(inCount)
    aColonnes(inCount) = oQuery_result.getString(1)
    inCompteur = inCompteur + 1
End If
```

Tous les noms de colonne, à l'exception des noms de colonne connus pour les champs de clé étrangère comme Index avec la propriété UNIQUE, sont stockés dans le tableau. Comme le nom de colonne du champ de clé étrangère appartient également au groupe, il peut être utilisé pour déterminer si l'unicité doit être vérifiée lors de la modification des données.

```
If inUnique = 1 Then
    stSql = "UPDATE "" " + aTablesEtrangeres(i, 0) + "" AS ""a"" SET "" " + _
aTablesEtrangeres(i, 1) + ""="" + inZC2 + "" WHERE "" " + _
aTablesEtrangeres(i, 1) + ""="" + inZC1 + "" AND (SELECT COUNT(*) FROM "" " + _
aTablesEtrangeres(i, 0) + "" WHERE "" " + aTablesEtrangeres(i, 1) + _
""="" + inZC2 + "")"
    If inCompteur > 0 Then
        stGroupeChamp = Join(aColonnes(), "" || "" || """)
```

S'il existe plusieurs champs, en dehors du champ de clé étrangère, qui forment ensemble un index UNIQUE, ils sont ici combinés pour un regroupement SQL. Sinon, seul aColonnes(0) apparaît comme stFieldgroup.

```

stNomDuChamp = ""
For ink = LBound(aColonnes()) To UBound(aColonnes())
    stNomDuChamp = stNomDuChamp + " AND " + aColonnes(ink) + " = _
    ""a"". "" + aColonnes(ink) + " "

```

Les parties SQL sont combinées pour une sous-requête corrélée.

```

Next ink
stSql = Left(stSql, Len(stSql) - 1)

```

La requête précédente se termine par un crochet. À présent, du contenu supplémentaire doit être ajouté à la sous-requête, cette fermeture doit donc être supprimée à nouveau. Après cela, la requête est développée avec les conditions supplémentaires.

```

stSql = stSql + stNomDuChamp + "GROUP BY ("" + stGroupeChamp + "") < 1"
End If

```

Si la clé étrangère n'a pas de connexion avec la clé primaire ou avec un index UNIQUE, peu importe si le contenu est dupliqué.

```

Else
    stSql = "UPDATE "" + aTablesEtrangeres(i, 0) + "" SET "" + _
    aTablesEtrangeres(i, 1) + ""=' + inZC2 + "' WHERE "" + _
    aTablesEtrangeres(i, 1) + ""=' + inZC1 + "'"
End If
oSQL_Commande.executeQuery(stSql)
NEXT

```

La mise à jour est effectuée aussi longtemps que différentes connexions à d'autres tables se produisent, autrement dit, tant que la table actuelle est la source d'une clé étrangère dans une autre table. C'est le cas deux fois pour la table Villes : dans la table Média et dans la table CodePostal.

Ensuite, l'ancienne valeur peut être supprimée de la zone de liste 1, car elle n'a plus de connexion avec d'autres tables.

```

stSql = "DELETE FROM "" + aTable(5) + "" WHERE ""ID""=' + inZC1 + "'"
oSQL_Commande.executeQuery(stSql)

```

Dans certains cas, la même méthode doit maintenant être utilisée pour une deuxième table qui a fourni des données pour les zones de liste. Dans notre exemple, la première table est la table CodePostal et la seconde est la table Villes.

Si le champ de texte est vide et que la zone de liste 2 ne contient également rien, nous vérifions si la case à cocher correspondante indique que toutes les entrées excédentaires doivent être supprimées. Cela signifie les entrées qui ne sont pas liées à d'autres tables par une clé étrangère.

