LibreOffice

Guide Base

Chapitre 5
Requêtes

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
Dan Lewis

Des éditions précédentes

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

Traduction

Jean-Michel Coste

Relecteurs

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 6.4.

Information

Pour l’apprentissage du langage, il y a quantité de cours disponibles.

Un exemple :https://sql.sh/2143-cours-sql-pdf de cours de Tony Archambeau licence Creative Commons Attribution.

Table de matières

Droits d'auteur....2
Contributeurs....2
De cette édition....2
Traduction....2
Relecteurs....2
Retour d'information....2
Date de publication et version du logiciel....2
Information....2
Informations générales sur les requêtes....4
Saisie de requêtes....4
Création de requêtes à l'aide de la boîte de dialogue Ébauche de requête....4
Utilisation de fonctions dans une requête....12
Définition de la relation dans la requête....16
Définition des propriétés de la requête....19
Amélioration des requêtes à l'aide du mode SQL....21
Utilisation d'un alias dans une requête....30
Requêtes pour la création de champs de list box....31
Requêtes comme base d'informations supplémentaires dans les formulaires....33
Possibilités de saisie de données dans les requêtes....34
Utilisation des paramètres dans les requêtes....38
Sous-requêtes....39
Sous-requêtes corrélées....40
Requêtes comme tables source pour les requêtes....40
Récapituler les données avec des requêtes....44
Accès plus rapide aux requêtes à l'aide des vues de table....45
Erreurs de calcul dans les requêtes....46
Annexe : Notation BNF....49

Informations générales sur les requêtes

Les requêtes vers une base de données sont l'outil le plus puissant dont nous disposons pour utiliser les bases de données de manière pratique. Elles peuvent rassembler les données de différentes tables, calculer les résultats si nécessaire et filtrer rapidement des enregistrements spécifiques à partir d'une masse de données. Les grandes bases de données Internet que les gens utilisent chaque jour existent principalement pour fournir un résultat rapide et pratique à l'utilisateur à partir d'une énorme quantité d'informations par une sélection judicieuse de mots-clés – y compris les publicités liées à la recherche qui encouragent les gens à faire des achats.

Saisie de requêtes

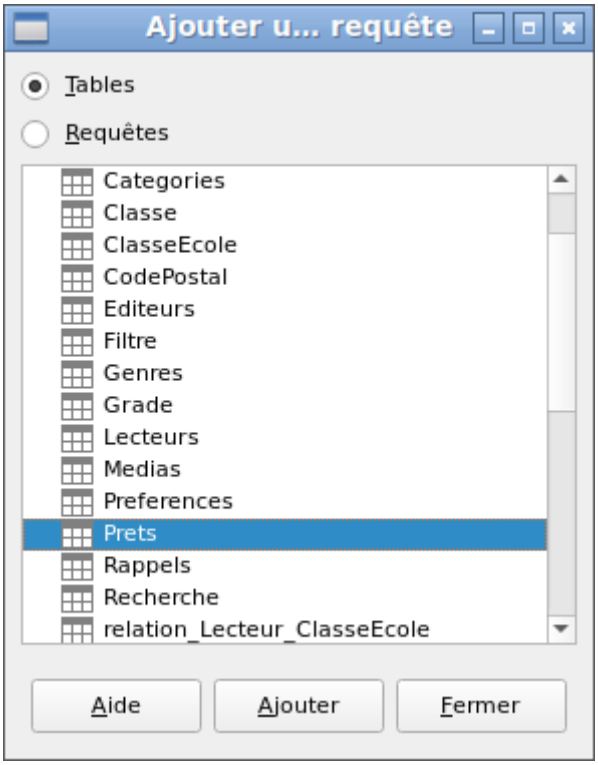
Les requêtes peuvent être saisies à la fois dans l'interface graphique et directement sous forme de code SQL. Dans les deux cas, une fenêtre s'ouvre, dans laquelle vous pouvez créer une requête et également la corriger si nécessaire.

Création de requêtes à l'aide de la boîte de dialogue Ébauche de requête

La création de requêtes à l'aide de l'assistant est brièvement décrite au chapitre 8 du Guide de mise en route, Débuter avec Base. Ici, nous allons expliquer la création directe de requêtes en mode Design.

Dans la fenêtre principale de la base de données, cliquez sur l'icône **Requêtes** dans la section Bases de données, puis dans la section Tâches, cliquez sur **Créer une requête en mode Ébauche**. Deux boîtes de dialogue apparaissent. L'une fournit la base pour une création en vue de conception de la requête ; l'autre sert à ajouter des tables, des vues ou des requêtes à la requête courante.

Comme notre formulaire simple fait référence à la table Prets, nous allons d'abord expliquer la création d'une requête à l'aide de cette table.



Dans les tables disponibles, sélectionnez la table Prets. Cette fenêtre permet de combiner plusieurs tables (ainsi que des vues et des requêtes). Pour sélectionner une table, cliquez sur son nom, puis sur le bouton Ajouter.
Ou bien, double-cliquez sur le nom de la table. Les deux méthodes ajoutent la table à la zone graphique de la boîte de dialogue Ébauche de requête (Figure 1).

Lorsque toutes les tables nécessaires ont été sélectionnées, cliquez sur le bouton Fermer. Des tables et des requêtes supplémentaires peuvent être ajoutées ultérieurement si nécessaire. Cependant, aucune requête ne peut être créée sans au moins une table, une sélection doit donc être effectuée pour commencer.

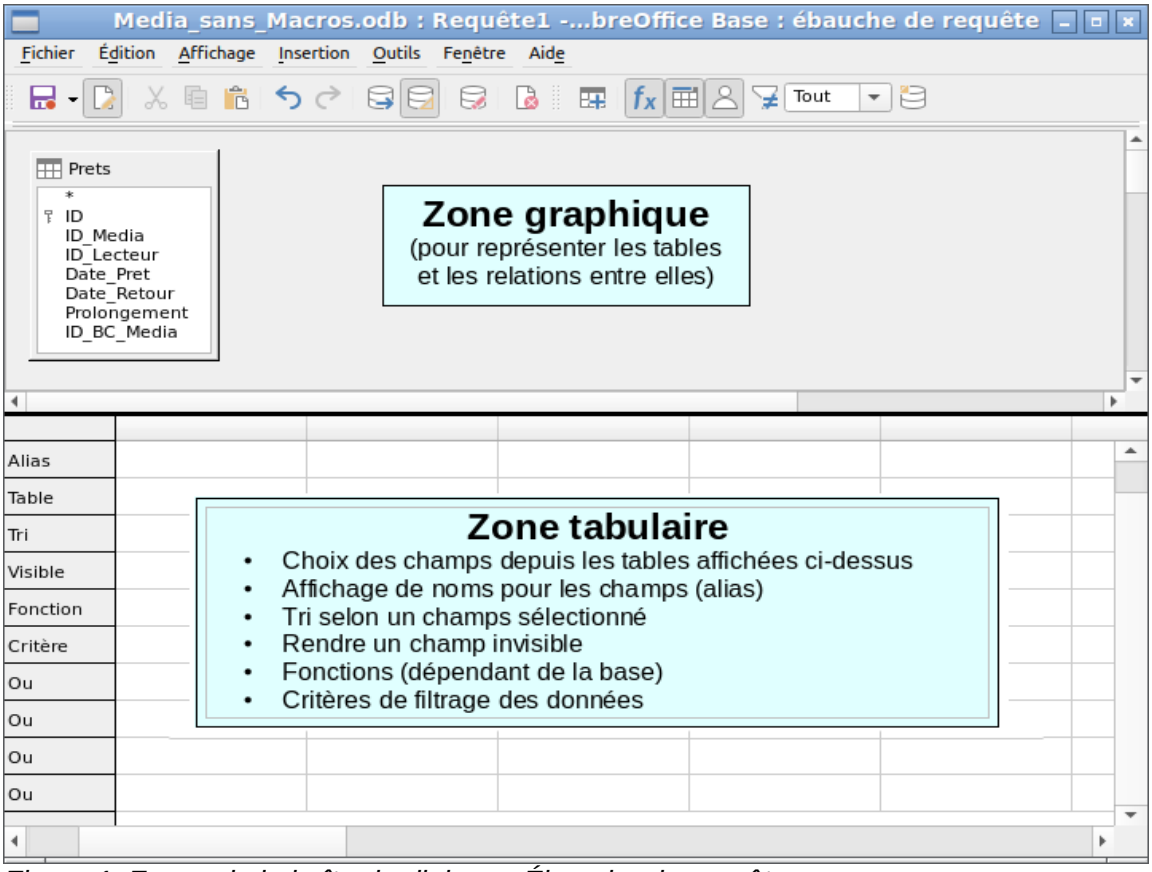
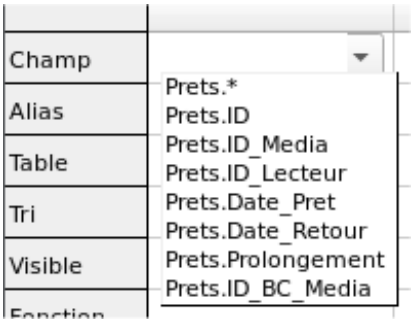


Figure 1: Zones de la boîte de dialogue Ébauche de requête

La Figure 1 affiche les divisions de base de la boîte de dialogue Ébauche de requête : la zone graphique affiche les tables à lier dans la requête. Leurs relations les unes avec les autres par rapport à la requête peuvent également être affichées. La zone tabulaire sert à la sélection des champs à afficher, ou à la définition des conditions liées à ces champs.

Cliquez sur le champ dans la première colonne de la zone tabulaire pour faire apparaître une flèche vers le bas. Cliquez sur cette flèche pour ouvrir la liste déroulante des champs disponibles. Le format est Nom_Table. Nom_Champ – c'est pourquoi tous les noms de champ ici commencent par le mot Prets.

