LibreOffice

Guide Base

Chapitre 8
Trucs et astuces

Droits d’auteur

Contributeurs

De cette édition

Pulkit Krishna

Des éditions précédentes

Pulkit Krishna
Hazel Russman
Jochen Schiffers
Robert Großkopf
Dan Lewis
Jost Lange
Jean Hollis Weber

Traduction

Jean-Michel COSTE

Relecteurs

Patrick Auclair

Retour 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 Guide Base.

Table des matières

Informations générales sur les tâches de base de données

Ce chapitre décrit quelques solutions aux problèmes rencontrés par de nombreux utilisateurs de Base.

Filtrage des données

Le filtrage des données à l'aide de l'interface graphique est décrit au chapitre 3, Tables. Nous décrivons ici une solution à un problème soulevé par de nombreux utilisateurs : comment utiliser les zones de liste pour rechercher le contenu des champs dans les tables, qui apparaissent ensuite filtrés dans la section de formulaire sous-jacente et peuvent être modifiés.

La base de ce filtrage est une requête modifiable (voir Chapitre 5, Requêtes) et une table supplémentaire, dans laquelle les données à filtrer sont stockées. La requête affiche à partir de sa table sous-jacente uniquement les enregistrements qui correspondent aux valeurs de filtre. Si aucune valeur de filtre n'est donnée, la requête affiche tous les enregistrements.

L'exemple suivant est basé sur une table « **Media** » qui inclut, entre autres, les champs suivants : **« ID »** (clé primaire), **« Titre »**, **« Catégorie »**. Les types de champ sont respectivement INTEGER, VARCHAR et VARCHAR.

Nous avons d'abord besoin d'une table **« Filtre »**. Elle contient une clé primaire et deux champs de filtre (bien sûr, vous pouvez en avoir plus si vous le souhaitez) : **« ID »** (clé primaire), **« Filtre_1 »**, **« Filtre_2 »**. Comme les champs de la table **« Media »**, à filtrer, sont de type **VARCHAR**, les champs **« Filtre_1 »** et **« Filtre_2 »** sont également de ce type. **« ID »** peut être un champ **oui/non**, car la table **« Filtre »** ne contiendra jamais plus d'un enregistrement.

Attention

Si la base de données utilisée n'est pas de type mono-utilisateur, une telle table de filtrage générerait également le filtrage pour les autres utilisateurs. Si nécessaire, vous pouvez travailler avec des tables temporaires qui ne sont visibles que par l'utilisateur qui les a créées. Ou vous pouvez simplement utiliser le nom d'utilisateur directement comme clé primaire de la table de filtrage. L'ID du champ ne serait alors pas un champ oui/non, mais un champ VARCHAR.

Vous pouvez également filtrer les champs qui apparaissent dans la table **« Media »** uniquement en tant que clés étrangères. Dans ce cas, vous devez donner aux champs correspondants de la table **« Filtre »** le type approprié pour les clés étrangères, généralement INTEGER.

La requête suivante doit être améliorée :

```
SELECT * FROM "Media"
```

Tous les enregistrements de la table **Media** sont affichés, incluant la clé primaire.

```
SELECT * FROM "Media"
WHERE "Titre" = IFNULL((SELECT "Filtre_1" FROM "Filtre"), "Titre")
```

Si le champ **Filtre_1** n'est pas **NULL**, les enregistrements dont le **Titre** est le même que **Filtre_1** sont affichés. Si le champ **Filtre_1** est **NULL**, la valeur du champ **Titre** est utilisée à la place. Il ressort évidemment que tous les enregistrements seront affichés, puisque **Titre** (de IFNULL) est identique au **Titre** (de WHERE **"Titre"** =). Sauf, si le champ **Titre** d'un enregistrement est vide (contient **NULL**), alors cet enregistrement ne sera pas affiché. Par conséquent, nous devons améliorer la requête :

```
SELECT *, IFNULL("Titre", '') AS "T" FROM "Media"
WHERE "T" = IFNULL((SELECT "Filtre_1" FROM "Filtre"), "T")
```

Conseil

IFNULL (expression, valeur) nécessite que l'expression ait le même type de champ que la valeur.

Si l'expression a le type de champ VARCHAR, utilisez deux guillemets simples "" comme valeur.

S'il a DATE comme type de champ, entrez une date comme valeur qui n'est pas contenue dans le champ de la table à filtrer. Utilisez ce format : {D 'AAAA-MM-JJ'}.

S'il s'agit de l'un des types de champ numérique, utilisez le type de champ NUMÉRIQUE pour la valeur. Saisissez un nombre qui n'apparaît pas dans le champ du tableau à filtrer.

Cette variante mènera au but recherché. Au lieu d'utiliser le champ **"Titre"** directement dans la clause WHERE on introduit dans le **SELECT** un alias de ce champ (nommé **"T"**), qui sera vide (") si **"Titre"** est NULL. Dans ces conditions, seul le champ **T** est considéré. Tous les enregistrements sont donc affichés même si **Titre** est **NULL.**

Malheureusement, vous ne pouvez pas faire cela en utilisant l'interface graphique. La commande ne peut être exécutée que directement avec SQL. Pour le rendre modifiable dans l'interface graphique, d'autres ajustements sont nécessaires :

```
SELECT "Media".*, IFNULL("Media"."Titre", '') AS "T"
FROM "Media"
WHERE "T" = IFNULL((SELECT "Filtre_1" FROM "Filtre"), "T")
```

Si la relation de la table avec les champs est maintenant établie, la requête devient modifiable dans l'interface graphique. À titre de test, vous pouvez mettre un titre dans **"Filtre"."Filtre_1"**.

Comme **"Filtre"."ID"** définit la valeur "0", l'enregistrement est sauvegardé et le filtrage peut être compris. Si **"Filtre"."Filtre_1"** est vidé, l'interface graphique traite cela comme **NULL**. Un nouveau test donne un affichage de tous les médias. Dans tous les cas, avant qu'un formulaire ne soit créé et testé, un seul enregistrement avec une clé primaire doit être entré dans la table Filtre. Il ne doit s'agir que d'un seul enregistrement, car les sous-requêtes illustrées ci-dessus ne peuvent transmettre qu'une seule valeur.

La requête peut maintenant être dévelloppée pour filtrer également un second champ :

```
SELECT "Media".*, IFNULL("Media"."Titre", '') AS "T",
            IFNULL("Media"."Categorie", '') AS "K"
FROM "Media"
WHERE "T" = IFNULL((SELECT "Filtre_1" FROM "Filtre"), "T")
   AND "K" = IFNULL((SELECT "Filtre_2" FROM "Filtre"), "K")
```

Ceci conclut la création de la requête modifiable.

Maintenant, pour la requête de base pour les deux zones de liste :

```
SELECT DISTINCT "Titre", "Titre"
FROM "Media" ORDER BY "Titre" ASC
```

La zone de liste doit afficher le **Titre**, puis transmettre également ce **Titre** au champ **Filtre_1** de la table **Filtre** qui sous-tend le formulaire. De plus, aucune valeur en double ne doit être affichée (condition **DISTINCT**). Et le tout doit bien sûr être trié dans le bon ordre.

Une requête correspondante est ensuite créée pour le champ **Categorie**, qui consiste à écrire ses données dans le champ **Filtre_2** de la table **Filtre**.

Si l'un de ces champs contient une clé étrangère, la requête est adaptée afin que la clé étrangère soit transmise à la table **Filtre** sous-jacente.

Le formulaire se compose de deux parties. Le **Formulaire 1** est le formulaire basé sur la table **Filtre**. Le **Formulaire 2** est le formulaire basé sur la requête. Le **Formulaire 1** n'a pas de barre de navigation **Propriétés> Données> Barre de navigation** (Non) et le cycle est défini sur **Enregistrement actif**. En outre, la propriété **Autoriser les ajouts** est définie sur Non. Le premier et le seul enregistrement de ce formulaire existe déjà.

Le formulaire 1 contient deux zones de liste avec les étiquettes appropriées. La **zone de liste 1** renvoie les valeurs de **Filtre_1** et est liée à la requête du champ **Titre**.

ZoneListe2 renvoie des valeurs pour **Filtre_2** et se rapporte à la requête pour le champ **Categorie**.

Le formulaire 2 contient un champ de contrôle de table, dans lequel tous les champs de la requête peuvent être répertoriés à l'exception des champs T et K. Le formulaire fonctionnerait toujours si ces champs étaient présents ; ils sont omis pour éviter une duplication déroutante du contenu des champs. De plus, le formulaire 2 contient un bouton, lié à la fonction de mise à jour du formulaire. Une barre de navigation supplémentaire peut être intégrée pour empêcher le scintillement de l'écran à chaque fois que le formulaire change, car la barre de navigation est présente dans un formulaire et pas dans l'autre.

Une fois le formulaire terminé, la phase de test commence. Lorsqu'une zone de liste est modifiée, le bouton du formulaire 2 est utilisé pour stocker cette valeur et mettre à jour le formulaire 2. Cela concerne désormais la valeur fournie par la zone de liste. Le filtrage peut être rétrospectif en choisissant un champ vide dans la zone de liste.

Ces modèles de requêtes, ici voulus simples, sont repris sous des formes plus élaborées dans la base d'exemple « **Exemple_Cherche_et_Filtre.odb** » dans la requête « **RequeteFiltre** ».

Recherche de données

La principale différence entre la recherche de données et le filtrage des données réside dans la technique de requête. L'objectif est de fournir, en réponse à des termes de recherche en langage courant, une liste résultante d'enregistrements qui peuvent ne contenir que partiellement ces termes réels.

Rechercher avec LIKE

La table dans laquelle la recherche est effectuée peut être la même que celle qui contient déjà les valeurs de filtre. La table **Filtre** est simplement complété par un champ nommé **TexteCherche** (**Varchar**). Cela signifie que la même table peut être consultée, filtrée et recherchée dans les formulaires en même temps.

Le formulaire est construit comme pour le filtrage. Au lieu d'une zone de liste, nous avons besoin d'un contrôle de saisie de texte, pour le terme de recherche, et peut être aussi d'un contrôle étiquette avec le libellé Recherche. Le contrôle « Zone de texte » du terme de recherche peut être autonome dans le formulaire ou avec les champs de filtrage, si les deux fonctions sont souhaitées.

Alors que le filtrage utilise un terme qui apparaît déjà dans la table sous-jacente, la recherche utilise des entrées arbitraires.

```
SELECT * FROM "Media"
WHERE "Titre" = (SELECT "TexteCherche" FROM "Filtre")
```

Cette requête conduit normalement à une liste vide de résultat pour les raisons suivantes :

- Il est rare que quelqu'un connaisse le titre complet par cœur lors de la saisie du terme de recherche. Cela n'afficherait pas le titre. Pour trouver le livre « Le guide de l'auto-stoppeur de la galaxie », il devrait suffire de saisir « Guide de l'auto-stoppeur », « auto-stop », et même simplement « auto » dans le champ de recherche.

- Si le champ TexteCherche est vide, aucun enregistrement de données ne sera affiché. La requête au dessus reverrait **NULL**, qui ne peut apparaître que dans une condition utilisant **IS NULL**.
- Même si cela était ignoré, la requête entraînerait l'affichage de tous les enregistrements qui n'ont pas d'entrée dans le champ **Titre**.

La dernière condition peut être supprimée si la condition de filtrage est :

```
SELECT * FROM "Media"
WHERE "Titre" = IFNULL((SELECT "TexteCherche" FROM "Filtre"), "Titre")
```

Avec ce raffinement du filtrage (que se passe-t-il si le titre est NULL?). On obtient un résultat plus conforme aux attentes. Mais la première condition n'est toujours pas remplie. La recherche devrait bien fonctionner lorsque seules des connaissances fragmentaires sont disponibles. La technique de requête doit donc utiliser la condition **LIKE** :

```
SELECT * FROM "Media"
WHERE "Titre" LIKE (SELECT '%' || "TexteCherche" ||'%' FROM "Filtre")
```

ou mieux encore :

```
SELECT * FROM "Media"
   WHERE "Titre" LIKE IFNULL((SELECT '%' || "TexteCherche" ||'%'
FROM "Filtre"), "Titre")
```

LIKE, associé à %, signifie que tous les enregistrements contenant n'importe où le terme de recherche sont affichés. % est un caractère générique pour n'importe quel nombre de caractères avant ou après le terme de recherche. Diverses questions subsistent après la création de cette version de la requête :

- Il est courant d'utiliser des lettres minuscules pour les termes de recherche. Alors, quel résultat j'obtiens si je tape "auto" au lieu de "Auto"?
- Quelles autres conventions écrites doivent être prises en compte ?
- Qu'en est-il des champs qui ne sont pas formatés en tant que champs de texte ? Pouvez-vous rechercher des dates ou des nombres avec le même champ de recherche ?
- Et si, comme dans le cas du filtre, vous voulez empêcher les valeurs NULL dans le champ de provoquer l'affichage de tous les enregistrements ?