```

ElseIf oCtlCoche1.State = 1 Then
    stCondition = ""
    If stTableSecondaire = aTable(5) Then
        For i = LBound(aForeignTables()) To UBound(aTablesEtrangeres())
            stCondition = stCondition + ""ID"" NOT IN (SELECT "" + _
            aTablesEtrangeres(i, 1) + "" FROM "" + aTablesEtrangeres(i, 0) + "") AND "
        Next
    Else
        For i = LBound(aForeignTables2()) To UBound(aTablesEtrangeres2())
            stCondition = stCondition + ""ID"" NOT IN (SELECT "" + _
            aTablesEtrangeres2(i, 1) + "" FROM "" + aForeignTables2(i, 0) + "") AND "
        Next
    End If

```

Le dernier AND doit être supprimé, sinon l'instruction de suppression se terminerait par AND.

```

stCondition = Left(stCondition, Len(stCondition) - 4)
stSql = "DELETE FROM "" + stTableSecondaire + "" WHERE " + stCondition + ""
oSQL_Commande.executeQuery(stSql)

```

Comme la table a déjà été purgée une fois, l'index de la table peut être vérifié et éventuellement corrigé vers le bas. Voir le sous-programme décrit dans l'une des sections précédentes.

```
MinimiseIndexTable(stTableSecondaire)
```

Ensuite, si nécessaire, la zone de liste sous la forme à partir de laquelle la boîte de dialogue Table_purge a été appelée peut être mise à jour. Dans certains cas, le formulaire entier doit être relu. Dans ce but, l'enregistrement en cours est déterminé au début du sous-programme de sorte qu'après l'actualisation du formulaire, l'enregistrement en cours puisse être rétabli.

```
oDlg.EndExecute() 'Fermeture dialogue...
oDlg.Dispose() '... et retirer de la mémoire
End Sub
```

Les boîtes de dialogue se terminent avec la commande endExecute() et sont complètement supprimées de la mémoire avec Dispose().

Écrire des macros avec Access2Base

Les versions de LibreOffice à partir de 4.2 ont intégré Access2Base. Cette bibliothèque introduit une couche de base avec son API spécifique (Application Programming Interface) entre le code de l'utilisateur et l'interface UNO habituelle. L'API fournie n'apporte pas en soi de nouvelles fonctionnalités mais, dans de nombreux cas, elle est plus lisible, concise et plus facile à utiliser que UNO.

L'API ressemble beaucoup à celle conçue par Microsoft pour le logiciel Access. Base et Access ont beaucoup en commun, mais certainement pas leurs styles de programmation natifs. Access2Base comble le vide.

Une documentation en anglais avec des exemples est disponible sur <http://www.access2base.com/access2base.html>

Pour illustrer brièvement comment Access2Base cache la complexité de UNO :

- La propriété (Access2Base simple) **Value** d'un contrôle a dans UNO comme équivalents, selon le type de contrôle ou son emplacement dans un formulaire, un contrôle de grille ou une boîte de dialogue : *CurrentValue*, *Date*, *EffectiveValue*, *HiddenValue*, *ProgressValue*, *RefValue*, *ScrollValue*, *SpinValue*, *State*, *StringItemList*, *Text*, *Time*, *ValueItemList* ou... *Value*.
- Pour obtenir les N premiers enregistrements d'une table ou d'une requête dans un tableau de base, une méthode consiste simplement à utiliser la méthode *GetRows(N)* sur un objet *Recordset*. Comparez avec les méthodes *getString*, *getNull*, *getDouble*, *getLong*... dans UNO que vous devez appliquer aux champs en fonction de leur type et du système de base de données utilisé.

Il existe deux catégories principales d'objets gérés par Access2Base, ciblant soit :

- L'interface utilisateur. Les classes d'objets typiques sont : *Form*, *SubForm*, *Dialog*, *Control*, *CommandBar*, *CommandBarControl* et *Event*. Leurs méthodes sont généralement appelées à partir d'une application de base.
- La base de données accédée. Les classes d'objets typiques sont : *Database*, *TableDef*, *QueryDef*, *Recordset* et *Field*. Leurs méthodes sont appelées soit à partir d'une application Base, soit à partir de toute autre application LibreOffice.

Traditionnellement, l'API Access2Base est appelée à partir du langage de base. À partir de LibreOffice 6.4, une passerelle permet également d'accéder à l'API à partir de scripts Python, sans aucune limitation par rapport à Basic. On peut intégrer de manière transparente des scripts Basic et Python dans la même application, même en partageant les mêmes instances d'objet.

Dans les paragraphes suivants, chaque exemple sera donné à la fois en Basic et en Python. Ils sont strictement équivalents.

Pour accéder à la bibliothèque à partir d'une application Base, attachez la procédure suivante à l'événement OpenDocument de votre fichier de base :

(BASIC)