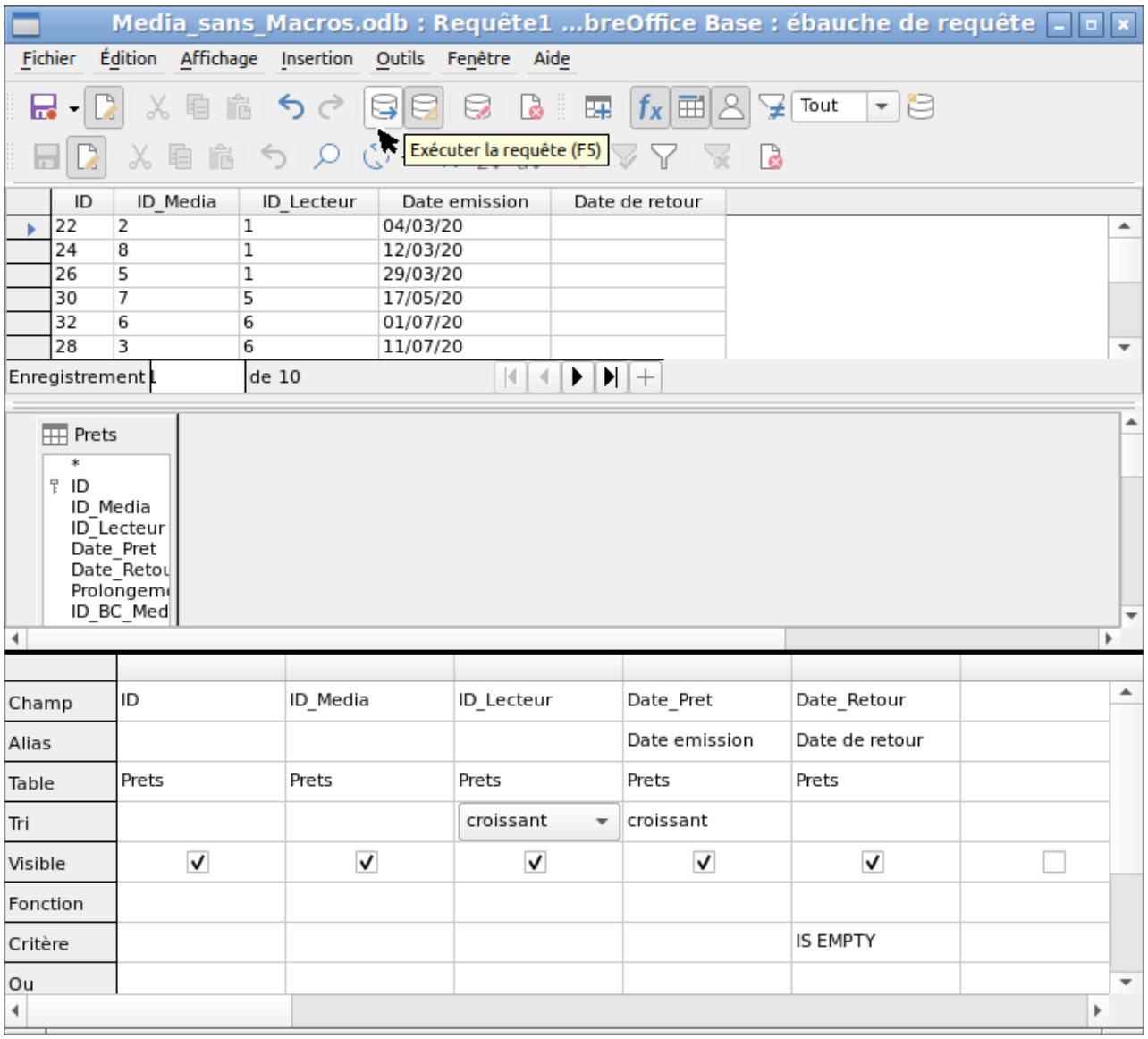


La désignation de champ sélectionnée Prets.* A une signification particulière. Ici, un clic vous permet d'ajouter tous les champs de la table sous-jacente à la requête. Lorsque vous utilisez cette désignation de champ avec le caractère générique * pour tous les champs, la requête ne peut plus être distinguée de la table.

Conseil

Pour un transfert rapide de tous les champs d'une table dans une requête, il suffit de cliquer sur l'étoile dans la vue de table dans l'interface graphique (Prets.* ci-dessus).

Un double-clic sur un champ insère ce champ dans la zone tabulaire à la prochaine position libre.



Les cinq premiers champs de la table Prêt sont sélectionnés. Les requêtes en mode Ébauche peuvent toujours être exécutées en tant que tests (icône Excétuter la requete). Cela fait apparaître une vue tabulaire des données au-dessus de la vue graphique de la table Prêt avec sa liste de champs (pour fermer cette fenêtre, faire F4 ou Affichage/Apercu). Un test d'exécution d'une requête est toujours utile avant de l'enregistrer, pour clarifier si la requête atteint réellement son objectif. Souvent, une erreur logique empêche une requête de récupérer des données. Dans d'autres cas, il peut arriver que les enregistrements que vous souhaitiez exclure s'affichent malencontreusement.

En principe, une requête qui produit un message d'erreur dans la base de données sous-jacente ne peut pas être enregistrée tant que l'erreur n'est pas corrigée.

	ID	ID_Media
▶	1	1
	2	6
	3	3
	9	5
	10	4
✱	<Autc	

Figure 2: Une requête modifiable

	ID_Media	ID_Lecteur
▶	2	1
	8	1
	5	1
	7	5
	6	6
	3	6

Figure 3: Une requête non modifiable

Dans le test ci-dessus, portez une attention particulière à la première colonne du résultat de la requête. Le marqueur d'enregistrement actif (ici flèche verte, mais peut être d'une couleur différente) apparaît toujours sur le côté gauche du tableau, pointant ici sur le premier enregistrement comme enregistrement actif. Alors que le premier champ du premier enregistrement de la figure 2 est mis en évidence ou que le curseur clignote dans la cellule, le champ correspondant de la figure 3 montre uniquement une bordure en pointillés. Le surlignage, ou le clignotement du curseur, indique que ce champ peut être modifié. Les enregistrements, en d'autres termes, sont modifiables. La bordure en pointillés indique que ce champ ne peut pas être modifié. La figure 2 contient également une ligne supplémentaire pour l'entrée d'un nouvel enregistrement, avec le champ ID déjà marqué comme <AutoChamp>. Cela montre également que de nouvelles entrées sont possibles.

Conseil

Une règle de base est qu'aucune nouvelle entrée n'est possible si la clé primaire de la table interrogée n'est pas incluse dans la requête.

Champ	ID	ID_Media	ID_Lecteur	Date_Pret	Date_Retour
Alias				Date emission	Date de retour

Les champs Date_Pret et Date_Retour reçoivent des alias. Cela ne les conduit pas à être renommés mais uniquement à apparaître sous ces noms pour l'utilisateur de la requête.

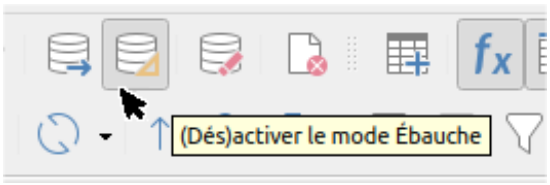
ID	ID_Media	ID_Lecteur	Date emission	Date de retour
22	2	1	04/03/20	

Le tableau ci-dessus montre comment les alias remplacent les noms de champ réels.

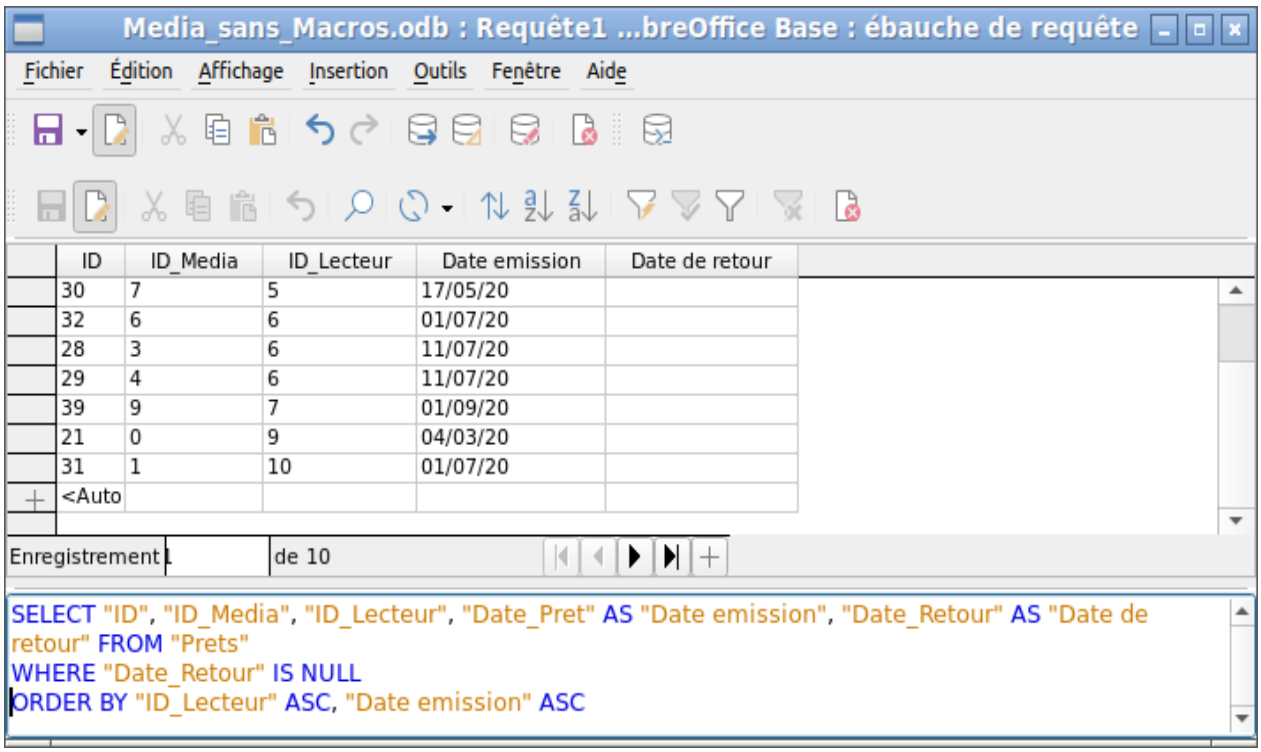
Date_Retour
Date de retour
Prets
☑
IS EMPTY

Le champ Date_Retour reçoit non seulement un alias, mais aussi un critère de recherche, ce qui entraînera l'affichage uniquement des enregistrements pour lesquels le champ Date_Retour est

vide. (Entrez `IS EMPTY` dans le ligne de critère du champ Date_Retour.) Ce critère d'exclusion entraînera uniquement l'affichage des enregistrements relatifs aux supports qui n'ont pas encore été renvoyés.