La variante suivante couvre une ou deux de ces possibilités :

```
SELECT * FROM "Media"
   WHERE LOWER("Titre")
   LIKE IFNULL((SELECT '%' ||LOWER("TexteCherche") ||'%' FROM "Filtre"),
   LOWER("Titre"))
```

La condition modifie le terme de recherche et le contenu du champ en minuscules. Cela permet également de comparer des phrases entières.

```
SELECT * FROM "Media"
   WHERE LOWER("Titre")
   LIKE IFNULL((SELECT '%' || LOWER("TexteCherche") ||'%' FROM "Filtre"),
      LOWER("Title")) OR LOWER("Categorie")
   LIKE (SELECT '%' || LOWER("TexteCherche") ||'%' FROM "Filtre")
```

La fonction IFNULL doit se produire une seule fois, de sorte que lorsque le TexteCherche est NULL, LOWER ("Titre") LIKE LOWER ("Titre") est interrogé. Et comme le titre doit être un champ qui ne peut pas être NULL, dans ce cas, tous les enregistrements sont affichés. Bien sûr, pour plusieurs recherches de champ, ce code devient d'autant plus long. Dans de tels cas, il est préférable d'utiliser une macro, pour permettre au code de couvrir tous les champs en une seule fois.

Mais le code fonctionne-t-il toujours avec des champs qui ne sont pas du texte ? Bien que la condition LIKE soit vraiment adaptée au texte, elle fonctionne également pour les nombres, les dates et les heures sans nécessiter de modifications. Donc, en fait, la conversion de texte n'a pas

besoin d'avoir lieu. Cependant, un champ de temps qui est un mélange de texte et de nombres ne peut pas interagir avec les résultats de la recherche, sauf si la requête est élargie, de sorte qu'un seul terme de recherche soit subdivisé sur tous les espaces entre le texte et les nombres. Ceci, cependant, gonflera considérablement la requête.

Conseil

Les requêtes utilisées pour filtrer et rechercher des enregistrements peuvent être directement incorporées dans le formulaire.

Toutes les conditions ci-dessus peuvent être saisies dans les propriétés du formulaire à la ligne Filtre.

```
SELECT * FROM "Media" WHERE "Titre" = IFNULL((SELECT "TexteCherche" FROM
"Filtre"), "Titre")
```

devient alors un formulaire qui utilise le contenu de la table Media.

Sous "Filtre", nous avons

```
("Media"."Titre" =
  IFNULL((SELECT "TexteCherche" FROM "Filtre"), "Media"."Titre"))
```

Dans l'entrée de filtre, veillez à ce que la condition soit mise entre parenthèses et fonctionne avec le terme "Table". "Champ".

L'avantage de cette variante est que le filtre peut être activé et désactivé lorsque le formulaire est ouvert.

Rechercher avec LOCATE

La recherche avec LIKE est généralement satisfaisante pour les bases de données avec des champs contenant une quantité de texte qui peut être facilement visualisée. Mais qu'en est-il des champs Mémo, qui peuvent contenir plusieurs pages de texte ? Dans ce cas, la recherche doit déterminer où se trouve le texte spécifié.

Pour localiser exactement le texte, **HSQLDB** a la fonction **LOCATE**. Elle prend un terme de recherche (le texte que vous souhaitez rechercher) comme argument. Vous pouvez également ajouter une position à rechercher. En bref : **LOCATE** (**terme de recherche**, **champ de texte de la base de données**, **position**).

Dans **FIREBIRD**, il faut utiliser **POSITION** à la place de LOCATE, pour cela consulter ce Lien.

La fonction **`SUBSTRING`** utilisée dans ce qui `suit` doit également être écrite dans une syntaxe différente pour Firebird. Au lieu de **`SUBSTRING (texte, position de départ [, longueur])`**, utilisez **`SUBSTRING (texte FROM position de départ [FOR longueur])`**.

L'explication suivante utilise une table appelée Table. La clé primaire s'appelle ID et doit être unique. Il existe également un champ appelé **Memo** qui a été créé comme un champ de type Memo (LONGVARCHAR). Ce champ Mémo contient quelques phrases de ce manuel.

Les captures d'écran de ce chapitre sont tirées de la base de données « **Exemple_Autotexte_Recherche_Orthographe.odb** », qui est jointe à ce manuel.

Les exemples de requêtes sont présentés sous forme de requêtes paramétrées. Le texte de recherche à saisir est toujours "office".

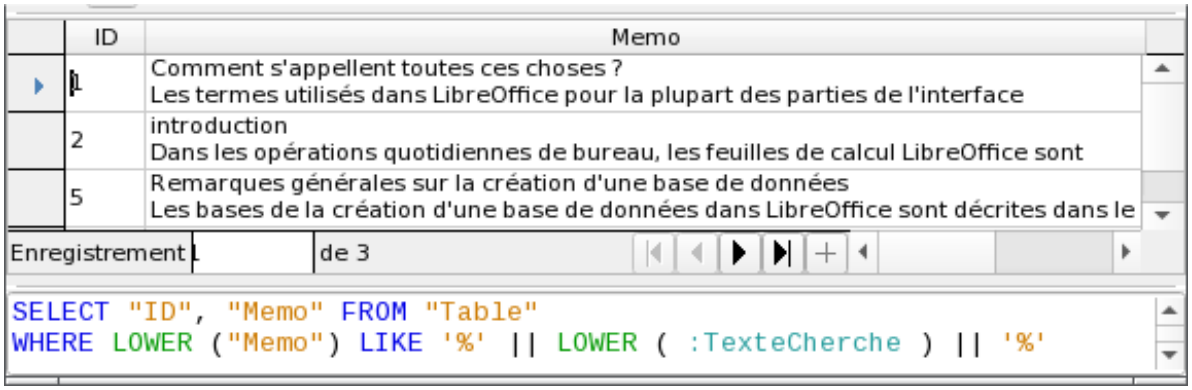


Figure 1: Recherche avec LIKE

Nous utilisons d'abord **LIKE**, qui ne peut être utilisé que dans certaines conditions. Si le texte recherché se trouve n'importe où, l'enregistrement correspondant s'affiche. La comparaison est entre une version minuscule du contenu du champ, utilisant **LOWER ("Memo")** et une version minuscule du texte de recherche utilisant **LOWER (:TexteCherche)**, pour rendre la recherche insensible à la casse. Plus le texte du champ Mémo est long, plus il devient difficile de voir le terme dans le texte récupéré. (**Cf. Requête_Like dans la base exemple**).

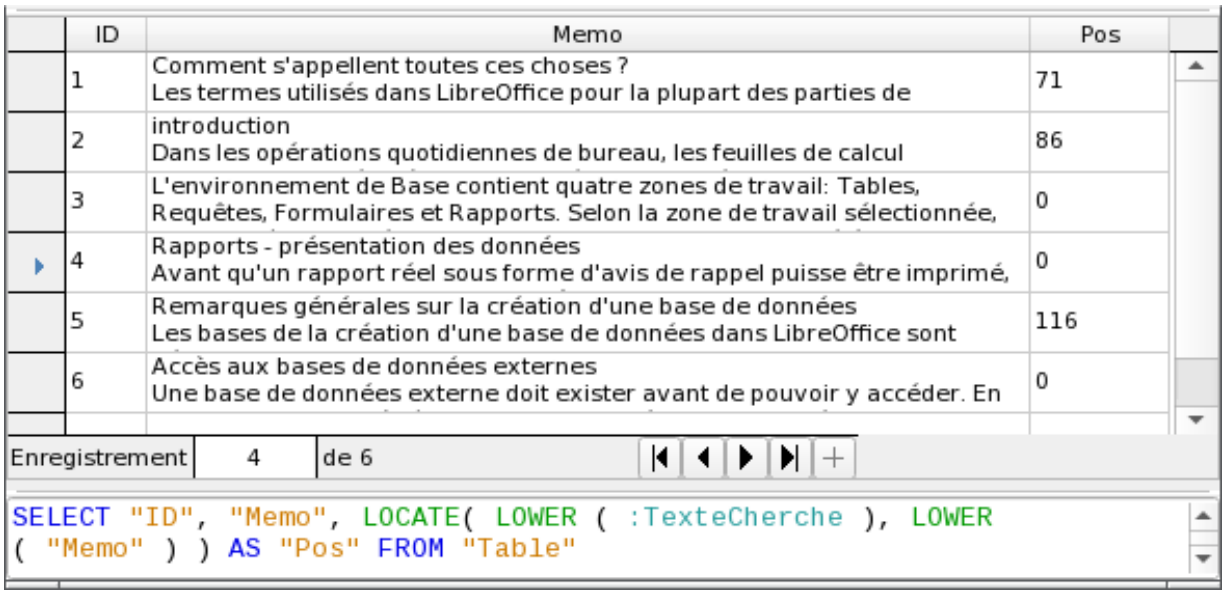


Figure 2: Recherche avec LOCATE

LOCATE vous montre plus précisément où se trouve votre terme de recherche. Dans les enregistrements 3, 4, 6, le terme n'apparaît pas. Dans ce cas, **LOCATE** donne la position "0". Il est assez facile de confirmer le nombre donné pour l'enregistrement 1, la chaîne "Office" est présente à la seconde ligne. Naturellement, il serait également possible d'afficher les résultats de LOCATE de la même manière que pour LIKE. (**Cf. Requête_Locate dans la base exemple**).

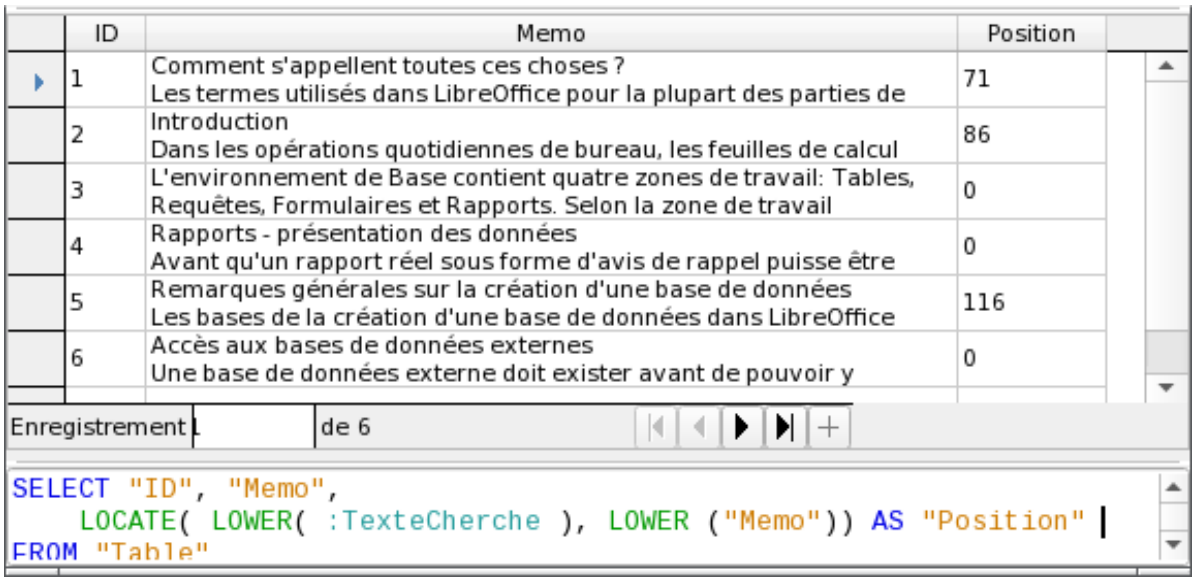


Figure 3: Recherche avec LOCATE

Dans la colonne Cible (Figure 4, les résultats de la recherche sont affichés avec plus de précision. La requête précédente a été utilisée comme base pour celle-ci. Cela permet d'utiliser le mot "Position" dans la requête externe au lieu d'avoir à répéter **LOCATE (LOWER (:TexteCherche), LOWER ("Memo"))** à chaque fois. En principe, ce n'est pas différent d'enregistrer la requête précédente et de l'utiliser comme source pour celle-ci.

`"Position" = 0` signifie qu'il n'y a pas de résultat. Dans ce cas, « **`**non trouvé**`** » est affiché.