```
Sub OuvreBDD(Optional oEvent As Object)
    If GlobalScope.BasicLibraries.HasByName("Access2Base") Then
        GlobalScope.BasicLibraries.LoadLibrary("Access2Base")
    End If
    Call Application.OpenConnection(ThisDatabaseDocument)
End Sub
```

(PYTHON)

```
from access2base import *
def OuvreBDD(event = None):
    Application.OpenConnection()
    g_exportedScripts = (DBOpen,)
```

Sinon, pour accéder à la base de données à partir d'une application autre Base, exécutez :

(BASIC)

```
Function OuvreBDD() As Object
    If GlobalScope.BasicLibraries.HasByName("Access2Base") Then
        GlobalScope.BasicLibraries.LoadLibrary("Access2Base")
    End If
    Set maBDD = Application.OpenDatabase("... Nom de la base de données... ")
End Function
```

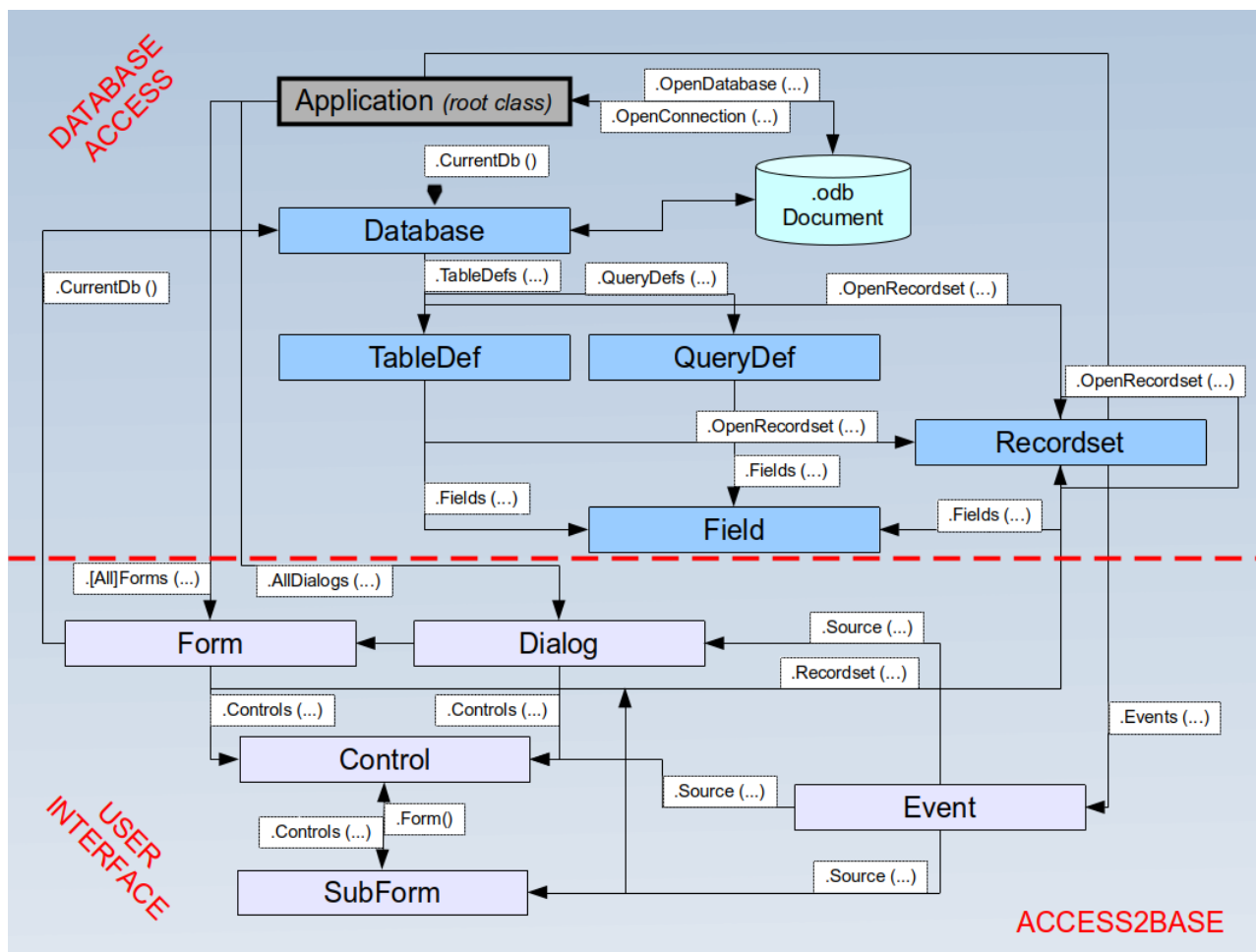
(PYTHON)