Pour mieux apprendre le langage SQL, il vaut la peine de basculer de temps en temps entre le mode conception et le mode SQL en désactivant le mode ébauche.



Ici, la formule SQL créée par nos choix précédents est révélée. Pour faciliter la lecture, des sauts de ligne ont été inclus. Malheureusement, l'éditeur ne stocke pas ces sauts de ligne, donc lorsque la requête est à nouveau appelée, elle apparaîtra comme une seule ligne continue jusqu'aux bords de la fenêtre.

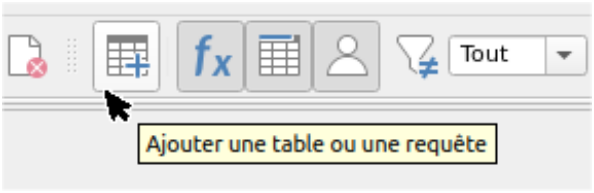
SELECT commence les critères de sélection. AS spécifie les alias de champ à utiliser. FROM montre la table qui doit être utilisée comme source de la requête. WHERE donne les conditions de la requête, à savoir que le champ Date_Retour doit être vide (IS NULL). ORDER BY définit les critères de tri, à savoir l'ordre croissant (ASC – croissant ou ascending) pour les deux champs *ID_Lecteur* et *Date emission*. Cette spécification de tri illustre comment l'alias du champ *Date_Pret* peut être utilisé dans la requête elle-même.

Conseil

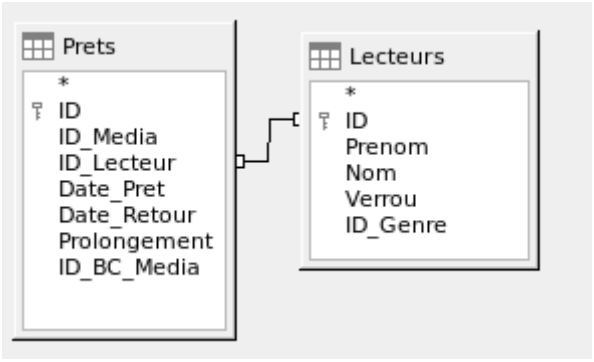
Lorsque vous travaillez en mode d'affichage de conception, utilisez IS EMPTY pour exiger qu'un champ soit vide. Lorsque vous travaillez en mode SQL, utilisez IS NULL, ce qui est requis par SQL (Structured Query Language).

Lorsque vous souhaitez trier par ordre décroissant à l'aide de SQL, utilisez DESC au lieu de ASC.

Jusqu'à présent, les champs ID_Media et ID_Lecteur ne sont visibles que sous forme de champs numériques. Les noms des lecteurs ne sont pas écrits en clair. Pour les afficher dans une requête, la table Lecteurs doit être incluse dans la requête. Pour cela, nous retournons au mode Ébauche. Ensuite, une nouvelle table peut être ajoutée à la vue conception de requête.



Ici, d'autres tables ou requêtes peuvent être ajoutés ultérieurement et rendues visibles dans l'interface utilisateur graphique. Si des liens entre les tables ont été déclarés au moment de leur création (voir Chapitre 3, Tables), alors ces tables sont affichées avec les liens directs correspondants.



Si un lien est absent, il peut être créé à ce stade en faisant glisser la souris de "Prets". "ID_Lecteur" vers "Lecteurs". "ID".

Note

La liaison des tables ne fonctionne pour le moment que dans la base de données interne ou dans les bases de données relationnelles externes. Par exemple, les tables d'une feuille de calcul ne peuvent pas être liés entre elles. Elles doivent d'abord être importés dans une base de données interne.

Pour créer un lien entre les tables, une simple importation suffit sans création supplémentaire de clé primaire.

Désormais, les champs de la table Lecteurs peuvent être saisis dans la zone tabulaire. Les champs sont initialement ajoutés à la fin de la requête.

ID_Lecteur	Date_Pret	Date_Retour	Prenom	Nom
	Date emission	Date de retour		
Prets	Prets	Prets	Lecteurs	Lecteurs

La position des champs peut être corrigée dans la zone tabulaire de l'éditeur à l'aide de la souris. Ainsi, par exemple, le champ Prenom a été déplacé en position directement avant le champ Date_Pret.

ID	ID_Media	ID_Lecteur	Prenom	Nom	Date emission	Date de retour
22	2	1	Lino	Ventura	04/03/20	
24	8	1	Lino	Ventura	12/03/20	
26	5	1	Lino	Ventura	29/03/20	
30	7	5	Marie	Laforêt	17/05/20	
32	6	6	Valérie	Benguigui	01/07/20	
28	3	6	Valérie	Benguigui	11/07/20	
29	4	6	Valérie	Benguigui	11/07/20	
39	9	7	Martine	Carol	01/09/20	
21	0	9	André	Raimbourg	04/03/20	

Enregistrement 1 de 10

Maintenant, les noms sont visibles. L'ID_Lecteur est devenu superflu. Le tri par Nom et Prenom a également plus de sens que le tri par ID_Lecteur.

Cette requête ne convient plus pour une utilisation en tant que requête autorisant de nouvelles entrées dans la table résultante, car il lui manque la clé primaire pour la table Lecteurs ajoutée. Les champs de la requête redeviennent modifiables uniquement si cette clé primaire est intégrée. Pour cette raison, rendre les résultats de requête modifiables est une fonctionnalité qui doit être utilisée avec une extrême prudence, si nécessaire sous le contrôle d'un formulaire.

Attention

Avoir une requête que vous pouvez modifier peut créer des problèmes. La modification des données dans la requête modifie également les données de la table sous-jacente et les enregistrements contenus dans la table. Les données peuvent ne pas avoir la même signification. Par exemple, changez le nom du lecteur, et vous avez également changé les livres que le lecteur a empruntés et retournés.

Si vous devez modifier des données, faites-le dans un formulaire afin de voir les effets de la modification des données.

Même lorsqu'une requête peut être modifiée ultérieurement, elle n'est pas si simple à utiliser comme formulaire avec des zones de liste, qui affichent les noms des lecteurs mais contiennent le ID_Lecteur de la table. Les zones de liste ne peuvent pas être ajoutées à une requête ; elles ne sont utilisables que dans les formulaires.

```
SELECT "Prets"."ID", "Prets"."ID_Media", "Prets"."ID_Lecteur", "Lecteurs"."Prenom",
"Lecteurs"."Nom", "Prets"."Date_Pret" AS "Date emission", "Prets"."Date_Retour" AS
"Date de retour"
FROM "Prets", "Lecteurs"
WHERE "Prets"."ID_Lecteur" = "Lecteurs"."ID" AND "Prets"."Date_Retour" IS NULL
ORDER BY "Prets"."ID_Lecteur" ASC, "Date emission" ASC
```

Si nous retournons maintenant à la vue SQL, nous voyons que tous les champs sont maintenant affichés entre guillemets doubles : "Nom_Table". "Nom_Champ". Ceci est nécessaire pour que la base de données sache de quelle table proviennent les champs précédemment sélectionnés. Après tout, les champs de différentes tables peuvent facilement avoir les mêmes noms de champ (ce qui n'est pas recommandé).
Dans la structure de tableau ci-dessus, cela est particulièrement vrai pour le champ ID.

Note

La requête suivante fonctionne sans placer les noms de table devant les noms de champ :

```
SELECT "ID", "Nombre", "Prix" FROM "Stock", "Distribution" WHERE
"Distribution"."ID_Stock" = "Stock"."ID"
```

Ici, l'ID est extrait de la table qui vient en premier dans la définition FROM. La définition de table dans la formule WHERE est également superflue, car ID_Stock n'apparaît qu'une seule fois (dans la table Distribution) et l'ID a été clairement extrait de la table Stock (à partir de la position de la table dans la requête).

Si un champ de la requête a un alias, il peut être référencé – par exemple lors du tri – par cet alias sans qu'un nom de table ne soit donné. Le tri est effectué dans l'interface utilisateur graphique selon la séquence des champs dans la vue tabulaire. Si à la place vous voulez trier d'abord par "Date_Pret" puis par "Prets". "ID_Lecteur", cela peut être fait si :

- La séquence des champs dans la zone de tableau de l'interface utilisateur graphique est modifiée (glisser-déposer "Date_Pret" à gauche de "Prets". "ID_Lecteur", ou
- Un champ supplémentaire est ajouté, défini pour être invisible, juste pour le tri (cependant, l'éditeur ne l'enregistrera que temporairement si aucun alias n'a été défini pour lui) [ajouter un autre champ "Date_Pret" juste avant "Prets". "ID_Lecteur" ou ajouter un autre champ "Prets". "ID_Lecteur" juste après "Date_Pret"], ou
- Le texte de la commande ORDER BY dans l'éditeur SQL est modifié en conséquence (ORDER BY "Date_Pret", "Prets.". "ID_Lecteur").