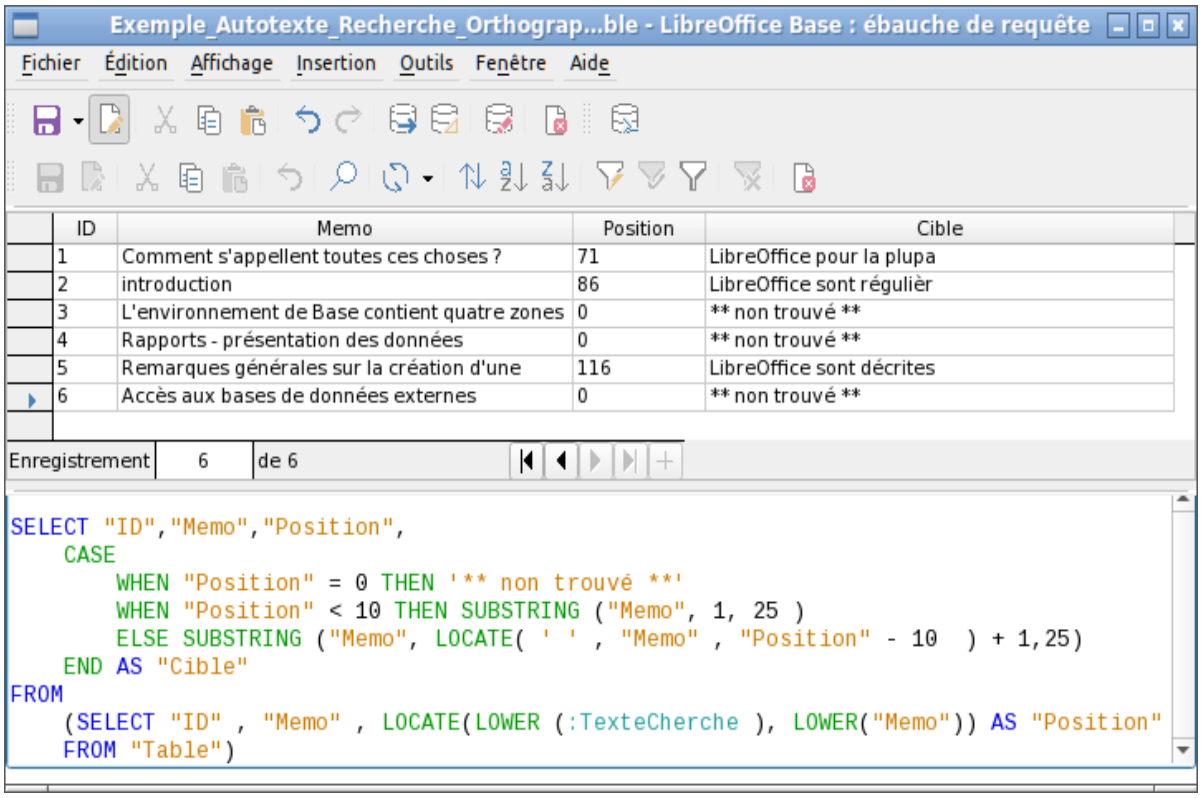


Figure 4: Cible extraite de la recherche

`"Position" < 10` signifie que le terme de recherche apparaît juste au début du texte. 10 caractères peuvent être facilement scannés à l'œil nu. Par conséquent, le texte entier est affiché. Ici au lieu de **SUBSTRING ("Memo", 1)**, nous aurions pu utiliser simplement **"Memo"**.

Pour tous les autres résultats, un espace est recherché **' '** jusqu'à 10 caractères avant le terme de recherche. Le texte affiché ne commence pas au milieu d'un mot mais après un espace. **`SUBSTRING("Memo", LOCATE(' ',"Memo","Position"-10)+1)`** garantit que le texte commence au début d'un mot qui se trouve au plus 10 caractères avant le terme de recherche.

En pratique, nous voudrions utiliser plus de caractères, car il y a beaucoup de mots plus longs que cela, et le terme de recherche peut se trouver dans un autre mot avec plus de 10 caractères devant lui. LibreOffice contient le terme de recherche "office" avec le "O" comme sixième caractère. Dans d'autres cas, le résultat pourrait être faux

La technique de requête est la même que pour la requête précédente. Seule la longueur de la cible à afficher a été réduite à 25 caractères. La fonction **SUBSTRING** requiert comme arguments le **texte à rechercher**, puis la **position de départ** du résultat, et comme troisième argument facultatif, la **longueur** de la chaîne de texte à afficher. Ici, il a été défini assez court à des fins de démonstration. Un avantage de le raccourcir est que les exigences de stockage pour un grand nombre d'enregistrements sont réduites et que l'emplacement est facilement visible. Un inconvénient visible de ce type de raccourcissement de chaîne est que la coupe est faite en stricte conformité avec la limite de 25 caractères, sans tenir compte de l'endroit où les mots commencent.

Ici, nous recherchons du 25e caractère dans "Memo" au caractère espace suivant. Le contenu à afficher se situe entre ces deux positions.

C'est beaucoup plus simple si la correspondance apparaît au début du champ. Ici **`LOCATE(' ',"Memo",25)`** donne la position exacte où commence le texte. Puisque nous voulons que le texte soit affiché depuis le début, cela correspond exactement à la longueur du terme affichable.

	ID	Memo	Posit...	Cible
▶	1	Comment s'appellent toutes ces choses ?	71	LibreOffice pour la plupart
	2	Introduction	86	LibreOffice sont régulièrement
	3	L'environnement de Base contient quatre zones de travail: Tables,	0	** Non trouvé **
	4	Rapports - présentation des données	0	** Non trouvé **
	5	Remarques générales sur la création d'une base de données	116	LibreOffice sont décrites
	6	Accès aux bases de données externes	0	** Non trouvé **

Enregistrement 1 de 6

```
SELECT "ID", "Memo", "Position",
    CASE
        WHEN "Position" = 0 THEN '** Non trouvé **'
        WHEN "Position" < 10 THEN SUBSTRING ("Memo",1, LOCATE (' ' ,"Memo", 25))
        ELSE SUBSTRING ("Memo", LOCATE (' ', "Memo", "Position" - 10) + 1,
            ( LOCATE (' ', "Memo", "Position" + 20 ) -
                ( LOCATE (' ', "Memo", "Position" - 10) + 1 )))
    END AS "Cible"
FROM
    (SELECT "ID", "Memo", LOCATE( LOWER( :TexteCherche ), LOWER( "Memo" ) ) AS "Position"
FROM "Table")
```

Figure 5: Cible sans coupure de mot.

La recherche de l'espace suivant le terme recherché n'est pas plus compliquée si le terme se trouve plus loin dans le champ. La recherche commence simplement là où se trouve la correspondance. Ensuite, 20 caractères supplémentaires sont comptés, qui doivent suivre en toutes circonstances. L'espace suivant est localisé à l'aide de **`LOCATE(' ',"Memo","Position"+20)`**. Cela donne uniquement l'emplacement dans le champ dans son ensemble, pas la longueur de la chaîne à afficher. Pour cela, nous devons soustraire la position à laquelle l'affichage du texte correspondant doit commencer. Cela a déjà été défini dans la requête par **`LOCATE(' ',"Memo","Position"-10)+1`**. De cette manière, la longueur correcte du texte peut être trouvée.

La même technique peut être utilisée pour enchaîner les requêtes. La requête précédente devient désormais la source de données de la nouvelle. Elle a été insérée, entre parenthèses, sous le terme FROM. Seuls les champs sont renommés dans une certaine mesure, car il y a maintenant plusieurs positions et correspondances. De plus, la position suivante reçoit une référence en utilisant **`LOCATE(LOWER(: TexteCherche), LOWER("Memo"),"Position01"+1)`**. Cela signifie que la recherche recommence à la position après le texte correspondant précédent.

La requête la plus externe définit les champs correspondants pour les deux autres requêtes et fournit également "Cible02" en utilisant la même méthode que celle utilisée précédemment pour "Cible01". En outre, cette requête la plus externe détermine s'il existe d'autres correspondances.

ID	Memo	Position01	Cible01	Position02	Cible02	Position03
1	Comment s'appellent	71	LibreOffice pour la	0	** Non trouvé **	0
2	Introduction	86	LibreOffice sont	0	** Non trouvé **	0
3	L'environnement de	0	** Non trouvé **	0	** Non trouvé **	0
4	Rapports - présentation	0	** Non trouvé **	0	** Non trouvé **	0
5	Remarques générales	116	LibreOffice sont	243	de LibreOffice,	353
6	Accès aux bases de	0	** Non trouvé **	0	** Non trouvé **	0

Enregistrement 5 de 6

```
SELECT "ID", "Memo", "Position01", "Cible01", "Position02",
    CASE
        WHEN "Position02" = 0 THEN '** Non trouvé **'
        WHEN "Position02" < 10 THEN SUBSTRING ("Memo",1, LOCATE (' ' ,"Memo", 25))
        ELSE SUBSTRING ("Memo", LOCATE (' ', "Memo", "Position02" - 10) + 1,
            ( LOCATE (' ', "Memo", "Position02" + 20 ) -
                ( LOCATE (' ', "Memo", "Position02" - 10) + 1 ))
            )
    END AS "Cible02",
    CASE
        WHEN "Position02" = 0 THEN 0
        ELSE LOCATE( LOWER( :TexteCherche ), LOWER( "Memo" ) , "Position02" + 1)
    END AS "Position03"
FROM
    (SELECT "ID", "Memo", "Position01",
    CASE
        WHEN "Position01" = 0 THEN '** Non trouvé **'
        WHEN "Position01" < 10 THEN SUBSTRING ("Memo",1, LOCATE (' ' ,"Memo", 25))
        ELSE SUBSTRING ("Memo", LOCATE (' ', "Memo", "Position01" - 10) + 1,
            ( LOCATE (' ', "Memo", "Position01" + 20 ) -
                ( LOCATE (' ', "Memo", "Position01" - 10) + 1 ))
            )
    END AS "Cible01",
    CASE
        WHEN "Position01" = 0 THEN 0
        ELSE LOCATE( LOWER( :TexteCherche ), LOWER( "Memo" ) ,  "Position01" + 1)
    END AS "Position02"
    FROM
    (SELECT "ID", "Memo", LOCATE( LOWER( :TexteCherche ), LOWER("Memo")) AS "Position01"
    FROM "Table")
    )
```

Figure 6: Recherche de cibles multiples

La position correspondante est donnée par "**Position03**". Seul l'enregistrement 5 a d'autres correspondances, et celles-ci peuvent être trouvées dans une autre sous-requête.

L'empilement des requêtes présentées ici pourrait être poursuivi si vous le souhaitez. Cependant, l'ajout de chaque nouvelle requête externe impose une charge supplémentaire sur le système. Il serait nécessaire de faire quelques tests pour déterminer jusqu'où il est utile et réaliste d'aller. Le chapitre 9, Macros, montre comment les macros peuvent être utilisées pour rechercher toutes les chaînes de texte correspondantes dans un champ grâce à l'utilisation d'un formulaire.

Note

Toutes ces requêtes sont visibles dans la base Exemple_Autotexte_Recherche_Orthographe.odb associée à ce manuel, mais également dans l' Annexe à la fin de ce document.

Gestion des images et des documents dans Base

Les formulaires Base utilisent des contrôles graphiques pour gérer les images. Si vous utilisez une base de données HSQLDB interne, les contrôles graphiques sont le seul moyen de lire des images hors de la base de données sans utiliser de macros. Ils peuvent également être utilisés comme liens vers des images en dehors du fichier de base de données.

Lecture d'images dans la base de données

La base de données nécessite une table qui remplit au moins les conditions suivantes :

Nom de champ	***Type de champ***	***Description***
ID	Integer	ID est la clé primaire de cette table.
Image	Contrôle picto	Contient l'image sous forme de données binaires.

Une clé primaire doit être présente, mais il n'est pas nécessaire qu'elle soit un entier. D'autres champs qui ajoutent des informations sur l'image doivent être ajoutés.

Les données qui seront lues dans le champ image ne sont pas visibles dans une table. Au lieu de cela, vous voyez le mot <OBJECT>. De la même manière, les images ne peuvent pas être saisies directement dans une table. Vous devez utiliser un formulaire qui contient un contrôle graphique. Le contrôle graphique s'ouvre lorsque vous cliquez dessus pour afficher une boîte de dialogue de sélection de fichiers. Ensuite, il affiche l'image qui a été lue à partir du fichier sélectionné.