```
from access2base import *
def OuvreBDD():
    return Application.OpenDatabase('... Nom de la base de données... ')
```

Le but de ce manuel n'est pas de reproduire la documentation du site Web susmentionné. Nous limiterons ce document à un résumé des principaux concepts de l'API.

Le modèle Objet

Ci-dessous, à partir de l'objet racine Application (*Application root object*), se trouve un schéma décrivant la navigation à travers les objets les plus utilisés :



À titre d'exemple, pour aider à lire le schéma :

- **Application** est le principal des deux objets racine.
- Les méthodes `CurrentDb()` et `OpenDatabase()` fournissent un objet **Database**.
- La collection `Database.TableDefs()` répertorie toutes les tables stockées dans la base de données. Chaque table est représentée par une instance d'objet **TableDef**.
- La collection `TableDef.Fields()` répertorie tous les champs présents dans la table de base de données. Chaque champ est représenté par une instance d'objet **Field**.
- Les champs peuvent également être les enfants de requêtes stockées (**QueryDefs**) ou d'ensembles de données (**Recordsets**), avec les mêmes attributs.

Quelques exemples

Imprimer une liste de noms de tables et de champs

(BASIC)

```
Sub ScanTables()
Dim oBaseDonnees As Object, oTable As Object, oChamp As Object
Dim i As Integer, j As Integer
Set oBaseDonnees = Application.CurrentDb()
With oBaseDonnees
For i = 0 To .TableDefs.Count - 1
Set oTable = .TableDefs(i) ' Obtenir la définition de chaque table
DebugPrint oTable.Name
```

```

        For j = 0 To oTable.Fields.Count - 1
            Set oChamp = oTable.Fields(j) ' Obtenir chaque champ individuellement
            DebugPrint "", oChamp.Name, oChamp.TypeName
        Next j
    Next i
End With
End Sub

```

(PYTHON)

```

def ScanTables():
    oBaseDonnees = Application.OpenDatabase("/home/somedir/TT NorthWind.odt")
    for oTable in oBaseDonnees.TableDefs():
        DebugPrint(oTable.Name)
        for oField in oTable.Fields():
            DebugPrint("", oField.Name, oField.TypeName)

```

Stocker les données produites par une requête dans un tableau Base ou un tuple Python

(BASIC)

```

Sub ChargeRequete()
    Dim oEnregistrements As Object, vDonnees As Variant
    Set oEnregistrements = Application.CurrentDb().OpenRecordset("maRequete")
    vDonnees = oEnregistrements.GetRows(1000)
    oEnregistrements.mClose()
End Sub

```

(PYTHON)

```

def ChargeRequete():
    oEnregistrements= Application.CurrentDb().Openrecordset("maRequete")
    vData = oEnregistrements.GetRow(1000)
    oEnregistrements.Close()

```

Définir les valeurs par défaut dans les entrées de formulaire

Pour spécifier qu'après chaque entrée d'enregistrement, un contrôle est pré-rempli avec la dernière valeur définie, affectez la routine suivante à l'événement *Après le changement d'enregistrement* du formulaire :

(BASIC)