Conseil

Une requête peut nécessiter un champ qui ne fait pas partie du résultat de la requête. Dans le graphique de la section suivante, Date_Retour est un exemple. Cette requête recherche les enregistrements qui ne contiennent pas de date de retour. Ce champ fournit un critère pour la requête mais aucune donnée visible utile.

Utilisation de fonctions dans une requête

L'utilisation de fonctions permet à une requête de fournir plus qu'une simple vue filtrée des données dans une ou plusieurs tables. La requête suivante calcule le nombre de supports prêtés, en fonction du champ ID_Lecteur.

Champ	ID	ID_Lecteur	Date_Retour
Alias	Nombre		
Table	Prets	Prets	Prets
Tri			
Visible	☑	☑	☑
Fonction	Nombre	Groupe	
Critère			IS EMPTY

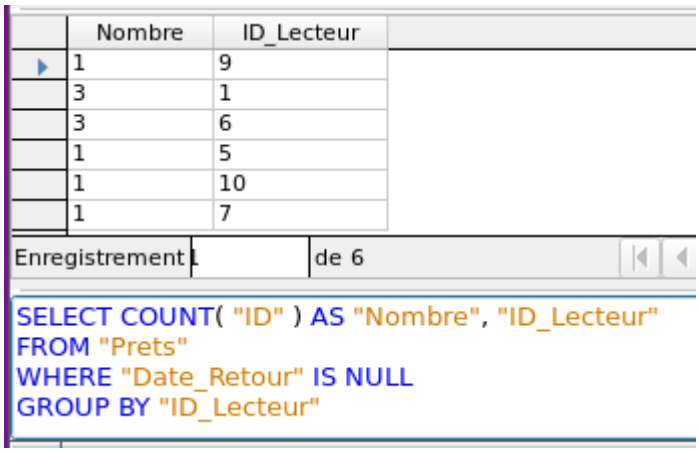
Figure 4: Fonction de comptage et de groupe

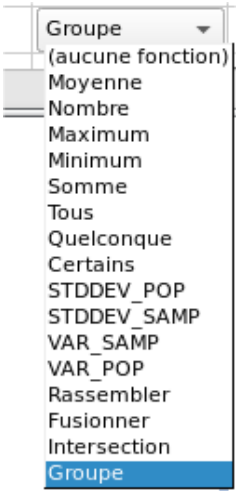
Pour l'ID de la table Prets, la fonction Nombre est sélectionnée. En principe, le champ de la table choisi pour cela ne fait aucune différence. La seule condition est la suivante : le champ ne doit être

vide dans aucun des enregistrements. Pour cette raison, le champ de clé primaire, qui n'est jamais vide, est le choix le plus approprié. Tous les enregistrements avec un contenu de champ autre que NULL sont comptés.

Pour le champ ID_Lecteur, qui donne accès aux informations du lecteur, la fonction de regroupement est choisie. De cette manière, les enregistrements avec le même ID_Lecteur sont regroupés. Le résultat montre le nombre d'enregistrements pour chaque ID_Lecteur.

En tant que critère de recherche, Date_Retour est défini sur "IS EMPTY" et en colonne non visible, comme dans l'exemple précédent. (Ci-dessous, le code SQL pour cela est WHERE "Date_Retour" IS NULL.)





Le résultat de la requête montre que ID_Lecteur '1' a un total de 3 supports en prêt. Si la fonction Nombre avait été affectée à Date_Retour au lieu de l'ID, chaque ID_Lecteur aurait un média '0' en prêt, puisque Date_Retour est prédéfini comme NULL.

La formule correspondante en code SQL est indiquée ci-dessus.

Dans l'ensemble, l'interface utilisateur graphique fournit les fonctions illustrées à droite, qui correspondent aux fonctions de la HSQLDB sous-jacente.

Pour une explication des fonctions, voir "Amélioration des requêtes à l'aide du mode SQL" en page 21.

Si un champ d'une requête est associé à une fonction, tous les champs restants mentionnés dans la requête doivent également être associés à des fonctions pour être affichés. Si cela n'est pas garanti, vous obtenez le message d'erreur suivant :

Une traduction quelque peu libre serait : L'expression suivante ne contient aucune fonction d'agrégation ou de regroupement.

Pour cette raison, dans l'exemple figurant sur la page précédente (figure 4), le champ Date_Retour n'est pas affiché.

Conseil

Lors de l'utilisation du mode d'affichage Ébauche, un champ n'est visible que si la ligne Visible contient une coche pour le champ. Lors de l'utilisation du mode SQL, un champ n'est visible que lorsqu'il suit le mot-clé SELECT.

Note

Lorsqu'aucun champ n'est associé à une fonction, le nombre de lignes dans la sortie de la requête est déterminé par les conditions de recherche. Lorsqu'un champ est associé à une fonction, le nombre de lignes dans la sortie de la requête est déterminé par l'existence d'un regroupement ou non. S'il n'y a pas de regroupement, il n'y a qu'une seule ligne dans la sortie de la requête. S'il y a regroupement, le nombre de lignes correspond au nombre de valeurs distinctes du champ de regroupement. Ainsi, tous les champs visibles doivent être associés à une fonction ou être associés à une instruction de regroupement pour éviter ce conflit dans la sortie de la requête.

Après cela, la requête complète est répertoriée dans le message d'erreur, mais malheureusement sans que le champ incriminé soit spécifiquement nommé. Dans ce cas, le champ Date_Retour a été ajouté en tant que champ affiché. Ce champ n'est associé à aucune fonction et n'est pas non plus inclus dans l'instruction de regroupement.

Les informations fournies à l'aide du bouton Plus ne sont pas très éclairantes pour l'utilisateur normal de la base de données. Il affiche simplement le code d'erreur SQL.

Pour corriger l'erreur, supprimez la coche dans la ligne Visible pour le champ Date_Retour. Sa condition de recherche (critère) est appliquée lors de l'exécution de la requête, mais elle n'est pas visible dans la sortie de la requête.

En utilisant l'interface graphique, des calculs de base et des fonctions supplémentaires peuvent être effectués.

Champ	ID	ID_Media	ID_Lecteur	Date	Nombre("Rappels"."Date")	Date_Retour
Alias	Nombre			NombreRappels	MontantRappels	
Table	Prets	Prets	Prets	Rappels		Prets
Tri						
Visible	☑	☑	☑	☑	☑	☐
Fonction	Groupe	Groupe	Groupe	Nombre		
Critère						IS EMPTY

Supposons qu'une bibliothèque n'émette pas d'avis de rappel lorsqu'un article doit être retourné, mais émet des avis de retard dans les cas où la période de prêt a expiré et que l'article n'a pas été retourné. C'est une pratique courante dans les écoles et les bibliothèques publiques qui n'accordent des prêts que pour de courtes périodes fixes. Dans ce cas, l'émission d'un avis de retard signifie automatiquement qu'une amende doit être payée. Comment calculons-nous ces amendes ?

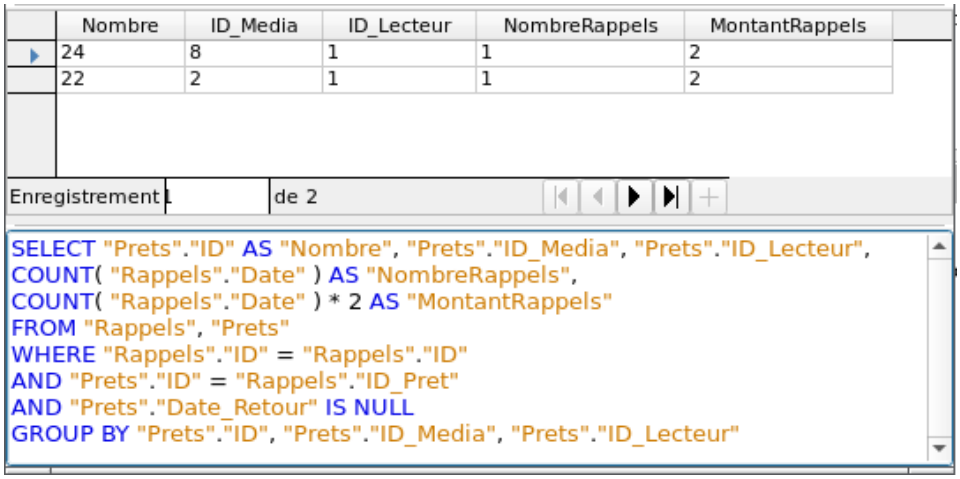
Dans la requête présentée ci-dessus, les tables Prêt et Rappels sont interrogées conjointement. À partir du nombre d'entrées de données dans le tableau Rappels, le nombre total d'avis de rappel est déterminé. L'amende pour les médias en retard est fixée dans la requête à 2,00 €. Au lieu d'un nom de champ, la désignation du champ est donnée comme Nombre (Rappels. Date) * 2. L'interface utilisateur graphique ajoute les guillemets et convertit le terme « compte » en la commande SQL appropriée.