Les images qui doivent être insérées directement dans la base de données doivent être aussi petites que possible. Comme Base ne fournit aucun moyen (sauf en utilisant des macros) d'exporter des images dans leur taille d'origine, il est logique de n'utiliser que la taille nécessaire, par exemple pour l'impression dans un rapport. Les images originales dans la gamme mégapixel sont complètement inutiles et gonflent la base de données. Après avoir ajouté seulement quelques images, le HSQLDB interne donne une erreur : **Java. NullPointerException** et ne peut plus stocker l'enregistrement. Même si les images ne sont pas aussi grandes, il peut arriver que la base de données devienne inutilisable. Une bonne méthode est de réduire le nombre de couleurs (256 est suffisant).

De plus, les images ne doivent pas être intégrées dans des tables conçues pour y faire des recherches. Si, par exemple, vous disposez d'une base de données du personnel et que des images à utiliser dans les passes doivent être incluses, il est préférable de les stocker dans une table séparée avec une clé étrangère dans la table principale. Cela signifie que la table principale peut être scannée beaucoup plus rapidement, car elle ne nécessite pas autant de mémoire.

Liens vers des images et des documents

Avec une structure de dossiers soigneusement conçue, il est plus pratique d'accéder directement aux fichiers externes. Les fichiers en dehors de la base de données peuvent être aussi volumineux que nécessaire, sans avoir aucun effet sur le fonctionnement de la base de données elle-même. Malheureusement, cela signifie également que renommer un dossier sur votre propre ordinateur ou sur Internet peut vous faire perdre l'accès au fichier image, s'il n'est pas dans un sous dossier de la base.

Si vous ne souhaitez pas lire les images directement dans la base de données mais uniquement les lier, vous devez apporter une petite modification au tableau précédent :

Nom de champ	***Type de champ***	***Description***
ID	Integer	ID est la clé primaire de cette table.

Image	Zone Texte	Contient le chemin d'accès à l'image.

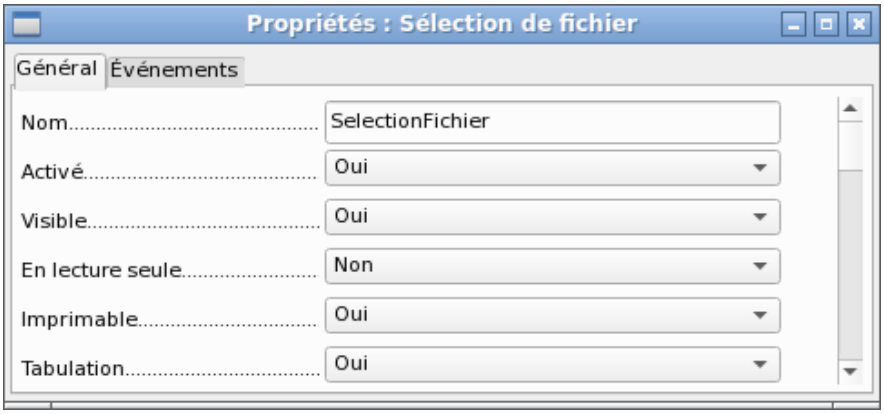
Si le type de champ est défini sur texte, le contrôle graphique du formulaire transmettra le chemin d'accès au fichier. L'image est toujours accessible par le contrôle graphique exactement comme une image interne.

Malheureusement, vous ne pouvez pas faire la même chose avec un document. Il n'est même pas possible de lire le chemin, car les commandes graphiques sont conçues pour les images graphiques et la boîte de dialogue du sélecteur de fichiers n'affiche que les fichiers au format graphique.

Avec une image, le contenu peut au moins être vu dans le contrôle graphique, en utilisant le chemin d'accès au fichier. Avec un document, il ne peut y avoir aucun affichage même si le chemin est stocké dans une table (sauf peut-être pour un document PDF, si le système possède un visualiseur). Nous devons d'abord agrandir un peu le tableau afin qu'au moins une petite quantité d'informations sur le document puisse être rendue visible.

Nom du champ	***Type du champ***	***Description***
ID	Integer	ID est la clé primaire de cette table.
Description	Zone Texte	Description du document, termes de recherche
Chemin	Zone Texte	Contient le chemin d'accès au document.

Pour rendre le chemin d'accès au document visible, nous devons créer un champ de sélection de fichier dans le formulaire.



Un champ de sélection de fichier n'a pas d'onglet pour les données dans sa boîte de dialogue de propriétés. Il n'est donc lié à aucun champ de la table sous-jacente.

Lier des documents avec un chemin absolu

En utilisant le champ de sélection de fichier, le chemin peut être affiché mais pas stocké. Pour cela, une procédure spéciale est nécessaire, liée à **Evénements> Texte modifié** :

```
SUB LireChemin(oEvent AS OBJECT)
    DIM oFormulaire AS OBJECT
    DIM oChamp AS OBJECT
    DIM oChamp2 AS OBJECT
    DIM stUrl AS STRING
    oChamp = oEvent.Source.Model
    oFormulaire = oChamp.Parent
    oChamp2 = oFormulaire.getByName("imgImage")
    IF oChamp.Text <> "" THEN
        stUrl = ConvertToUrl(oChamp.Text)
        oChamp2.BoundField.updateString(stUrl)
    END IF
```

```
END SUB
```

L'événement qui déclenche la procédure lui est transmis et aide à trouver le formulaire et le champ dans lesquels le chemin doit être stocké. Utiliser **`oEvent AS OBJECT`** rend l'accès plus simple lorsqu'un autre utilisateur souhaite utiliser une macro du même nom dans un sous-formulaire. Il rend le champ de sélection de fichier accessible via **`oEvent.Source.Model`**. Le formulaire est accessible en tant que parent du champ de sélection de fichier. Le nom du formulaire est donc sans importance. Depuis le formulaire, le champ appelé "*imgImage*" est désormais accessible. Ce champ est normalement utilisé pour stocker les chemins vers les fichiers image. Dans ce cas, l'URL du fichier sélectionné y est écrite. Pour garantir que l'URL fonctionne avec les conventions du système d'exploitation, le texte du champ de sélection de fichier est converti dans une forme généralement valide à l'aide de **ConvertToUrl**.

La table de base de données contient désormais un chemin au format absolu : `file:///…`

Si les entrées de chemin sont lues à l'aide d'un contrôle graphique, cela donnera un chemin relatif. Pour rendre cela utilisable, il faut l'améliorer. La procédure pour ce faire est beaucoup plus longue, car elle implique une comparaison entre le chemin d'entrée et le chemin réel.

Lier des documents avec un chemin relatif

La macro suivante est liée à la propriété "*Texte modifié*" du champ de sélection de fichier.

```
SUB LireChemin
    DIM oDoc AS OBJECT
    DIM oDrawpage AS OBJECT
    DIM oFormulaire AS OBJECT
    DIM oChamp AS OBJECT
    DIM oChamp2 AS OBJECT
    DIM arDebut_Url()
    DIM ar()
    DIM ar1()
    DIM ar2()
    DIM stTexte AS STRING
    DIM stUrl_complete AS STRING
    DIM stUrl_Texte AS STRING
    DIM stUrl AS STRING
    DIM stUrl_Coupe AS STRING
    DIM ink AS INTEGER
    DIM i AS INTEGER
    oDoc = thisComponent
    oDrawpage = oDoc.Drawpage
    oFormulaire = oEvent.Source.Model.Parent
    oChamp = oFormulaire.getByName("imgImage")
    oChamp2 = oFormulaire.getByName("SelectionFichier")
```

Tout d'abord, comme dans toutes les procédures, les variables sont déclarées. Ensuite, les champs importants pour la saisie des chemins sont recherchés. L'ensemble du code suivant n'est alors exécuté que s'il y a effectivement quelque chose dans le champ de sélection de fichier, c'est-à-dire qu'il n'a pas été vidé par un changement d'enregistrement.

```
    IF oChamp2.Text <> "" THEN
        arDebut_Url = split(oDoc.Parent.Url, oDoc.Parent.Title)
        ar = split(ConvertToUrl(oFeld2.Text),"/")
        stTexte = ""
```

Le chemin d'accès au fichier de base de données est lu. Ceci est effectué, comme indiqué ci-dessus, d'abord en lisant l'URL entière, puis en la divisant en un tableau de sorte que le premier élément du tableau contienne le chemin direct.

Ensuite, tous les éléments du chemin trouvés dans le champ de sélection de fichier sont lus dans le tableau **ar**. Le séparateur est **/**. Cela peut être fait directement sous Linux. Sous Windows, le contenu de **oChamp2** doit être converti en une URL, qui utilisera une barre oblique (pas une barre oblique inverse) comme délimiteur de chemin.

Le but du fractionnement est d'obtenir le chemin d'accès au fichier en coupant simplement le nom du fichier à la fin. Par conséquent, à l'étape suivante, le chemin d'accès au fichier est reconstitué et placé dans la variable **stTexte**. La boucle ne se termine pas avec le dernier élément du tableau **ar** mais avec l'élément précédent.

```
        FOR i = LBound(ar()) TO UBound(ar()) - 1
            stTexte = stTexte & ar(i) & "/"
        NEXT
        stTexte = Left(stTexte, Len(stTexte)-1)
        arDebut_Url(0) = Left(arDebut_Url(0), Len(arDebut_Url(0))-1)
```

Le / final est à nouveau supprimé, sinon une valeur de tableau vide apparaîtrait dans le tableau suivant, ce qui interférerait avec la comparaison de chemin. Pour une comparaison correcte, le texte doit être converti en une URL correcte commençant par `file:///`. Enfin, le chemin d'accès au fichier de la base de données est comparé au chemin qui a été créé.

```
    stUrl_Texte = ConvertToUrl(stTexte)
    ar1 = split(stUrl_Texte,"/")
    ar2 = split(arDebut_Url(0),"/")
    stUrl = ""
    ink = 0
    stUrl_Coupe = ""
```

les tableaux **`ar1`** et **`ar2`** sont comparés étape par étape dans une boucle.

```
        FOR i = LBound(ar2()) TO UBound(ar2())
            IF i <= UBound(ar1()) THEN
```

Le code suivant n'est exécuté que si le nombre **i** n'est pas supérieur au nombre d'éléments dans **ar1**. Si la valeur dans **ar2** est la même que la valeur correspondante dans **ar1**, et qu'aucune valeur incompatible n'a été trouvée jusqu'à ce point, le contenu commun est stocké dans une variable qui peut finalement être coupée de la valeur du chemin.

```
        IF ar2(i) = ar1(i) AND ink = 0 THEN
            stUrl_Coupe = stUrl_Coupe & ar1(i) & "/"
        ELSE
```

S'il y a une différence à tout moment entre les deux tableaux, alors pour chaque valeur différente, le signe pour remonter d'un répertoire sera ajouté à la variable **stUrl**.

```
            stUrl = stUrl & "../"
            ink = 1
        END IF
```

Dès que l'index stocké dans **i** est supérieur au nombre d'éléments dans **ar1**, chaque valeur supplémentaire dans **ar2** entraînera le stockage d'un../ supplémentaire dans la variable **stUrl**.

```
        ELSE
            stUrl = stUrl & "../"
        END IF
        NEXT
        stUrl_complete = ConvertToUrl(oFeld2. Text)
        Champ.boundField. UpdateString(stUrl & Right(stUrl_complete,
Len(stUrl_complete)-Len(stUrl_Coupe)))
    END IF
END SUB
```

Lorsque la boucle via **ar2** est terminée, nous avons établi si, et de combien, le fichier auquel accéder est plus haut dans l'arborescence que le fichier de base de données. Maintenant, **stUrl_complete** peut être créé à partir du texte dans le champ de sélection de fichier. Celui-ci contient également le nom du fichier. Enfin, la valeur est transférée dans le contrôle graphique. La valeur de l'URL commence par stUrl, qui contient le nombre de points nécessaire (../). Ensuite, le début de **stUrl_complete**, la partie qui s'est avérée être la même pour la base de données et le fichier externe, est coupé. La manière de couper la chaîne est stockée dans **stUrl_Coupe**.

Affichage d'images et de documents liés

Les images liées peuvent être affichées directement dans un contrôle graphique. Mais affichage plus grand serait plus approprié pour montrer les détails.

Les documents ne sont normalement pas visibles dans Base, sauf PDF dans de bonnes conditions.