```

Sub SetDefaultNewRec(poEvent As Object)
    Dim oFormulaire As Object, oControl As Object
    Set oFormulaire = Application.Events(poEvent).Source ' Obtenir le formulaire
    actuel
    Set oControle = oFormulaire.Controls("txtPays")
    oControle.DefaultValue = oControle.Value
End Sub

```

(PYTHON)

```

def SetDefaultNewRec(poEvent):
    oFormulaire = Application.Events(poEvent).Source
    oControl = oForm.Controls("txtCountry")
    oControl.DefaultValue = oControl.Value

```

Fonctions de base de données

Un ensemble de fonctions est fourni pour raccourcir à une seule ligne l'accès aux valeurs de la base de données : *DLookup*, *DMax*, *DMin*, *Dsum*. Ils acceptent tous les mêmes arguments : un nom de champ ou une expression basée sur des noms de champ, un nom de table ou de requête et une clause SQL-where sans le mot clé *WHERE*. Par exemple :

(BASIC)

```

Function Lookup(psField As String, psSearchField As String, _

```

```

        psSearchValue As String) As Variant
Lookup = Application.DLookup(psField, "maTable", psSearchField & "=" _
    & psSearchValue & "'")
End Function

```

(PYTHON)

```

def Lookup(psField, psSearchField, psSearchValue):
    return Application.Dlookup(psField, "myTable"
        , psSearchField + "=" + psSearchValue + "'")

```

Commandes spéciales

Le DoCmd (2e classe racine) propose un ensemble de fonctions pratiques permettant d'exécuter en une instruction Basic des actions complexes bien que fréquentes et pratiques. Pour n'en nommer que quelques-unes :

CopyObject	Copie une table ou une requête dans la même base de données ou entre deux bases de données.
OpenSQL	Exécute une instruction SQL SELECT donnée et affiche le résultat dans une feuille de données.
OutputTo	Stocke les données d'une table ou d'une requête dans un fichier HTML. Stocke le contenu réel d'un formulaire dans un fichier PDF.
SelectObject	Activer la fenêtre donnée (formulaire, rapport,...)
SendObject	Envoyer par mail avec le formulaire donné en pièce jointe.

L'objet "Basic" en Python

À partir de Python, un objet supplémentaire a été introduit dans la passerelle Access2Base, simplement appelé « Basic », pour permettre d'exécuter avec la bibliothèque BASIC un certain nombre de fonctions intégrées de base bien connues. Elles sont invoquées exactement avec les mêmes arguments et se comportent strictement de la même manière dans les deux environnements. Parmi elles :

- Fonctions système : Basic.ConvertToUrl and Basic.ConvertFromUrl, Basic.GlobalScope, Basic.CreateUnoService.
- Fonctions de l'interface utilisateur : Basic.MsgBox, Basic.InputBox
- Fonctions de date : Basic.DateAdd, Basic.DateDiff
- Introspection d'objets UNO : Basic.Xray

Index des Procédures et Fonctions

Function ID_Determination(oChamp As Object) As Integer	
Chapitre : Renvoyer la valeur des zones de liste	
Base exemple :	47
Function Lookup(psField As String, psSearchField As String, psSearchValue As String) As Variant	
Chapitre : Fonctions de base de données	
Base exemple :	126
Function OfficeVersion()	
Chapitre : Fournir la version actuelle de LibreOffice	
Base exemple :	46
Function OuvreBDD() As Object	
Chapitre : Écrire des macros avec Access2Base	
Base exemple :	123
Function String_to_SQL(st As String)	
Chapitre : Préparation des données à partir de champs de texte pour respecter les conventions SQL	
Base exemple : BoiteCombinee_ListeChamps.....	41
Function TailleColonne(TableName As String, Fieldname As String) As Integer	