Avertissement

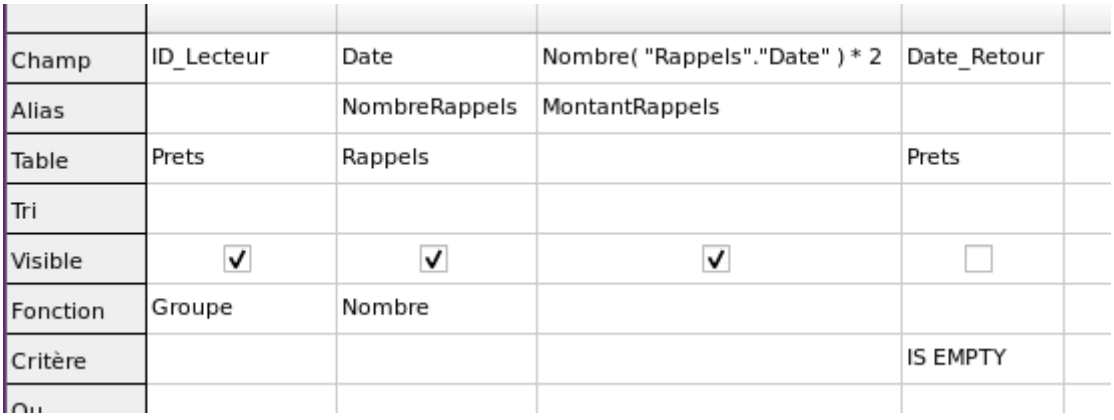
Uniquement pour les personnes qui utilisent une virgule comme séparateur décimal :

Si vous souhaitez entrer des nombres avec des décimales à l'aide de l'interface graphique, vous devez vous assurer qu'un point décimal plutôt qu'une virgule est utilisé comme séparateur décimal dans l'instruction SQL finale. Les virgules sont utilisées comme séparateurs de champ, de sorte que de nouveaux champs de requête seraient créés pour la partie décimale.

Une entrée avec une virgule dans la vue SQL mène toujours à un champ supplémentaire contenant la valeur numérique de la partie décimale.



La requête donne maintenant pour chaque support encore en prêt les amendes qui se sont accumulées, en fonction des avis de rappel émis et du champ de multiplication supplémentaire. La structure de requête suivante sera également utile pour calculer les amendes dues par les utilisateurs individuels.



Les champs "Prets"."ID" et "Prets"."ID_Media" ont été supprimés. Ils ont été utilisés dans la requête précédente pour créer, en regroupant, un enregistrement distinct pour chaque support.

Maintenant, nous allons regrouper uniquement par lecteur. Le résultat de la requête ressemble à ceci :

ID_Lecteur	NombreRappels	MontantRappels
1	2	4

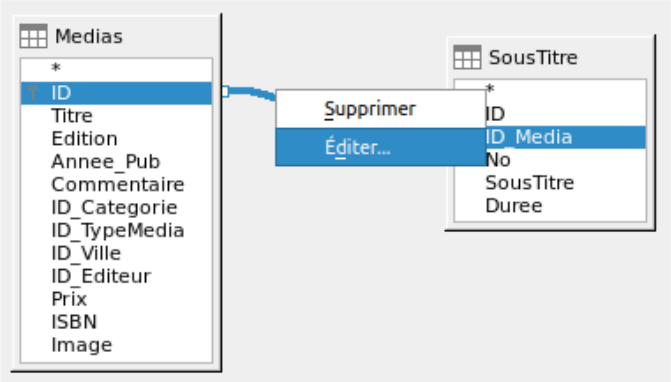
Enregistrement 1 de 1

Au lieu de lister séparément les supports pour ID_Lecteur = 1, tous les champs "Rappels", "Date" ont été comptés et le total de 4,00 € a été inscrit comme amende due.

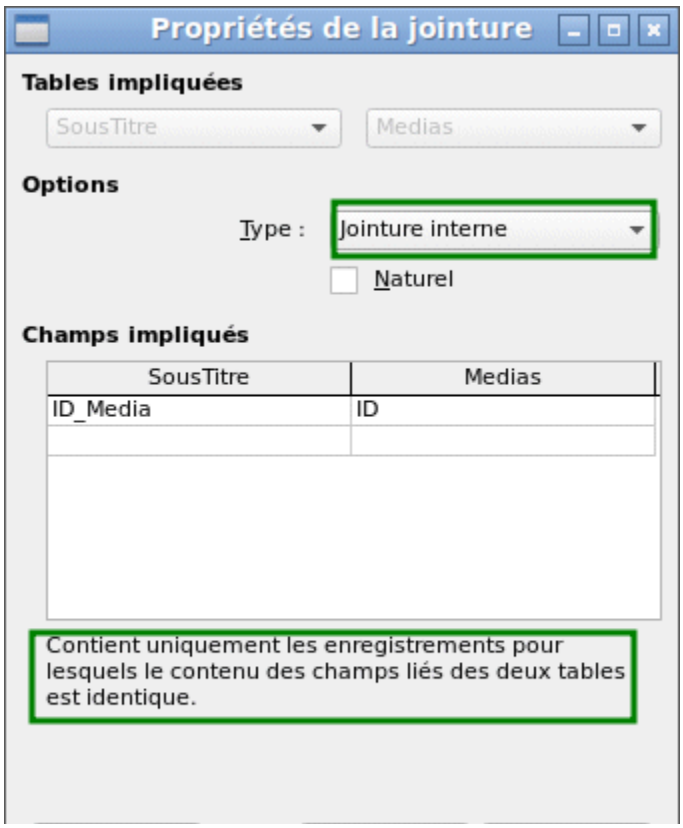
Définition de la relation dans la requête

Lorsque des données sont recherchées dans des tables ou des formulaires, la recherche est généralement limitée à une table ou à un formulaire. Même le chemin d'un formulaire principal vers un sous-formulaire n'est pas navigable par la fonction de recherche intégrée. À ces fins, les données à rechercher sont mieux collectées à l'aide d'une requête.

La requête simple pour le champ Titre de la table Medias affiche les entrées de test pour cette table, 13 enregistrements en tout. Mais si vous entrez SousTitre dans la table de requête, le contenu d'enregistrement de la table Medias est réduit à seulement 2 titres. Seulement pour ces deux titres il y a aussi des sous-titres dans le tableau. Pour tous les autres titres, aucun sous-titre n'existe. Cela correspond à la condition de jointure selon laquelle seuls les enregistrements pour lesquels le champ ID_Media de la table SousTitre est égal au champ ID de la table Medias doivent être affichés. Tous les autres enregistrements sont exclus.



Les conditions de jointure doivent être ouvertes pour être modifiées pour afficher tous les enregistrements souhaités. Nous nous référons ici non pas aux jointures entre les tables dans la conception de relations, mais aux jointures dans les requêtes.



Par défaut, les relations sont définies en tant que jointures internes. La fenêtre fournit des informations sur le fonctionnement de ce type de jointure dans la pratique.

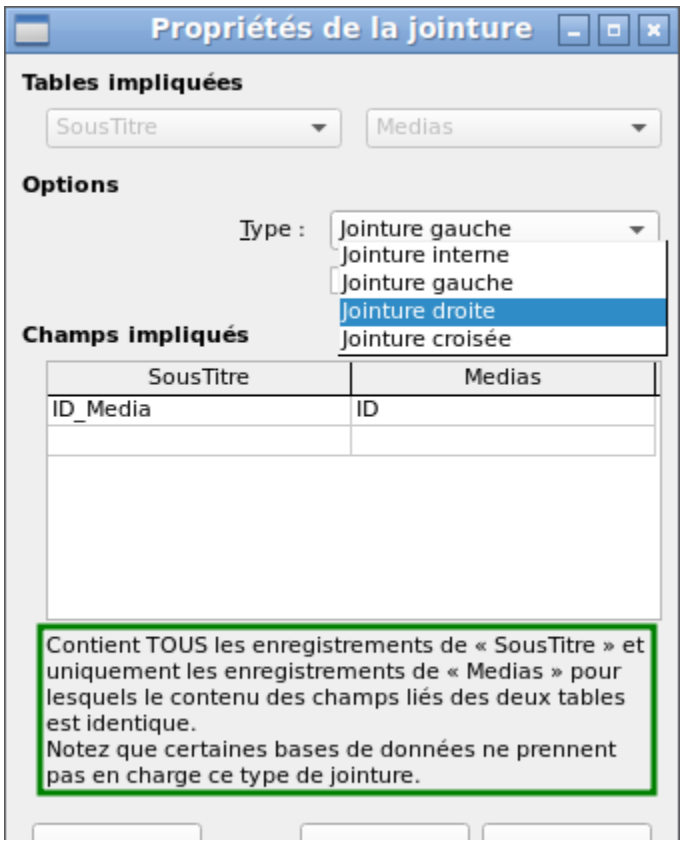
Les deux tables précédemment sélectionnées sont répertoriées comme Tables impliquées. Elles ne sont pas sélectionnables ici. Les champs pertinents des deux tables sont lus à partir des définitions de table. Si aucune relation n'est spécifiée dans la définition de table, une relation peut être créée à ce stade pour la requête. Cependant, si vous avez planifié votre base de données de manière ordonnée à l'aide de HSQLDB, il ne devrait pas être nécessaire de modifier ces champs.

Le paramètre le plus important est l'option Jointure. Ici, les relations peuvent être choisies de manière à ce que tous les enregistrements de la table SousTitres soient sélectionnés, mais uniquement les enregistrements de Médias qui ont un sous-titre entré dans la table SousTitres.

Ou vous pouvez choisir le contraire : que dans tous les cas, tous les enregistrements de la table Medias soient affichés, qu'ils aient ou non un sous-titre.

L'option Naturel spécifie que les champs liés dans les tables sont traités comme égaux. Vous pouvez également éviter d'avoir à utiliser ce paramètre en définissant correctement vos relations au tout début de la planification de votre base de données.

Pour le type Jointure droite, la description montre que tous les enregistrements de la table Medias seront affichés (SousTitre RIGHT JOIN Medias). Comme il n'y a pas de sous-titre qui n'a pas de titre dans Media mais qu'il y a certainement des titres dans Media qui manquent de sous-titre, c'est le choix droit.