Pour rendre ce type d'affichage possible, nous devons à nouveau utiliser des macros. Cette macro est lancée à l'aide d'un bouton sur le formulaire qui contient le contrôle graphique.

```
SUB Voir(oEvent AS OBJECT)
    DIM oDoc AS OBJECT
    DIM oDrawpage AS OBJECT
    DIM oFormulaire AS OBJECT
    DIM oChamp AS OBJECT
    DIM oShell AS OBJECT
    DIM stUrl AS STRING
    DIM stChamp AS STRING
    DIM arDebut_Url()
    oDoc = thisComponent
    oDrawpage = oDoc.Drawpage
    oFormulaire = oDrawpage.Forms.getByName("Formulaire")
    oChamp = oFormulaire.getByName("imgCheminImage")
    stUrl = oChamp.BoundField.getString
```

Le contrôle graphique dans le formulaire est localisé. Comme la table ne contient pas l'image elle-même mais seulement un chemin d'accès stocké sous forme de chaîne de texte, ce texte est récupéré en utilisant **`getString`**.

Ensuite, le chemin d'accès au fichier de base de données est déterminé. Le fichier odb, le conteneur des formulaires, est accessible à l'aide de **oDoc.Parent**. L'URL entière, y compris le nom de fichier, est lue à l'aide de **oDoc.Parent.Url**. Le nom de fichier est également stocké dans **oDoc.Parent.Title**. Le texte est séparé à l'aide de la fonction de fractionnement avec le nom de fichier comme séparateur. Cela donne le chemin d'accès au fichier de base de données comme premier et unique élément du tableau.

```
    arDebut_Url = split(oDoc.Parent.Url, oDoc.Parent.Title)
    oShell = createUnoService("com.sun.star.system. SystemShellExecute")
    stChamp = convertToUrl(arDebut_Url(0) + stUrl)
    oShell.execute(stChamp,,0)
END SUB
```

Les programmes externes peuvent être lancés à l'aide de la structure **com.sun.star.system. SystemShellExecute**. Le chemin d'accès absolu au fichier, composé du chemin d'accès au fichier de base de données et du chemin relatif stocké en interne à partir du fichier de base de données, est transmis au programme externe. Le système d'exploitation détermine quel programme est appelé pour ouvrir le fichier.

La commande **oShell.execute** prend trois arguments. Le premier est un fichier **exécutable ou** le **chemin d'accès** à un fichier de données lié à un programme par le système. Le second est une **liste d'arguments** pour le programme. Le troisième est un **nombre** qui détermine comment les erreurs doivent être signalées. Les possibilités sont **0** (message d'erreur par défaut), **1** (pas de message) et **2** (autoriser uniquement l'ouverture d'URL absolues).

Lire des documents dans la base de données

Lors de la lecture des documents, les conditions suivantes doivent toujours être respectées :

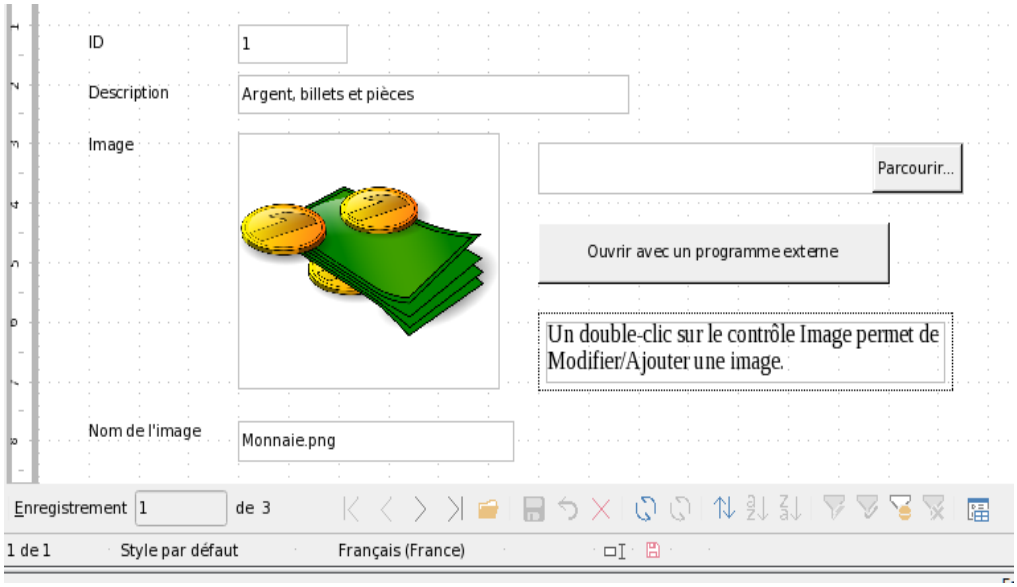
- Plus les documents sont volumineux, plus la base de données devient lourde. Par conséquent, pour les documents volumineux, une base de données externe est préférable à une base de données interne.
- Tout comme les images, les documents ne sont pas consultables. Ils sont stockés sous forme de données binaires et peuvent donc être placés dans un champ image.
- Les documents lus dans la base de données interne HSQLDB ne peuvent être lus qu'à l'aide de macros. Vous ne pouvez pas le faire avec des requêtes SQL.

Les macros suivantes pour la lecture d'entrée et de sortie dépendent d'une table qui comprend une description des données et le nom de fichier d'origine, ainsi qu'une version binaire du fichier. Le nom de fichier n'est pas automatiquement stocké avec le fichier, mais il peut fournir des informations utiles sur le type de données stockées dans un fichier qui doit être lu. Ce n'est qu'alors que le fichier peut être lu en toute sécurité par d'autres programmes.

La table contient les champs suivants :

Nom du champ	***Type du champ***	***Description***
ID	Integer	ID est la clé primaire de cette table.
Description	Zone Texte	Description du document, termes de recherche, etc.
Image	Contrôle picto	L'image ou le fichier sous forme binaire.
NomImage	Zone Texte	Le nom du fichier, y compris le suffixe du fichier. Important pour la lecture ultérieure.

Le formulaire de lecture des fichiers entrants et sortants ressemble à ceci :



Si des fichiers image sont présents dans la base de données, ils peuvent être visualisés dans le contrôle graphique du formulaire. Tous les autres types de fichiers sont masqués.

La macro suivante, pour lire un fichier, est déclenchée par **Propriétés> Sélection de fichier > Evénements> Texte modifié.**

```
SUB EntreeFichier_AvecNom(oEvent AS OBJECT)
    DIM oFormulaire AS OBJECT
    DIM oChamp AS OBJECT
    DIM oChamp2 AS OBJECT
    DIM oChamp3 AS OBJECT
    DIM oStream AS OBJECT
    DIM oSimpleFileAccess AS OBJECT
    DIM stUrl AS STRING
    DIM stNom AS STRING
    oChamp = oEvent.Source.Model
    oFormulaire = oChamp.Parent
    oChamp2 = oFormulaire.getByName("txt_NomFichier")
    oChamp3 = oFormulaire.getByName("imgImage")
    IF oChamp.Text <> "" THEN
        stUrl = ConvertToUrl(oChamp. Text)
        ar = split(stUrl,"/")
        stNom = ar(UBound(ar))
```

```
        oChamp2. BoundField.updateString(stNom)
        oSimpleFileAccess = createUnoService("com.sun.star.ucb. SimpleFileAccess")
        oStream = oSimpleFileAccess.openFileRead(stUrl)
        oChamp3.BoundField.updateBinaryStream(oStream, oStream.getLength())
    END IF
END SUB
```

Comme l'événement déclencheur de la macro fournit le nom d'un autre champ de formulaire, il n'est pas nécessaire de vérifier si les champs se trouvent dans le formulaire principal ou un sous-formulaire. Il suffit que tous les champs soient de la même forme.

Le champ "**txt_NomFichier**" stocke le nom du fichier à rechercher. Dans le cas des images, ce nom doit être saisi à la main sans utiliser de macro. Ici, à la place, le nom de fichier est déterminé via une URL et saisi automatiquement lorsque les données sont lues.

Le champ "**imgImage**" stocke les données réelles à la fois pour les images et pour les autres fichiers.

Le chemin complet, y compris le nom de fichier, est lu à partir du champ de sélection de fichier à l'aide de **oChamp.Text**. Pour garantir que l'URL n'est pas affectée par les conditions spécifiques au système d'exploitation, le texte qui a été lu est converti au format d'URL standard à l'aide de ***ConvertToUrl***. Cette URL universellement valide est divisée dans un tableau. Le séparateur est /. Le dernier élément du chemin est le nom du fichier. **Ubound(ar)** donne l'index de ce dernier élément. Le nom de fichier réel peut ensuite être lu en utilisant **ar(Ubound(ar))** et transféré dans le champ sous forme de chaîne.

La lecture du fichier lui-même nécessite le **Service UNO com.sun.star.ucb.SimpleFileAccess**. Ce service peut lire le contenu du fichier sous forme de flux de données. Celui-ci est stocké temporairement dans l'objet **oStream** puis inséré en tant que flux de données dans le champ lié au champ "**imgImage**" de la table. Cela nécessite la longueur du flux de données à fournir ainsi que l'objet **oStream**.

Les données sont maintenant à l'intérieur du champ de formulaire comme avec une entrée normale. Cependant, si le formulaire est simplement fermé à ce stade, les données ne sont pas stockées. Le stockage nécessite d'appuyer sur le bouton Enregistrer l'enregistrement de la barre de navigation, cela se produit également automatiquement lors du passage à l'enregistrement suivant.

Déterminer les noms des fichiers image

Dans la méthode ci-dessus, il a été brièvement mentionné que le nom du fichier utilisé pour l'entrée dans un contrôle graphique ne peut pas être déterminé directement. Voici une macro pour déterminer ce nom de fichier, qui correspond au formulaire ci-dessus. Le nom de fichier ne peut pas être déterminé avec certitude par un événement directement lié au contrôle graphique. Par conséquent, la macro est lancée en utilisant **Propriétés du formulaire> Événements> Avant l'action d'enregistrement.**

```
SUB LireNomImage(oEvent AS OBJECT)
    oFormulaire = oEvent.Source
    IF InStr(oFormulaire.ImplementationName, "ODatabaseForm") THEN
        oChamp = oFormulaire.getByName("imgImage")
        oChamp2 = oFormulaire.getByName("txtNomFichier")
        IF oChamp.ImageUrl <> "" THEN
            stUrl = ConvertToUrl(oChamp.ImageUrl)
            ar = split(stUrl,"/")
            stNom = ar(UBound(ar))
            oChamp2.BoundField.updateString(stNom)
        END IF
    END IF
END SUB
```

Avant l'action d'enregistrement, deux implémentations avec des noms d'implémentation différents sont exécutées. Le formulaire est le plus facilement accessible à l'aide de l'implémentation **ODatabaseForm**.

Dans le contrôle graphique, l'URL de la source de données est accessible à l'aide de **ImageUrl**. Cette URL est lue, le nom de fichier est déterminé à l'aide de la procédure précédente *EntreeFichier_AvecNom*, et est transféré dans le champ *txtNomFichier*.

Suppression des noms de fichiers d'image de la mémoire

Si après l'exécution de la macro ci-dessus, vous passez à l'enregistrement suivant, le chemin d'accès à l'image d'origine est toujours disponible. Si un fichier non-image est maintenant lu à l'aide du champ de sélection de fichier, le nom de fichier de l'image écrasera le nom de ce fichier, sauf si vous utilisez la macro suivante.

Malheureusement, le chemin d'accès ne peut pas être supprimé par la macro précédente, car le fichier image n'est lu que lorsque l'enregistrement est enregistré. Supprimer le chemin à ce stade supprimerait l'image.

La macro est lancée à l'aide de **Propriétés du formulaire> Événements> Après l'action d'enregistrement.**

```
SUB ReinitialiseNomImage(oEvent AS OBJECT)
    oFormulaire = oEvent.Source
    IF InStr(oFormulaire.ImplementationName, "ODatabaseForm") THEN
        oChamp = oFormulaire.getByName("imgImage")
        IF oChamp.ImageUrl <> "" THEN
            oChamp.ImageUrl = ""
        END IF
    END IF
END SUB
```

Comme dans la procédure "*LireImages*", le contrôle graphique est accessible. S'il existe une entrée dans ImageUrl, elle est supprimée.

Lire et afficher des images et des documents

Pour les fichiers non graphiques et les images de taille d'origine, le bouton **Ouvrir un fichier avec un programme externe** doit être enfoncé. Ensuite, les fichiers du dossier temporaire peuvent être lus et affichés à l'aide du programme lié au suffixe de fichier dans le système d'exploitation.