Chapitre : Fonction pour mesurer la longueur de l'entrée zone combinée	
Base exemple : BoiteCombinee_ListeChamps.....	69
Function ValeurDate(oChamp As Object) As Date	
Chapitre : Conversion de dates d'un formulaire en une variable de date	
Base exemple : DateFormulaire_en_Variable.....	51
Sub aContenuRequete	
Chapitre : Accès aux requêtes	
Base exemple :	84
Sub Activer_Courriel	
Chapitre : Appeler un programme de messagerie avec un contenu prédéfini	
Base exemple : Activer_Fichier_Courriel.....	96
Sub Activer_SiteWeb_ou_Courriel	
Chapitre : Appeler des applications pour ouvrir des fichiers	
Base exemple : Activer_Fichier_Courriel.....	95
Sub Affiche_Barres_Outils	
Chapitre : Formulaires sans barre d'outils dans la fenêtre	
Base exemple : Formulaire_a_formulaire.....	99
Sub AfficheTexte(oEvent As Object)	
Chapitre : Affichage du texte dans les boîtes combinées	
Base exemple : BoiteCombinee_ListeChamps.....	60
Sub Cache_Barres_Outils	
Chapitre : Formulaires sans barre d'outils dans la fenêtre	
Base exemple : Formulaire_a_formulaire.....	98
Sub CalcDataImport(oEvent As Object)	
Chapitre : Importation directe des données de Calc	
Base exemple : Exemple_Import_Données_Calc.....	82
Sub Calcul_Prix_TTC(oEvent As Object)	
Chapitre : Calculs directs dans un formulaire	

Base exemple : Calcul_Direct_Formulaire.....	44
Sub Calcul_Totaux(oEvent As Object)	
Chapitre : Calculs directs dans un formulaire	
Base exemple : Calcul_Direct_Formulaire.....	45
Sub ChangeData(oEvent As Object)	
Chapitre : Enregistrer des commandes SQL et le exécuter à la demande.	
Base Exemple : Exemple_InsertUpdateDelete_SQL.odt.....	41
Sub Charge_Donnees(loiD As Long)	
Chapitre : Boîte de dialogue pour modifier les enregistrements dans une table	
Base exemple : Dialogues.....	104
Sub ChargeRequete()	
Chapitre : Stocker les données produites par une requête dans un tableau Base ou un tuple Python	
Base exemple :	125
Sub Compactage_BaseDeDonnees	
Chapitre : Compactage de la base de données	
Base exemple :	89
Sub Confirmation_Enregistre(oEvent As Object)	
Chapitre : Enregistrer avec confirmation	
Base exemple :	77
Sub Controles_Hierarchiques(oEvent As Object)	
Chapitre : Zones de liste hiérarchiques	
Base exemple : Recherche_et_Filtre.....	73
Sub CopieDonnees	
Chapitre : Copie de données d'une base de données à une autre	
Base exemple : CopieDonnees_Source_Cible (BD Source).....	80
Sub Creer_Table_Temporaire	
Chapitre : Tables temporaires comme stockage intermédiaire individuel	
Base exemple :	100
Sub De_Formulaire_a_Formulaire_avec_dossiers(oEvent As Object)	
Chapitre : Navigation d'un formulaire à un autre	
Base exemple : Formulaire_a_formulaire.....	70
Sub De_formulaire_a_formulaire(oEvent As Object)	
Chapitre : Navigation d'un formulaire à un autre	
Base exemple : Formulaire_a_formulaire.....	69
Sub DebutDialogue0	
Chapitre : Lancement et fin des dialogues	
Base exemple : Dialogues.....	102
Sub Defilement(oEvent As Object)	
Chapitre : Faire défiler les enregistrements avec une barre de défilement	
Base exemple : Exemple_Defiler_Enregistrements.....	40
Sub DefinirFiltrage	
Chapitre : Filtrage des données via le filtre de formulaire	
Base exemple : Exemple_Defiler_Enregistrements.....	39
Sub Determine_CauseEvenement(oEvent As Object)	
Chapitre : Un événement – plusieurs implémentations	
Base exemple :	77
Sub Diminue_index_Table(stTable As String)	

Chapitre : Diminution de l'index de table pour les champs à valeur automatique	