Après avoir confirmé la jointure droite, les résultats de la requête semblent tels que nous les voulions. Le titre et le sous-titre sont affichés ensemble dans une seule requête. Naturellement, les titres apparaissent plus d'une fois comme avec la relation précédente. Cependant, tant que les résultats ne sont pas comptés, cette requête peut être utilisée comme base pour une fonction de recherche. Voir les fragments de code dans ce chapitre, au chapitre 8, Tâches de base de données, et au chapitre 9, Macros.

Titre	SousTitre
Bases de données avec OpenOffice.org 3	
Bilbo le Hobbit	
Bon à tirer	Un menuisier dansait
Bon à tirer	Villejuif
Bon à tirer	Pericoloso sporgersi
Bon à tirer	Le Déjeuner de soleil
Bon à tirer	L'Arabe
Émile et les Détectives	
I hear you knocking	Egg Or The Hen
I hear you knocking	Outlaw Blues
I hear you knocking	Sabre dance (Single version)
I hear you knocking	The stumble
I hear you knocking	You can't catch me
Indignez vous !	
La nouvelle orthographe	
La vie mensongère des adultes	
Le livre Postfix	
Les nuits de Reykjavik	

Enregistrement 1 de 21

Définition des propriétés de la requête

Depuis la version 4.1 de LibreOffice, il est possible de définir des propriétés supplémentaires dans l'éditeur de requêtes.

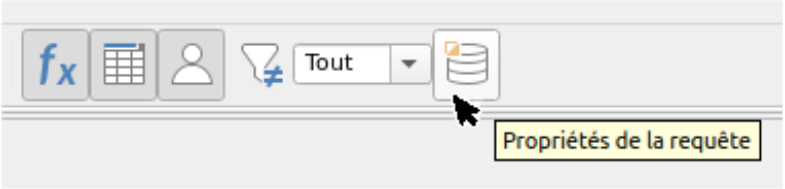
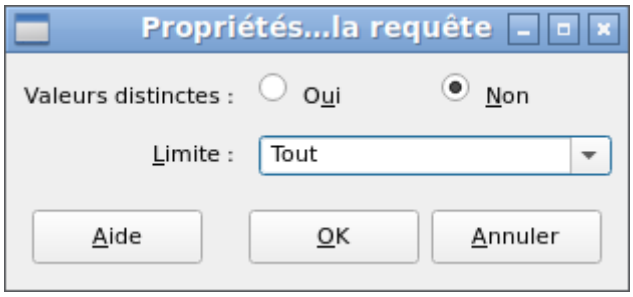


Figure 5: Ouverture des propriétés de requête dans l'éditeur de requête

À côté du bouton pour ouvrir les propriétés de la requête se trouve une zone de liste déroulante pour réguler le nombre d'enregistrements affichés ainsi qu'un bouton pour les valeurs distinctes. Ces fonctions sont répétées dans la boîte de dialogue suivante :



Le paramètre Valeurs distinctes détermine si la requête doit supprimer les enregistrements en double.

	Prenom	Nom	Date_Retour
▶	Guy	Bedos	
	Valérie	Benguigui	
	Valérie	Benguigui	
	Valérie	Benguigui	
	Martine	Carol	
	Marie	Laforêt	
	André	Raimbourg	
	Lino	Ventura	
	Lino	Ventura	
	Lino	Ventura	

Supposons qu'une requête soit faite pour déterminer quels lecteurs ont encore des articles en prêt. Leurs noms sont affichés si le champ de date de retour est vide. Les noms sont affichés plus d'une fois pour les lecteurs qui ont plusieurs articles en prêt.

	Prenom	Nom	Date_Retour
▶	Guy	Bedos	
	Valérie	Benguigui	
	Martine	Carol	
	Marie	Laforêt	
	André	Raimbourg	
	Lino	Ventura	

Si vous choisissez Valeurs distinctes, les enregistrements avec le même contenu disparaissent.

La requête ressemble alors à ceci :

```
SELECT DISTINCT
"Lecteurs"."Prenom", "Lecteurs"."Nom", "Prets"."Date_Retour"
FROM "Prets", "Lecteurs"
WHERE "Prets"."ID_Lecteur" = "Lecteurs"."ID" AND "Prets"."Date_Retour" IS NULL
ORDER BY "Lecteurs"."Nom" ASC
```

La requête d'origine :

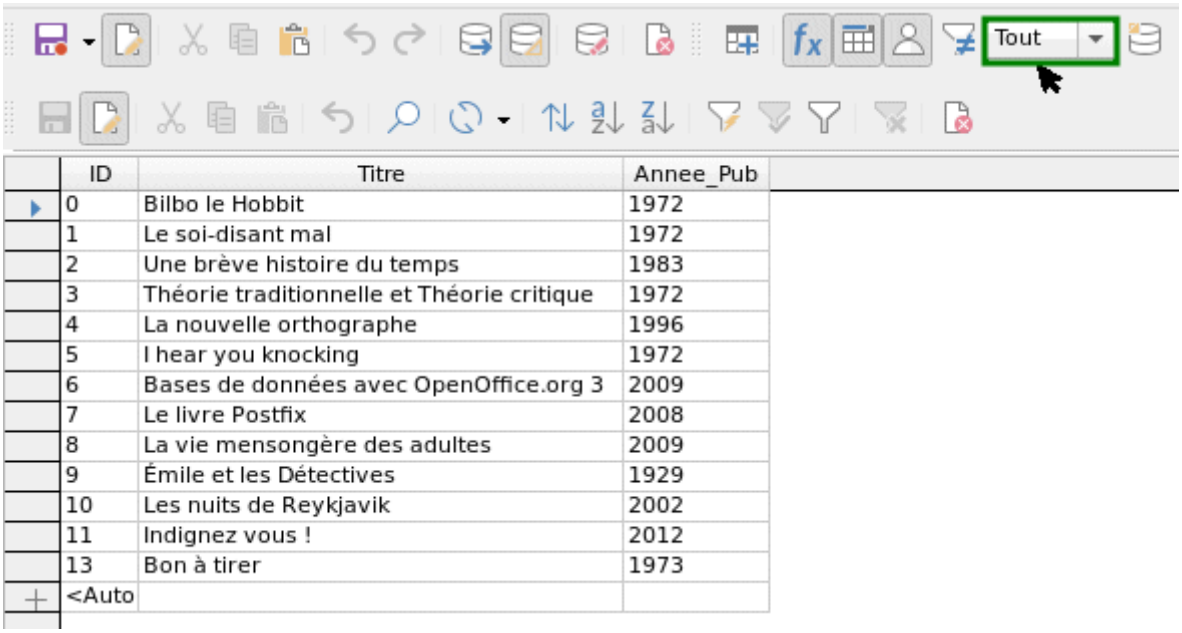
```
SELECT "Lecteurs"."Prenom", "Lecteurs"."Nom"…
```

L'ajout de DISTINCT à la requête supprime les enregistrements en double.

```
SELECT DISTINCT "Lecteurs"."Prenom", "Lecteurs"."Nom"…
```

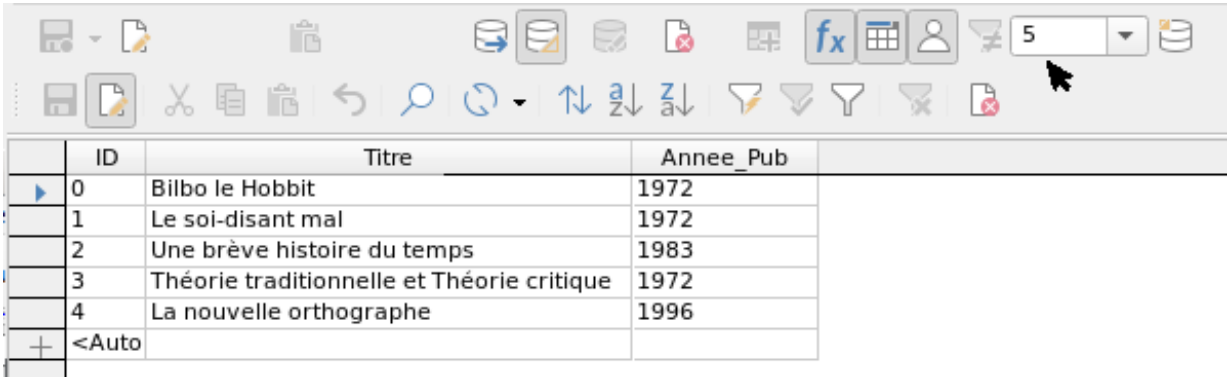
La possibilité de spécifier des enregistrements uniques était également présente dans les versions antérieures. Cependant, il était nécessaire de passer de la vue de conception à la vue SQL pour insérer le qualificatif DISTINCT. Cette propriété est donc rétrocompatible avec les versions précédentes de LO sans poser de problème.

Le paramètre Limite détermine le nombre d'enregistrements à afficher dans la requête. Le nombre d'enregistrements affichés est limité au nombre choisi (5, 10, 20, 50, Tout).



	ID	Titre	Annee_Pub
▶	0	Bilbo le Hobbit	1972
	1	Le soi-disant mal	1972
	2	Une brève histoire du temps	1983
	3	Théorie traditionnelle et Théorie critique	1972
	4	La nouvelle orthographe	1996
	5	I hear you knocking	1972
	6	Bases de données avec OpenOffice.org 3	2009
	7	Le livre Postfix	2008
	8	La vie mensongère des adultes	2009
	9	Émile et les Détectives	1929
	10	Les nuits de Reykjavik	2002
	11	Indignez vous !	2012
	13	Bon à tirer	1973
+	<Auto		

Tous les enregistrements de la table Medias sont affichés. La requête est modifiable car elle inclut la clé primaire.