La macro est lancée en utilisant **Propriétés : Bouton> Evénements> Exécuter l'action.**

```
SUB LireImage_avecNom(oEvent AS OBJECT)
    DIM oDoc AS OBJECT
    DIM oDrawpage AS OBJECT
    DIM oFormulaire AS OBJECT
    DIM oChamp AS OBJECT
    DIM oChamp2 AS OBJECT
    DIM oStream AS OBJECT
    DIM oShell AS OBJECT
    DIM oChemin AS OBJECT
    DIM oSimpleFileAccess AS OBJECT
    DIM stNom AS STRING
    DIM stChemin AS STRING
    DIM stChamp AS STRING
    oFormulaire = oEvent.Source.Model.Parent
    oChamp = oFormulaire.getByName("imgImage")
    oChamp2 = oFormulaire.getByName("txtNomImage")
    stNom = oChamp2.Text
    IF stNom = "" THEN
        stNom = "DbImage"
    END IF
    oStream = oChamp.BoundField.getBinaryStream
    oChemin = createUnoService("com.sun.star.util. PathSettings")
    stChemin = oChemin.Temp & "/" & stNom
    oSimpleFileAccess = createUnoService("com.sun.star.ucb.SimpleFileAccess")
    oSimpleFileAccess.writeFile(stChemin, oStream)
    oShell = createUnoService("com.sun.star.system.SystemShellExecute")
```

```
    stChamp = convertToUrl(stChemin)
    oShell.execute(stChamp,,0)
END SUB
```

La position des autres champs concernés dans le formulaire est donnée par le bouton. S'il manque un nom de fichier, le fichier reçoit simplement le nom "Fichier".
Le contenu du champ de formulaire "**imgImage**" correspond à celui du champ Image de la table. Il est lu comme un flux de données. Le chemin d'accès au dossier temporaire est utilisé comme chemin pour ces données, il peut être défini en utilisant **Outils> Options> LibreOffice> Chemins**. Si les données doivent ensuite être utilisées à d'autres fins, et pas seulement affichées, elles peuvent être copiées à partir de ce chemin. Dans la macro, le fichier est ouvert directement après une lecture réussie, en utilisant le programme qui a été lié au suffixe de fichier par le système d'exploitation.

Toutes ces procédures font partie du fichier Exemple_Connexions_Images.odb dans la base d'exemples disponible avec ce manuel.

Extraits de code

Ces extraits de code proviennent de demandes faites dans des listes de diffusion. Des problèmes particuliers surgissent qui pourraient peut-être être utiles comme solutions pour vos propres expériences de base de données.

Obtenir l'âge actuel de quelqu'un

Une requête doit calculer l'âge réel d'une personne à partir d'une date de naissance. Voir également les fonctions dans l'annexe à ce Guide de base.

```
SELECT DATEDIFF('yy',"DateNais", CURDATE()) AS "Age" FROM "Individus"
```

Cette requête donne l'âge comme une différence en années. Mais, l'âge d'un enfant né le 31 décembre 2011 serait donné à 1 an au 1er janvier 2012, car il s'agit d'une année bissextile. Nous devons donc également tenir compte de la position de la journée dans l'année. Ceci est accessible en utilisant la fonction DAYOFYEAR(). Une autre fonction effectuera la comparaison.

```
SELECT CASEWHEN (DAYOFYEAR("Datenais") > DAYOFYEAR(CURDATE()),
      DATEDIFF ('yy',"Datenais", CURDATE())-1,
      DATEDIFF ('yy',"Datenais", CURDATE()))
AS "Age" FROM "Individus"
```

Maintenant, nous obtenons l'âge actuel correct en années.

CASEWHEN peut également être utilisé pour faire apparaître le texte DateNais aujourd'hui dans un autre champ, si `DAYOFYEAR("DateNais") = DAYOFYEAR(CURDATE()).`

Une subtile objection pourrait alors surgir : "Et les années bissextiles?". Pour les personnes nées après le 28 février, il y aura une erreur d'un jour. Pas un problème sérieux dans l'usage quotidien, mais ne devrions-nous pas rechercher la précision ?

```
CASEWHEN (
   (MONTH("DateNais") > MONTH(CURDATE())) OR
   ((MONTH("DateNais") = MONTH(CURDATE())) AND
   (DAY("DateNais") >   DAY(CURDATE()))),
DATEDIFF('yy',"DateNais", CURDATE())-1,
DATEDIFF('yy',"DateNais", CURDATE()))
```

Le code ci-dessus atteint cet objectif. Tant que le mois de la date de naissance est supérieur au mois en cours, la fonction de différence d'année soustrait un an. De même, un an sera soustrait lorsque les deux mois sont identiques, mais le jour du mois de la date de naissance est supérieur au jour de la date actuelle. Malheureusement, cette formule n'est pas compréhensible par l'interface graphique. Seule une *commande SQL directe* gérera cette requête avec succès et cela empêcherait notre requête d'être modifiée. Mais la requête doit être modifiable, voici donc comment tromper l'interface graphique :

```
CASE
   WHEN MONTH("DateNais") > MONTH(CURDATE())
      THEN DATEDIFF('yy',"DateNais", CURDATE())-1
   WHEN (MONTH("DateNais") = MONTH(CURDATE()) AND
      DAY("DateNais") > DAY(CURDATE()))
      THEN DATEDIFF('yy',"DateNais", CURDATE())-1
   ELSE DATEDIFF('yy',"DateNais", CURDATE())
END
```

Avec cette formulation, l'interface graphique ne réagit plus avec un message d'erreur. L'âge est maintenant donné avec précision même pendant les années bissextiles et la requête reste modifiable.

Affichage des anniversaires qui auront lieu dans les prochains jours

À l'aide d'un petit extrait de calcul, nous pouvons déterminer à partir de la table qui fêtera son anniversaire dans les huit prochains jours.

```
SELECT *
FROM "Table"
WHERE
   DAYOFYEAR("Date") BETWEEN DAYOFYEAR(CURDATE()) AND
      DAYOFYEAR(CURDATE()) + 7
   OR DAYOFYEAR("Date") < 7 –
      DAYOFYEAR(CAST(YEAR(CURDATE())||'-12-31' AS DATE)) +
      DAYOFYEAR(CURDATE())
```

La requête affiche tous les enregistrements dont la date d'entrée se situe entre le jour actuel de l'année et les 7 jours suivants.

Pour afficher 8 jours même à la fin d'une année, le jour du début de l'année doit être soigneusement vérifié. Cette vérification se produit uniquement pour les numéros de jour qui sont au plus 7 jours après le dernier numéro de jour de l'année en cours (généralement 365) plus le numéro de jour de la date actuelle. Si la date actuelle est à plus de 7 jours de la fin de l'année, le total est <1. Aucun enregistrement dans la table n'a une date comme celle-là, donc dans de tels cas, cette condition partielle n'est pas remplie.

Dans la formule ci-dessus, les années bissextiles donneront un résultat erroné, car leurs dates sont déplacées par l'occurrence du 29 février. Le code doit être plus complet pour éviter cette erreur :

```
SELECT *
FROM "Table"
WHERE
   CASE
      WHEN
         DAYOFYEAR(CAST(YEAR("Date")||'-12-31' AS DATE)) = 366
         AND DAYOFYEAR("Date") > 60 THEN DAYOFYEAR("Date") – 1
      ELSE
         DAYOFYEAR("Date")
   END
BETWEEN
   CASE
      WHEN
         DAYOFYEAR(CAST(YEAR(CURDATE())||'-12-31' AS DATE)) = 366
         AND DAYOFYEAR(CURDATE()) > 60 THEN DAYOFYEAR(CURDATE()) – 1
      ELSE
         DAYOFYEAR(CURDATE())
   END
AND
   CASE
      WHEN
         DAYOFYEAR(CAST(YEAR(CURDATE())||'-12-31' AS DATE)) = 366
```

```
            AND DAYOFYEAR(CURDATE()) > 60 THEN DAYOFYEAR(CURDATE()) + 6
         ELSE
            DAYOFYEAR(CURDATE()) + 7
      END
   OR DAYOFYEAR("DateNais") < 7 –
      DAYOFYEAR(CAST(YEAR(CURDATE())||'-12-31' AS DATE)) +
      DAYOFYEAR(CURDATE())
```

Les années bissextiles peuvent être reconnues en ayant 366 comme nombre total de jours plutôt que 365. Ceci est utilisé pour la détermination correspondante.

D'une part, chaque valeur de date doit être testée pour voir si elle se situe dans une année bissextile, et également pour le décompte correct pour le 60e jour (31 jours en janvier et 29 en février). Dans ce cas, toutes les valeurs `DAYOFYEAR` suivantes pour la date doivent être augmentées de 1. Ensuite, le 1er mars d'une année bissextile correspondra exactement au 1er mars d'une année normale.

D'autre part, l'année en cours `(CURDATE())` doit être testée pour voir s'il s'agit en fait d'une année bissextile. Ici aussi, le nombre de jours doit être augmenté de 1.

L'affichage de la valeur finale pour les 8 prochains jours n'est pas non plus si simple, car l'année n'est toujours pas incluse dans la requête. Cependant, ce serait une condition facile à ajouter : `YEAR("Date") = YEAR(CURDATE())` pour l'année courante ou `YEAR("Date") = YEAR(CURDATE()) + 1` pour l'année en cours.

Ajout de jours à la valeur de la date

Lors du prêt de supports, la bibliothèque peut souhaiter connaître le jour exact où le support doit être retourné. Malheureusement, le HSQLDB interne ne fournit pas la fonction `DATEADD()` qui est disponible dans de nombreuses bases de données externes et également dans Firebird interne. Voici une manière détournée d'y parvenir pour une durée limitée.

Tout d'abord, un tableau est créé contenant une séquence de dates couvrant la période souhaitée. Pour cela, Calc est ouvert et le nom "ID" est placé dans le champ A1 et "Date" dans le champ B1. Dans le champ A2, nous saisissons 1 et dans le champ B2 la date de début, par exemple 15/01/2015. Sélectionnez A2 et B2 et faites-les glisser vers le bas. Cela créera une séquence de nombres dans la colonne A et une séquence de dates dans la colonne B.

Ensuite, tout ce tableau, y compris les en-têtes de colonnes, est sélectionné et importé dans Base : **clic droit> Coller> Nom de la table> Dates**. **Sous Options, Définition et données et Utiliser la première ligne comme noms de colonne** sont validés par un clic. Toutes les colonnes sont transférées. Après cela, assurez-vous que le champ ID a le type Integer [INTEGER] et le champ Date le type Date [DATE]. Une clé primaire n'est pas nécessaire, car les enregistrements ne seront pas modifiés ultérieurement. Puisqu'une clé primaire n'a pas été définie, la table est protégée en écriture.

Conseil

Vous pouvez également utiliser une technique de requête pour créer une telle vue. Si vous utilisez une table de filtrage, vous pouvez même contrôler la date de début et la plage de valeurs de date.

```
SELECT DISTINCT CAST
   ("Y"."No" +
      (SELECT "Annee" FROM "Filter" WHERE "ID" = True) – 1 || '-' ||
   CASEWHEN("M"."No" < 10, '0' || "M"."No", '' || "M"."No") || '-' ||
   CASEWHEN("D"."No" < 10, '0' || "D"."No", '' || "D"."No")
   AS DATE) AS "Date"
FROM "Noto31" AS "D", "Noto31" AS "M", "Noto31" AS "Y"
```

```
WHERE "Y"."No" <= (SELECT "Annee" FROM "Filter" WHERE "ID" = True) AND
"M"."No" <= 12 AND "D"."No" <= 31
```

Cette vue accède à une table **"Noto31"** qui contient uniquement les nombres de 1 à 31 et est protégée en écriture. Un autre tableau de filtre contient l'année de départ et la plage d'année que la vue doit couvrir. La date est assemblée à partir de ceux-ci, créant une expression de date (année, mois, jour) dans le texte, qui peut ensuite être convertie en date. HSQLDB accepte tous les jours jusqu'à 31 par mois et des chaînes comme le 31/02/2015. Cependant, le 31/02/2015 est transmis en tant que 3/03/2015. Par conséquent, lors de la préparation de la vue, vous devez utiliser DISTINCT pour exclure les valeurs de date en double.

Ici, la vue suivante est effective :

```
SELECT "a"."Date",
   (SELECT COUNT(*) FROM "Vue_Date" WHERE "Date" ≤ "a"."Date")
      AS "lfdNo"
FROM "Vue_Date" AS "a"
```

Grâce à la numérotation des lignes, la valeur de la date est convertie en un nombre. Comme vous ne pouvez pas supprimer des données dans une vue, aucune protection supplémentaire en écriture n'est nécessaire.