Base exemple :	89
Sub Ecrire_donnees_vider_cache	
Chapitre : Sécuriser votre base de données	
Base exemple : Média avec Macro.....	86
Sub EcrireContenu(oEvent As Object)	
Chapitre : Mise en évidence des termes recherchés dans les formulaires et les résultats	
Base exemple : Autotexte_Recherche_Orthographe.....	57
Sub Efface_Champs_Donnees	
Chapitre : Boîte de dialogue simple pour saisir de nouveaux enregistrements	
Base exemple : Dialogues.....	102
Sub Efface_Champs_Donnees2	
Chapitre : Boîte de dialogue pour modifier les enregistrements dans une table	
Base exemple : Dialogues.....	109
Sub Enregistre_Donnees1	
Chapitre : Boîte de dialogue simple pour saisir de nouveaux enregistrements	
Base exemple : Dialogues.....	103
Sub Enregistre_Donnees2(oEvent As Object)	
Chapitre : Boîte de dialogue pour modifier les enregistrements dans une table	
Base exemple : Dialogues.....	109
Sub EnregistreRapportPDF(oRapport As Object)	
Chapitre : Lancement, formatage, impression directe et fermeture d'un rapport	
Base exemple :	92
Sub EnregistrerValeurTexteSelection(oEvent As Object)	
Chapitre : Transférer une valeur de clé étrangère d'une zone combinée vers un champ numérique	
Base exemple : BoiteCombinee_ListeChamps.....	62
Sub Ferme_Formulaire_Sans_Sauvegarder(oEvent as Object)	
Chapitre : Limiter les listes de sélection en saisissant les premières lettres	
Base exemple : Recherche_et_Filtre.....	51
Sub Filtre	
Chapitre : Filtrage des enregistrements	
Base exemple : Recherche_et_Filtre.....	36
Sub Filtre_direct(oEvent As Object)	
Chapitre : Limiter les listes de sélection en saisissant les premières lettres	
Base exemple : Recherche_et_Filtre.....	48
Sub Filtre_Info_Additionnelle(oEvent As Object)	
Chapitre : Filtrage des enregistrements	
Base exemple : Recherche_et_Filtre.....	38
Sub FiltreZoneListe	
Chapitre : Zone de liste (ListBox)	
Base exemple :	28
Sub FinDialogue0	
Chapitre : Lancement et fin des dialogues	
Base exemple : Dialogues.....	102
Sub Formulaire_Direct	
Chapitre : Lancement des formulaires directement depuis l'ouverture de la base de données	
Base exemple :	100

Sub GenereActionEnregistrement(oEvent As Object)	
Chapitre : Générer des actions de base de données	
Base exemple : BoiteCombinee_ListeChamps.....	69
Sub HauteurLigneRapportAuto(oRapport As Object)	
Chapitre : Lancement, formatage, impression directe et fermeture d'un rapport	
Base exemple :	91
Sub Horodatage	
Chapitre : Saisie des temps en millisecondes	
Base exemple :	76
Sub ID_et_Date_Courante	
Chapitre : Clé primaire à partir du numéro courant et de l'année	
Base exemple : Numero_Serie_Anee.....	78
Sub ImprimeFermeRapport(oRapport As Object)	
Chapitre : Lancement, formatage, impression directe et fermeture d'un rapport	
Base exemple :	91
Sub ImprimePublipostage	
Chapitre : Faire un publipostage depuis Base	
Base exemple :	93
Sub LanceRapport	
Chapitre : Impression d'un rapport à partir d'un formulaire interne	
Base exemple :	90
Sub LanceRapportExterne(oEvent As Object)	
Chapitre : Impression de rapports à partir d'un formulaire externe	
Base exemple :	92
Sub Lanceur (oEvent As Object)	
Chapitre : Limiter les listes de sélection en saisissant les premières lettres	
Base exemple : Recherche_et_Filtre.....	49
Sub Ligne_Precedente	
Chapitre : Boîte de dialogue pour modifier les enregistrements dans une table	
Base exemple : Dialogues.....	107
Sub Ligne_Suivante	
Chapitre : Boîte de dialogue pour modifier les enregistrements dans une table	