	ID	Titre	Annee_Pub
▶	0	Bilbo le Hobbit	1972
	1	Le soi-disant mal	1972
	2	Une brève histoire du temps	1983
	3	Théorie traditionnelle et Théorie critique	1972
	4	La nouvelle orthographe	1996
+	<Auto		

Seuls les cinq premiers enregistrements sont affichés (ID 0-4). Un tri n'a pas été demandé, donc l'ordre de tri par défaut par clé primaire est utilisé. Malgré la limitation de la sortie, la requête peut

être modifiée davantage. Cela distingue l'entrée dans l'interface graphique de ce qui, dans les versions précédentes, n'était accessible qu'en utilisant SQL.

```
SELECT "ID", "Titre", "Annee_Pub" FROM "Medias" LIMIT 5
```

La requête d'origine est simplement complétée avec "LIMIT 5". La taille de la limite peut être celle dont vous avez besoin.

Avertissement

La définition de limites dans l'interface graphique n'est pas rétrocompatible. Dans les versions LO antérieures à 4.1, une limite ne pouvait être définie qu'en mode SQL direct. La limite nécessitait alors un tri (`ORDER BY…`) ou une condition (`WHERE…`).

Amélioration des requêtes à l'aide du mode SQL

Si, pendant la saisie graphique, vous utilisez **Affichage> Activer / Désactiver le mode Ébauche**, (ou l'icône correspondante) vous voyez la commande SQL correspondant à ce qui apparaissait précédemment en mode Ébauche. C'est le meilleur moyen pour les débutants d'apprendre le langage de requête standard pour les bases de données. Parfois, c'est aussi le seul moyen de saisir une requête dans la base de données lorsque l'interface graphique ne peut pas traduire vos besoins dans des commandes SQL nécessaires.

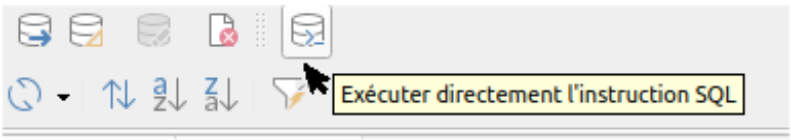
```
SELECT * FROM "Nom_Table"
```

Cela montrera tout ce qui se trouve dans la table Nom_Table. Le caractère"*" représente tous les champs de la table.

```
SELECT * FROM "Nom_Table" WHERE "Nom_Champ" = 'Serge'
```

Ici, il y a une restriction importante. Seuls les enregistrements pour lesquels le champ Nom_Champ contient le mot "Serge" – le terme exact, pas par exemple "Serge Reggiani".

Parfois, les requêtes dans Base ne peuvent pas être effectuées à l'aide de l'interface graphique, car certaines commandes peuvent ne pas être reconnues. Dans de tels cas, il est nécessaire de désactiver le mode Ébauche et d'utiliser la commande **Édition> Exécuter directement l'instruction SQL** pour accéder directement à la base de données. Cette méthode présente l'inconvénient que vous ne pouvez travailler avec la requête qu'en mode SQL.



L'utilisation directe des commandes SQL est également accessible à l'aide de l'interface utilisateur graphique, comme le montre la figure ci-dessus. Cliquez sur l'icône en surbrillance (Exécuter la commande SQL directement) pour désactiver l'icône Ébauche Off / On. Désormais, lorsque vous cliquez sur l'icône Exécuter, la requête exécute directement les commandes SQL.

Voici un exemple des nombreuses possibilités disponibles pour poser des questions à la base de données et spécifier le type de résultat requis :

```
SELECT [{LIMIT <décalage> <limit> | TOP <limit>}][ALL | DISTINCT]
{ <Formule de sélection> | "Nom_Table".* | * } [,…]
[INTO [CACHED | TEMP | TEXT] "Nouvelle_Table"]
FROM "Liste_Tables"
[WHERE Expression-SQL]
[GROUP BY Expression-SQL [,…]]
[HAVING Expression-SQL]
```

```
[{ UNION [ALL | DISTINCT] | {MINUS [DISTINCT] | EXCEPT [DISTINCT] } |
INTERSECT [DISTINCT] } Déclaration de requête]
[ORDER BY Expression-de-Tri [,…]]
[LIMIT <limite> [OFFSET <décalage>]] ;
```

[{LIMIT <décalage> <limite> | TOP <limite>}] :

Cela limite le nombre d'enregistrements à afficher. `LIMIT 10 20` commence au 11e enregistrement et affiche les 20 enregistrements suivants. `TOP 10` affiche toujours les 10 premiers enregistrements. C'est la même chose que `LIMIT 0 10`. LIMIT 10 0 omet les 10 premiers enregistrements et affiche tous les enregistrements à partir du 11e.

Vous pouvez faire la même chose en utilisant la dernière condition SELECT de la formule `[LIMIT <limite> [OFFSET <décalage>]]`. `LIMIT 10` affiche seulement 10 enregistrements. L'ajout de `OFFSET 20` fait que l'affichage commence au 21e enregistrement. Cette forme finale de limitation d'affichage nécessite soit une instruction de tri (`ORDER BY…`) soit une condition (`WHERE…`).

Toutes les valeurs entrées pour la limite doivent être entières. Il n'est pas possible de remplacer une entrée par une sous-requête afin que, par exemple, les cinq derniers enregistrements d'une série puissent être affichés à chaque fois.

[ALL | DISTINCT]

`SELECT ALL` est la valeur par défaut. Tous les enregistrements qui remplissent les conditions de recherche s'affichent. Exemple : `SELECT ALL "Prenom" FROM "Nom_Table"` donne tous les prénoms ; si "Serge" apparaît trois fois et "Valérie" quatre fois dans la table, ces prénoms sont affichés trois et quatre fois respectivement. `SELECT DISTINCT "Prenom" FROM "Nom_Table"` supprime les résultats de requête qui ont le même contenu. Dans ce cas, "Serge" et "Valérie" n'apparaissent qu'une seule fois. `DISTINCT` fait référence à l'ensemble de l'enregistrement auquel la requête accède. Ainsi, si, par exemple, le nom de famille est également demandé, les enregistrements pour "Serge Reggiani" et "Serge Gainsbourg" seront considérés comme distincts. Même si vous spécifiez la condition `DISTINCT`, les deux seront affichés.

<Select-Formulation>

```
{ Expression | COUNT(*) |
{ COUNT | MIN | MAX | SUM | AVG | SOME | EVERY | VAR_POP | VAR_SAMP
| STDDEV_POP | STDDEV_SAMP }
([ALL | DISTINCT]] Expression) } [[AS] "Nom_Affiché"]
```

Les noms de champ, les calculs, les totaux d'enregistrement sont toutes des entrées possibles.

Note

Les calculs au sein d'une base de données conduisent parfois à des résultats inattendus. Supposons qu'une base de données contienne les notes attribuées pour certains travaux en classe et que vous souhaitiez calculer la note moyenne.

Vous devez d'abord additionner toutes les notes. Supposons que la somme soit de 80. Maintenant, cela doit être divisé par le nombre d'élèves (disons 30). La requête donne 2.

Cela se produit car vous travaillez avec des champs de type INTEGER. Le calcul doit donc donner une valeur entière. Vous devez avoir au moins une valeur de type DECIMAL dans votre calcul. Vous pouvez y parvenir soit en utilisant une fonction de conversion HSQLDB ou (plus simplement) vous pouvez diviser par 30,0 plutôt que 30. Cela donnera un résultat avec une décimale (2.6). La division par 30,00 donne deux décimales (2.66). Veillez à utiliser des points décimaux de style anglais. L'utilisation de virgules dans les requêtes est réservée aux séparateurs de champs.

De plus, différentes fonctions sont disponibles pour le champ affiché. À l'exception de COUNT (*) (qui compte tous les enregistrements), aucune de ces fonctions n'accède aux champs NULL (vides).

```
COUNT | MIN | MAX | SUM | AVG | SOME | EVERY | VAR_POP | VAR_SAMP |
STDDEV_POP | STDDEV_SAMP
```

`COUNT("Nom")` compte toutes les entrées pour le champ Nom.
`MIN("Nom")` affiche le premier nom par ordre alphabétique. Le résultat de cette fonction est toujours formaté comme il se présente dans le champ. Le texte est affiché sous forme de texte, les entiers sous forme d'entiers, les décimales sous forme de décimales, etc.
`MAX("Name")` affiche le dernier nom par ordre alphabétique.
`SUM("Nombre")` ne peut ajouter que les valeurs des champs numériques. La fonction échoue pour les champs de date.

Conseil

Cette fonction échoue également sur les champs horaires. Ce qui suit peut être utile :

```
SELECT (SUM(HOUR("Duree"))*3600 + SUM(MINUTE("Duree")))*60 +
SUM(SECOND("Duree")) AS "secondes" FROM "Table"
```

La somme affichée est composée d'heures, de minutes et de secondes distinctes. Ensuite, elle est modifié pour donner le total global en une seule unité, dans ce cas les secondes. Ici, seuls les entiers sont ajoutés par la fonction somme. Maintenant, si vous utilisez :

```
SELECT ((SUM(HOUR("Duree"))*3600 + SUM(MINUTE("Duree")))*60 +
SUM(SECOND("Duree"))) / 3600.0000 AS "heures" FROM "Table"
```

vous obtiendrez une durée en heures avec les minutes et les secondes comme décimales. Avec une mise en forme appropriée, vous pouvez ramener cette valeur à une valeur de temps normale dans une requête ou un formulaire.

`AVG("Nombre")` montre la moyenne du contenu d'une colonne. Cette fonction est également limitée aux champs numériques.
`SOME("Nom_Champ"), EVERY("Nom_Champ")` : Les champs utilisés avec ces fonctions doivent avoir le type de champ Oui/Non [BOOLEAN] (contenant uniquement 0 ou 1). De plus, elles produisent un résumé du contenu du champ auquel ils sont appliqués.