En utilisant une requête, nous pouvons maintenant déterminer une date spécifique, par exemple la date dans 14 jours :

```
SELECT "a"."Date_Pret",
   (SELECT "Date" FROM "Date" WHERE "ID" =
      (SELECT "ID" FROM "Date" WHERE "Date" = "a"."Date_Pret")+14)
      AS "Date_Retour"
FROM "Prets" AS "a"
```

La première colonne indique la date du prêt. Cette colonne est accessible par une sous-requête de corrélation qui est à nouveau divisée en deux requêtes. `SELECT "ID" FROM "Date"` donne la valeur du champ ID, correspondant à la date d'émission. 14 jours sont ajoutés à la valeur. Le résultat est affecté au champ ID par la sous-requête externe. Ce nouvel identifiant détermine ensuite la date qui entre dans le champ de date.

Malheureusement dans l'affichage de cette requête, le type de date n'est pas automatiquement reconnu, de sorte qu'il devient nécessaire d'utiliser le formatage. Dans un formulaire, l'affichage correspondant peut être stocké, de sorte que chaque requête produira une valeur de date.

Une variante directe pour déterminer la valeur de la date est possible en utilisant un moyen plus court :

```
SELECT "Date_Pret",
   DATEDIFF('dd', '1899-12-30', "Date_Pret") + 14
      AS "Date_Retour"
FROM "Prets"
```

La valeur numérique renvoyée peut être mise en forme dans un formulaire en tant que date, à l'aide d'un champ formaté. Cependant, il faut beaucoup de travail pour le rendre disponible pour un traitement SQL ultérieur dans une requête.

Ajout d'une heure à un horodatage

MySQL a une fonction appelée TIMESTAMPADD(). Une fonction similaire n'existe pas dans HSQLDB. Mais la valeur numérique interne de l'horodatage peut être utilisée pour faire l'addition ou la soustraction, en utilisant un champ formaté dans un formulaire.

Contrairement à l'ajout de jours à une date, les heures posent un problème qui peut ne pas être évident au début.

```
SELECT "HoroDatage"
   DATEDIFF('ss', '1899-12-30', "HoroDatage") / 86400.0000000000 +
```

```
        36/24 AS "HoroDatage+36 heures"
FROM "Table"
```

La nouvelle heure calculée est basée sur la différence avec l'heure zéro du système. Comme dans les calculs de date, il s'agit de la date du 30/12/1899.

Note

La date zéro du 30/12/1899 est censée avoir été choisie, car l'année 1900, contrairement à la plupart des années divisibles par 4, n'était pas une année bissextile. La balise "1" du calcul interne a donc été replacée au 31/12/1899 et non au 01/01/1900.

La différence est exprimée en secondes, mais le nombre interne compte les jours sous forme de nombres avant la virgule décimale et les heures, minutes et secondes sous forme de décimales. Puisqu'un jour contient 60 * 60 * 24 secondes, le deuxième décompte doit être divisé par 86400 pour pouvoir calculer correctement les jours et les fractions de jours. Si la HSQLDB interne doit donner des décimales, elles doivent être incluses dans le calcul, donc au lieu de 86400, nous devons diviser par 86400.0000000000. Les décimales dans une requête doivent utiliser un point décimal comme séparateur, quelles que soient les conventions locales. Le résultat aura 10 décimales après le point.

À ce résultat, il faut ajouter le nombre total d'heures sous forme de fraction de journée. Le chiffre calculé, correctement formaté, peut être créé dans la requête. Malheureusement, le formatage n'est pas enregistré, mais il peut être transféré avec le format correct à l'aide d'un champ formaté dans un formulaire ou un état.

Si des minutes ou des secondes doivent être ajoutées, veillez à ce qu'elles soient fournies sous forme de fractions de jour.

Si la date tombe entre novembre, décembre, janvier, etc., le calcul ne pose aucun problème. Ils semblent assez précis : ajouter 36 heures à un horodatage du 20/01/2015 13:00:00 donne 22/01/2015 00:00:00. Mais les choses sont différentes pour le 20/04/2015 13:00:00. Le résultat est le 22/04/2015 00:00:00. Le calcul tourne mal à cause de l'heure d'été. L'heure "perdue" ou "gagnée" par le changement d'heure n'est pas prise en compte. Dans un même fuseau horaire, il existe différentes manières d'obtenir un résultat "correct". Voici une variation simple :

```
SELECT "HoroDatage"
   DATEDIFF('dd', '1899-12-30', "HoroDatage") +
      HOUR("HoroDatage") / 24.0000000000 +
      MINUTE("HoroDatage") / 1440.0000000000 +
      SECOND("HoroDatage") / 86400.0000000000 +
      36/24
         AS "HoroDatage+36hours"
FROM "Table"
```

Au lieu de compter les heures, les minutes et les secondes depuis l'origine de la date, elles sont comptées à partir de la date actuelle. Le 20/05/2015, l'heure est 13:00 mais sans l'heure d'été, elle sera indiquée comme 12:00. La fonction HEURE prend en compte l'heure d'été et donne 13 heures comme partie horaire de l'heure. Cela peut ensuite être ajouté correctement à la partie quotidienne. Les minutes et les secondes sont traitées exactement de la même manière. Enfin, les heures supplémentaires sont ajoutées sous forme de partie fractionnaire d'une journée et le tout est affiché sous forme d'horodatage calculé à l'aide du formatage de cellule.

Deux choses doivent être gardées en vue dans ce calcul :

- Lors du passage de l'heure d'hiver à l'heure d'été, les valeurs horaires ne s'affichent pas correctement. Cela peut être corrigé à l'aide d'un tableau auxiliaire, qui prend les dates de début et de fin de l'heure d'été et corrige le décompte horaire. Une affaire un peu compliquée.

- L’affichage des heures n’est possible qu’avec des champs formatés. Le résultat est un nombre décimal, pas un horodatage qui pourrait être stocké directement en tant que tel dans la base de données. Soit il doit être copié dans le formulaire, soit converti d’un nombre décimal en un horodatage à l’aide d’une requête complexe. Le point de rupture dans la conversion est la valeur de la date, car des années bissextiles ou des mois avec différents nombres de jours peuvent être impliqués.

Obtenir un solde courant par catégories

Au lieu d’utiliser un livre de ménage, une base de données sur un PC peut simplifier la tâche fastidieuse consistant à additionner les dépenses de nourriture, de vêtements, de transport, etc. Nous voulons que la plupart de ces détails soient immédiatement visibles dans la base de données, notre exemple suppose donc que les revenus et les dépenses seront stockés sous forme de valeurs signées dans un champ appelé Montant. En principe, le tout peut être étendu pour couvrir des champs séparés et une sommation appropriée pour chacun.

```
SELECT "ID", "Montant", (SELECT SUM("Montant")   FROM "Especes"
      WHERE "ID" <= "a"."ID") AS "Balance"
   FROM "Especes" AS "a" ORDER BY "ID" ASC
```

Cette requête entraîne pour chaque nouvel enregistrement un calcul direct du solde courant du compte. Dans le même temps, la requête reste modifiable, car le champ Balance est créé via une sous-requête de corrélation. La requête dépend de l’ID de clé primaire créé automatiquement pour calculer l’état du compte. Cependant, les soldes sont généralement calculés sur une base quotidienne. Nous avons donc besoin d’une requête de date.

```
SELECT "ID", "Date", "Montant", (SELECT SUM("Montant") FROM "Especes"
WHERE "Date" <= "a"."Date") AS "Balance"
FROM "Especes" AS "a" ORDER BY "Date", "ID" ASC
```

Les dépenses apparaissent maintenant triées et additionnées par date. Reste la question de la catégorie, puisque nous voulons des soldes correspondants pour les différentes catégories de dépenses.

```
SELECT "ID", "Date", "Montant", "ID_Compte",
(SELECT "Compte" FROM "Compte" WHERE "ID" = "a"."ID_Compte") AS "Compte_Nom",
(SELECT SUM("Montant") FROM "Especes" WHERE "Date" <= "a"."Date" AND
"ID_Compte" = "a"."ID_Compte") AS "Balance",
(SELECT SUM("Montant") FROM "Especes" WHERE "Date" <= "a"."Date") AS
"Total_balance"
FROM "Especes" AS "a" ORDER BY "Date", "ID" ASC
```

Cela crée une requête modifiable dans laquelle, en plus des champs de saisie (Date, Montant, ID_Compte), le nom du compte, le solde pertinent et le solde total apparaissent ensemble. Comme les sous-requêtes corrélées sont partiellement basées sur les entrées précédentes ("Date" <= "a". "Date"), seules les nouvelles entrées seront traitées sans problème. Les modifications apportées à un enregistrement précédent ne sont initialement détectables que dans cet enregistrement. La requête doit être mise à jour si des calculs ultérieurs qui en dépendent doivent être effectués.

Numérotation des lignes

Les champs incrémentés automatiquement sont très utiles. Cependant, ils ne vous indiquent pas avec certitude combien d’enregistrements sont présents dans la base de données ou sont réellement disponibles pour être interrogés. Les enregistrements sont souvent supprimés et de nombreux utilisateurs essaient en vain de déterminer quels numéros ne sont plus présents afin de faire correspondre le numéro courant.

```
SELECT "ID", (SELECT COUNT("ID") FROM "Table" WHERE "ID" <= "a"."ID") AS "No."
FROM "Table" AS "a"
```

Le champ ID est lu et le second champ est déterminé par une sous-requête corrélée, qui cherche à déterminer combien de valeurs de champ dans ID sont inférieures ou égales à la valeur de champ actuelle. À partir de là, un numéro de ligne courant est créé.

Chaque enregistrement auquel vous souhaitez appliquer cette requête contient des champs. Pour appliquer cette requête aux enregistrements, vous devez d'abord ajouter ces champs à la requête. Vous pouvez les placer dans l'ordre de votre choix dans la clause SELECT. Si vous disposez des enregistrements dans un formulaire, vous devez modifier le formulaire afin que les données du formulaire proviennent de cette requête.

Par exemple, l'enregistrement contient champ1, champ2 et champ3. La requête complète serait :

```
SELECT "ID", "Champ1", "Champ2", "Champ3", (SELECT COUNT("ID") FROM "Table"
WHERE "ID" <= "a"."ID") AS "No." FROM "Table" AS "a"
```

Une numérotation pour un groupement correspondant est également possible :

```
SELECT "ID", "Calcul", (SELECT COUNT("ID") FROM "Table" WHERE "ID" <= "a"."ID"
AND "Calcul" = "a"."Calcul") AS "No." FROM "Table" AS "a" ORDER BY "ID" ASC,
"No." ASC
```

Ici, une table contient différents nombres calculés. ("Calcul"). Pour chaque nombre calculé, "No." est exprimé séparément dans l'ordre croissant après le tri sur le champ ID. Cela produit une numérotation à partir de 1.

Si l'ordre de tri réel dans la requête correspond aux numéros de ligne, un type de tri approprié doit être mappé. Pour cela, le champ de tri doit avoir une valeur unique dans tous les enregistrements. Sinon, deux emplacements auront la même valeur. Cela peut en fait être utile si, par exemple, l'ordre de passage dans un concours doit être représenté, car des résultats identiques conduiront alors à une position commune. Pour que l'ordre de place soit exprimé de telle manière que, en cas de positions conjointes, la valeur suivante soit omise, la requête doit être construite différemment :

```
SELECT "ID", (SELECT COUNT("ID") + 1 FROM "Table" WHERE "Temps" < "a"."Temps")
AS "Place" FROM "Table" AS "a"
```

Toutes les entrées sont évaluées pour lesquelles le champ Heure a une valeur plus petite. Cela couvre tous les athlètes qui ont atteint le poste de vainqueur avant l'athlète actuel. A cette valeur s'ajoute le chiffre 1. Celui-ci détermine la place de l'athlète actuel. Si le temps est identique à celui d'un autre athlète, ils sont placés conjointement. Cela permet de passer des commandes telles que la 1ère place, la 2e place, la 2e place, la 4e place.