Base exemple : Dialogues.....	108
Sub LigneMax(oEvent As Object)	
Chapitre : Faire défiler les enregistrements avec une barre de défilement	
Base exemple : Exemple_Defiler_Enregistrements.....	39
Sub Limiter (oEvent As Object)	
Chapitre : Limiter les listes de sélection en saisissant les premières lettres	
Base exemple : Recherche_et_Filtre.....	49
Sub LireContenu(oEvent As Object)	
Chapitre : Mise en évidence des termes recherchés dans les formulaires et les résultats	
Base exemple : Autotexte_Recherche_Orthographe.....	56
Sub ListeChampAffiche	
Chapitre : Renvoyer la valeur des zones de liste	
Base exemple :	47
Sub MarqueMauvaisMotDirect(oEvent As Object)	
Chapitre : Vérification orthographique lors de la saisie des données	
Base exemple : Autotexte_Recherche_Orthographe.....	58

Sub MiseAJour	
Chapitre : Mise à jour automatique des formulaires	
Base exemple :	35
Sub montreAvertissement(oChamp As Object, iType As Integer)	
Chapitre : Tester et modifier les contrôles	
Base exemple :	22
Sub Navigation(oEvent As Object)	
Chapitre : Faire défiler les enregistrements avec une barre de défilement	
Base exemple : Exemple_Defiler_Enregistrements.....	40
Sub Ouvre_Element(oEvent As Object)	
Chapitre : Ouvrir des rapports, tableaux, requêtes depuis un formulaire	
Base Exemple : Exemple_Formulaire_a_formulaire.odb.....	70
Sub Ouvre_Form_Rapp_Tabl_Vue_Req(stNom AS STRING, inType AS INTEGER)	
Chapitre : Ouvrir des rapports, tableaux, requêtes depuis un formulaire	
Base Exemple : Exemple_Formulaire_a_formulaire.....	71
Sub Ouvre_Formulaire_Principal	
Chapitre : Navigation d'un formulaire à un autre	
Base exemple : Formulaire_a_formulaire.....	70
Sub OuvreBDD(Optional oEvent As Object)	
Chapitre : Écrire des macros avec Access2Base	
Base exemple :	123
Sub PleinEcran_off	
Chapitre : Formulaires sans barre d'outils dans la fenêtre	
Base exemple : Formulaire_a_formulaire.....	99
Sub PleinEcran_on	
Chapitre : Formulaires sans barre d'outils dans la fenêtre	
Base exemple : Formulaire_a_formulaire.....	99
Sub PleinEcran(boSwitch As Boolean)	
Chapitre : Formulaires en mode plein écran	
Base exemple : Formulaire_a_formulaire.....	99
Sub Pointeur_Souris(oEvent As Object)	
Chapitre : Changer le pointeur de la souris lors du parcours d'un contrôle	
Base exemple : Activer_Fichier_Courriel.....	97
Sub Purge_Table(oEvent As Object)	
Chapitre : Utilisation d'une boîte de dialogue pour nettoyer les mauvaises données dans les tables	
Base exemple :	114
Sub Rechercher(stTable As String)	
Chapitre : Recherche d'enregistrements de données	
Base exemple : Cherche_et_Filtre.....	52
Sub Remplir_ChampsTexte	
Chapitre : Impression via des champs de texte	
Base exemple : BaseDeDonnees_LettreType_Directe.....	94
Sub SauveBase(inMax As Integer)	
Chapitre : Sécuriser votre base de données	
Base exemple : Média avec Macro.....	85
Sub SauveMaintenant	
Chapitre : Sécuriser votre base de données	
Base exemple : Média avec Macro.....	86

Sub ScanTables()	
Chapitre : Imprimer une liste de noms de tables et de champs	
Base exemple :	124
Sub SetDefaultNewRec(poEvent As Object)	
Chapitre : Définir les valeurs par défaut dans les entrées de formulaire	
Base exemple :	126
Sub Supprime_Ligne(oEvent As Object)	
Chapitre : Boîte de dialogue pour modifier les enregistrements dans une table	
Base exemple : Dialogues.....	111
Sub SupprimerFiltre	
Chapitre : Filtrage des données via le filtre de formulaire	
Base exemple : Exemple_Defiler_Enregistrements.....	39