Ce serait plus problématique si les numéros de ligne étaient nécessaires ainsi qu'un numéro d'emplacement. Cela peut être utile si plusieurs enregistrements doivent être combinés sur une seule ligne.

```
SELECT "ID", (SELECT COUNT("ID") + 1 FROM "Table" WHERE "Temps" < "a"."Temps")
AS "Place",
CASE WHEN
(SELECT COUNT("ID") + 1 FROM "Table" WHERE "Temps" = "a"."Temps") = 1
THEN (SELECT COUNT("ID") + 1 FROM "Table" WHERE "Temps" < "a"."Temps")
ELSE (SELECT (SELECT COUNT("ID") + 1 FROM "Table" WHERE "Temps" < "a"."Temps")
+ COUNT("ID") FROM "Table" WHERE "Temps" = "a"."Temps" "ID" < "a"."ID"
END
AS "LineNumber" FROM "Table" AS "a"
```

La deuxième colonne donne toujours l'ordre de la place. La troisième colonne vérifie d'abord si une seule personne a franchi la ligne avec ce temps. Si tel est le cas, la commande passée est directement convertie en un numéro de ligne. Sinon, une valeur supplémentaire est ajoutée à la commande. Pour le même temps ("Heure" = "a"."Heure") au moins 1 est ajouté, s'il y a une autre personne avec l'ID de clé primaire, dont la clé primaire est plus petite que la clé primaire dans l'enregistrement actuel (" ID "<" a "." ID "). Cette requête donne donc des valeurs identiques pour la commande à condition qu'aucune deuxième personne avec la même heure n'existe. Si une deuxième personne avec la même heure existe, l'ID détermine quelle personne a le numéro de ligne le plus petit.

Incidemment, ce tri par numéro de ligne peut servir quel que soit le but souhaité par les utilisateurs de la base de données. Par exemple, si une série d'enregistrements est triée par nom, les enregistrements portant le même nom ne sont pas triés au hasard mais en fonction de leur clé primaire, qui est bien entendu unique. De cette manière également, la numérotation peut conduire à un tri des enregistrements.

La numérotation des lignes est également un bon prélude à la combinaison d'enregistrements individuels en un seul enregistrement. Si une requête de numérotation de ligne est créée en tant que vue, une autre requête peut lui être appliquée sans créer de problème. À titre d'exemple simple, voici à nouveau la première requête de numérotation avec un champ supplémentaire :

```
SELECT "ID", "Nom", (SELECT COUNT("ID") FROM "Table" WHERE "ID" <= "a"."ID")
AS "No." FROM "Table" AS "a"
```

Cette requête est transformée en vue Vue1. La requête peut être utilisée, par exemple, pour rassembler les trois premiers noms sur une seule ligne :

```
SELECT "Nom" AS "Nom_1", (SELECT "Nom" FROM "Vue1" WHERE "No." = 2) AS
"Nom_2", (SELECT "Nom" FROM "Vue1" WHERE "No." = 3) AS "Nom_3" FROM "Vue1"
WHERE "No." = 1
```

De cette manière, plusieurs enregistrements peuvent être convertis en champs adjacents. Cette numérotation va simplement du premier au dernier enregistrement.

Si tous ces individus doivent se voir attribuer le même nom de famille, cela peut être effectué comme suit :

```
SELECT "ID", "Nom", "Nom", (SELECT COUNT("ID") FROM "Table" WHERE "ID" <=
"a"."ID" AND "Nom" = "a"."Nom") AS "No." FROM "Table" AS "a"
```

Maintenant que la vue a été créée, la famille peut être assemblée.

```
SELECT "Nom", "Nom" AS "Nom_1", (SELECT "Nom" FROM "Vue1" WHERE "No." = 2 AND
"Nom" = "a"."Nom") AS "Nom_2", (SELECT "Nom" FROM "Vue1" WHERE "No." = 3 AND
"Nom" = "a"."Nom") AS "Nom_3" FROM "Vue1" AS "a" WHERE "No." = 1
```

De cette manière, dans un carnet d'adresses, tous les membres d'une même famille ("Nom") peuvent être rassemblés afin que chaque adresse ne soit prise en compte qu'une seule fois lors de l'envoi d'une lettre, mais tous ceux qui devraient recevoir la lettre sont répertoriés.

Nous devons être prudents ici, car nous ne voulons pas d'une fonction en boucle sans fin. La requête de l'exemple ci-dessus limite à 3 les enregistrements parallèles qui doivent être convertis en champs. Cette limite a été choisie délibérément. Aucun autre nom n'apparaîtra même si la valeur de "No." est supérieur à 3.

Dans quelques cas, une telle limite est clairement compréhensible. Par exemple, si nous créons un calendrier, les lignes peuvent représenter les semaines de l'année et les colonnes les jours de la semaine. Comme dans le calendrier d'origine, seule la date détermine le contenu du champ, la numérotation des lignes est utilisée pour numéroter les jours de chaque semaine en continu, puis les semaines de l'année deviennent les enregistrements. Cela nécessite que la table contienne un champ de date avec des dates continues et un champ pour les événements. De plus, la date la plus ancienne créera toujours un "No." = 1. Donc, si vous voulez que le calendrier commence le lundi, la première date doit être le lundi. La colonne 1 est alors lundi, la colonne 2 mardi et ainsi de suite. La sous-requête se termine alors par "No." = 7. De cette manière, les sept jours de la semaine peuvent être affichés côte à côte et une vue de calendrier correspondante créée.

Obtenir un saut de ligne dans une requête

Parfois, il est utile d'assembler plusieurs champs à l'aide d'une requête et de les séparer par des sauts de ligne, par exemple lors de la lecture d'une adresse complète dans un rapport.

Le saut de ligne dans la requête est représenté par `Char (13)`. Exemple :

```
SELECT "Prenom"||' '||"Nom"||Char(13)||"Rue"||Char(13)||"Ville" FROM "Table"
```

Cela donne :

Prenom Nom
Rue
Ville

Une telle requête, avec une numérotation de ligne jusqu'à 3, vous permet d'imprimer des étiquettes d'adresse sur trois colonnes en créant un rapport. La numérotation est nécessaire à cet égard pour que trois adresses puissent être placées l'une à côté de l'autre dans un enregistrement. C'est la seule façon dont ils resteront côte à côte lorsqu'ils seront lus dans le rapport.

Dans certains systèmes d'exploitation (Windows, notamment), il est nécessaire d'inclure char (10) à côté de char (13) dans le code.

```
SELECT "Prenom"||' '||"Nom"||Char(13)||Char(10)||"Rue"||Char(13)||
Char(10)||"Ville" FROM "Table"
```

Regrouper et résumer

Pour les autres bases de données et pour les versions plus récentes de HSQLDB, la commande **Group_Concat()** est disponible. Elle peut être utilisée pour regrouper des champs individuels dans un enregistrement en un seul champ. Ainsi, par exemple, il est possible de stocker les prénoms et noms de famille dans une table, puis de présenter les données de telle sorte qu'un champ affiche les noms de famille comme noms de famille tandis qu'un deuxième champ contient tous les prénoms pertinents de manière séquentielle, séparés par des virgules.

Cet exemple est similaire à bien des égards à la numérotation des lignes. Le regroupement dans un champ commun est une sorte de complément à cela.

NomDeFamille	***Prenom***
Titegoutte	Anne
Térieur	Alain
Titegoutte	Emma
Térieur	Alex
Titegoutte	Justine
Titegoutte	Corinne
Titegoutte	Germaine

est converti par la requête en :

NomDeFamille	***Prenom***
Titegoutte	Anne, Emma, Justine, Corinne, Germaine
Térieur	Alain, Alex

Cette procédure peut, dans certaines limites, être exprimée en HSQLDB. L'exemple suivant fait référence à une table appelée Noms avec les champs ID, Prénom et Nom. La requête suivante est d'abord exécutée sur la table et enregistrée en tant que vue appelée Vue_Groupe.

```
SELECT "Nom", "Prenom", (SELECT COUNT("ID") FROM "Nom" WHERE "ID" <= "a"."ID"
AND "Nom" = "a"."Nom") AS "GroupNo" FROM "Nom" AS "a"
```

Vous pouvez lire dans le chapitre Requêtes comment cette requête accède au contenu du champ dans la même ligne de requête. Il produit une séquence numérotée ascendante, regroupée par

nom. Cette numérotation est nécessaire pour la requête suivante, de sorte que dans l'exemple, un maximum de 5 prénoms soit répertorié.

```
SELECT "Nom",
(SELECT "Prenom" FROM "Vue_Groupe" WHERE "Nom" = "a"."Nom" AND "GroupNo" = 1)
||
IFNULL((SELECT ', ' || "Prenom" FROM "Vue_Groupe" WHERE "Nom" = "a"."Nom" AND
"GroupNo" = 2), '') ||
IFNULL((SELECT ', ' || "Prenom" FROM "Vue_Groupe" WHERE "Nom" = "a"."Nom" AND
"GroupNo" = 3), '') ||
IFNULL((SELECT ', ' || "Prenom" FROM "Vue_Groupe" WHERE "Nom" = "a"."Nom" AND
"GroupNo" = 4), '') ||
IFNULL((SELECT ', ' || "Prenom" FROM "Vue_Groupe" WHERE "Nom" = "a"."Nom" AND
"GroupNo" = 5), '')
AS "Prenoms"
FROM "Vue_Groupe" AS "a"
```

À l'aide de sous-requêtes, les prénoms des membres du groupe sont recherchés l'un après l'autre et combinés. À partir de la deuxième sous-requête, vous devez vous assurer que les valeurs "NULL" ne définissent pas la combinaison entière sur "NULL". C'est pourquoi un résultat chaîne vide «» plutôt que "NULL" est affiché.

Annexe

Texte des requêtes SQL

Ces requêtes sont celles présentées dans les copies d'écran du chapitre Recherche de données.

Requête LOCATE – page 10

```
SELECT "ID", "Memo",
   LOCATE(LOWER(: TexteCherche), LOWER ("Memo")) AS "Position"
FROM "Table"
```

Requête cible - page 10

```
SELECT "ID", "Memo", "Position",
   CASE
      WHEN "Position" = 0 THEN '** Non trouvé **'
      WHEN "Position" < 10 THEN SUBSTRING ("Memo", 1, 25)
      ELSE SUBSTRING ("Memo", LOCATE(' ', "Memo", "Position" – 10) + 1, 25)
END AS "Cible"
FROM
   (SELECT "ID", "Memo", LOCATE(LOWER(: TexteCherche), LOWER ("Memo")) AS
"Position"
FROM "Table")
```

Cible sans coupures – page 12

```
SELECT "ID", "Memo", "Position",
   CASE
      WHEN "Position" = 0 THEN '** Non trouvé **'
      WHEN "Position" < 10 THEN SUBSTRING ("Memo",1, LOCATE (' ',"Memo", 25))
      ELSE SUBSTRING ("Memo", LOCATE (' ', "Memo", "Position" – 10) + 1,
         (LOCATE (' ', "Memo", "Position" + 20) -
            (LOCATE (' ', "Memo", "Position" – 10) + 1)))
   END AS "Cible"
FROM
   (SELECT "ID", "Memo", LOCATE(LOWER(: TexteCherche), LOWER("Memo")) AS
"Position"
FROM "Table")
```

Cibles multiples

```
SELECT "ID", "Memo", "Position01", "Cible01", "Position02",
```

```
   CASE
      WHEN "Position02" = 0 THEN '** Non trouvé **'
      WHEN "Position02" < 10 THEN SUBSTRING ("Memo",1, LOCATE (' ',"Memo",
25))
      ELSE SUBSTRING ("Memo", LOCATE (' ', "Memo", "Position02" - 10) + 1,
         (LOCATE (' ', "Memo", "Position02" + 20) -
            (LOCATE (' ', "Memo", "Position02" - 10) + 1))
         )
   END AS "Cible02",
   CASE
      WHEN "Position02" = 0 THEN 0
      ELSE LOCATE(LOWER(: TexteCherche), LOWER("Memo"), "Position02" + 1)
   END AS "Position03"
FROM
   (SELECT "ID", "Memo", "Position01",
   CASE
      WHEN "Position01" = 0 THEN '** Non trouvé **'
      WHEN "Position01" < 10 THEN SUBSTRING ("Memo",1, LOCATE (' ',"Memo",
25))
      ELSE SUBSTRING ("Memo", LOCATE (' ', "Memo", "Position01" - 10) + 1,
         (LOCATE (' ', "Memo", "Position01" + 20) -
            (LOCATE (' ', "Memo", "Position01" - 10) + 1))
         )
   END AS "Cible01",
   CASE
      WHEN "Position01" = 0 THEN 0
      ELSE LOCATE(LOWER(: TexteCherche), LOWER("Memo"), "Position01" + 1)
   END AS "Position02"
   FROM
   (SELECT "ID", "Memo", LOCATE(LOWER(: TexteCherche), LOWER("Memo")) AS
"Position01"
   FROM "Table")
   )
```