`SOME` renvoie *TRUE* (ou 1) si au moins une entrée pour le champ est 1, et il renvoie *FALSE* (ou 0) uniquement si toutes les entrées sont 0. `EVERY` renvoie 1 uniquement si chaque entrée du champ est 1, et renvoie *FALSE* si au moins une entrée est 0.

Conseil

Le type de champ booléen est Oui/Non [BOOLEAN]. Cependant, ce champ ne contient que 0 ou 1. Dans les conditions de recherche de requête, utilisez TRUE, 1, FALSE ou 0. Pour la condition Oui, vous pouvez utiliser TRUE ou 1. Pour la condition Non, utilisez FALSE ou 0. Si vous essayez d'utiliser "Oui" ou "Non" à la place, vous obtenez un message d'erreur. Ensuite, vous devrez corriger votre erreur

Exemple :

```
SELECT "Classe", EVERY("Nageur")
FROM "Table1"
GROUP BY "Classe";
```

Le champ Classe contient le nom de la classe de natation. Nageur est un champ booléen décrivant si un élève sait nager ou non (1 ou 0). Le champ Etudiant contient les noms des étudiants. La Table1 contient ces champs : sa clé primaire, Classe, Nageur et Etudiant. Seuls les champs Classe et Nageur sont nécessaires pour cette requête.

Étant donné que la requête est regroupée par les entrées du champ Classe, EVERY renverra une valeur pour le champ Nageur pour chaque classe. Lorsque chaque personne dans une classe de natation sait nager, EVERY renvoie TRUE. Sinon EVERY renvoie FALSE car au moins un élève de la classe ne sait pas nager. Puisque la sortie pour le champ Nageur est une case à cocher, une coche indique TRUE, et aucune coche n'indique FALSE.

`VAR_POP | VAR_SAMP | STDDEV_POP | STDDEV_SAMP` sont des fonctions statistiques et n'affectent que les champs entiers et décimaux. Toutes ces fonctions renvoient 0, si les valeurs du groupe sont toutes égales.

Les fonctions statistiques ne permettent pas d'utiliser la limitation DISTINCT. Fondamentalement, ils calculent toutes les valeurs couvertes par la requête, tandis que DISTINCT exclut les enregistrements avec les mêmes valeurs de l'affichage.

`[AS] "Nom_Affiché"`: Les champs peuvent recevoir une désignation différente (alias) dans la requête.

"Nom_Table".* | * [,…]

Chaque champ à afficher est donné avec ses noms de champs, séparés par des virgules. Si des champs de plusieurs tables sont saisis dans la requête, une combinaison du nom du champ avec le nom de la table est nécessaire:: `"Nom_Table"."Nom_Champ"`.

Au lieu d'une liste détaillée de tous les champs d'une table, son contenu total peut être affiché. Pour cela, vous utilisez le symbole "*". Il est alors inutile d'utiliser le nom de la table, si les résultats ne s'appliquent qu'à une seule table. Cependant, si la requête inclut tous les champs d'une table et au moins un champ d'une deuxième table, utilisez: `"Nom_Table 1".*, "Nom_Table 2"."Nom_Champ"`.

[INTO [CACHED | TEMP | TEXT] "nouvelle_table"]

Le résultat de cette requête doit être écrit directement dans une nouvelle table qui est nommée ici. Les propriétés de champ de la nouvelle table sont définies à partir des définitions de champ contenues dans la requête. L'écriture dans une nouvelle table ne fonctionne pas à partir du mode SQL car cela ne gère que les résultats affichables. Au lieu de cela, vous devez utiliser **Outils > SQL**. La table résultante n'est initialement pas modifiable car elle n'a pas de clé primaire.

FROM <Liste_Tables>

```
"Nom_Table 1" [{CROSS | INNER | LEFT OUTER | RIGHT OUTER} JOIN
"Nom_Table 2" ON Expression] [,…]
```

Les tables qui doivent être recherchées conjointement sont généralement dans une liste séparée par des virgules. La relation entre les tables est alors définie en plus par le mot-clé `WHERE`.

Si les tables sont liées par une jointure `JOIN` plutôt que par une virgule, leur relation est définie par le terme commençant par `ON` qui apparaît directement après la deuxième table.

Un simple `JOIN` a pour effet que seuls les enregistrements pour lesquels les conditions des deux tables s'appliquent sont affichés..

Exemple :

```
SELECT "Table1"."Nom", "Table2"."Classe"
FROM "Table1", "Table2"
WHERE "Table1"."ID_Classe" = "Table2"."ID"
```

est equivalent à :

```
SELECT "Table1"."Nom", "Table2"."Classe"
FROM "Table1"
   JOIN "Table2"
ON "Table1"."ID_Classe" = "Table2"."ID"
```

Ici, les noms et les classes correspondantes sont affichés. Si un nom (étudiant) n'a aucune classe répertoriée pour lui, ce nom n'est pas inclus dans l'affichage. Si une classe n'a pas de nom (étudiant), elle n'est pas non plus affichée. L'ajout de `INNER` ne change rien à cela.

```
SELECT "Table1"."Nom", "Table2"."Classe"
FROM "Table1"
   LEFT JOIN "Table2"
ON "Table1"."ID_Classe" = "Table2"."ID"
```

Si `LEFT` est ajouté, tous les noms de Table1 sont affichés même s'ils n'ont pas de classe. Si, au contraire, `RIGHT` est ajouté, toutes les classes sont affichées même si elles ne contiennent aucun nom. L'ajout de `OUTER` n'a pas besoin d'être montré ici. (Right Outer Join est la même chose que Right Join ; Left Outer Join est la même chose que Left Join.)

```
SELECT "Table1"."Joueur1", "Table2"."Joueur2"
FROM "Table1" AS "Table1"
   CROSS JOIN "Table2" AS "Table1"
WHERE "Table1"."Joueur1" <> "Table2"."Joueur2"
```

Un `CROSS JOIN` nécessite que la table soit fournie avec un alias, mais l'ajout du terme `ON` n'est pas toujours nécessaire. Tous les enregistrements de la première table sont associés à tous les enregistrements de la seconde table. Ainsi, la requête ci-dessus donne tous les appariements possibles d'enregistrements de la première table avec ceux de la deuxième table à l'exception des appariements entre enregistrements pour le même joueur. Dans le cas d'un `CROSS JOIN`, la condition ne doit pas inclure de lien entre les tables spécifiées dans le terme `ON`. Au lieu de cela, la condition `WHERE` peur être entrée. Si les conditions sont formulées exactement comme dans le cas d'un simple JOIN, vous obtenez le même résultat :

```
SELECT "Table1"."Nom", "Table2"."Classe"
FROM "Table1"
   JOIN "Table2"
ON "Table1"."ID_Classe" = "Table2"."ID"
```

donne le même résultat que

```
SELECT "Table1"."Nom", "Table2"."Classe"
FROM "Table1" AS "Table1"
   CROSS JOIN "Table2" AS "Table2"
WHERE "Table1"."ID_Classe" = "Table2"."ID"
```

[WHERE Expression-SQL]

L'introduction standard des conditions pour demander un filtrage plus précis des données. Ici aussi, les relations entre les tables sont généralement définies si elles ne sont pas liées entre elles par JOIN.

[GROUP BY Expression-SQL [,…]]

Utilisez cette option lorsque vous souhaitez diviser les données de la requête en groupes avant d'appliquer les fonctions à chacun des groupes séparément. La division est basée sur les valeurs du champ ou des champs contenus dans l'expression `GROUP BY`.

Exemple :

```
SELECT "Nom", SUM("Entree"-"Sortie") AS "Balance"
FROM "Table1"
GROUP BY "Nom";
```

Les enregistrements ayant `le même` contenu dans le champ nom sont additionnés. Dans le résultat de la requête, la somme des entrées – sorties est donnée pour chaque personne. Ce champ doit être appelé Balance. Chaque ligne du résultat de la requête contient une valeur du champ Nom et le solde calculé pour cette valeur spécifique.

Conseil

Lorsque les champs sont traités à l'aide d'une fonction particulière (par exemple `COUNT`, `SUM`…), tous les champs qui ne sont pas traités avec une fonction mais qui doivent être affichés sont regroupés à l'aide de `GROUP BY`.

[HAVING Expression-SQL]

La formule `HAVING` ressemble étroitement à la formule `WHERE`. La différence est que la formule WHERE s'applique aux valeurs des champs sélectionnés dans la requête. La formule HAVING s'applique aux valeurs calculées sélectionnées. Plus précisément, la formule WHERE ne peut pas utiliser une fonction d'agrégation dans le cadre d'une condition de recherche ; la formule HAVING le fait.

La formule HAVING a deux objectifs comme le montrent les deux exemples ci-dessous. Dans le premier, la condition de recherche exige que le temps d'exécution minimum soit inférieur à 40 minutes. Dans le deuxième exemple, la condition de recherche exige que le solde d'un individu soit positif.

Les résultats de la requête pour la première liste les noms des personnes dont le temps d'exécution a été inférieur à 40 minutes au moins une fois et le temps d'exécution minimum. Les personnes dont les temps d'exécution sont tous supérieurs à 40 minutes ne sont pas répertoriées.

Les résultats de la requête pour la seconde liste les noms des personnes qui ont un résultat total supérieur à l'entrée et leur solde. Les personnes dont le solde est égal ou inférieur à 0 ne sont pas répertoriées.

```
Exemples :
SELECT "Nom", "Duree"
FROM "Table1"
GROUP BY "Nom", "Duree"
HAVING MIN("Duree") < '00:40:00';

SELECT "Nom", SUM("Entree"-"Sortie") AS "Balance"
FROM "Table1"
GROUP BY "Nom"
HAVING SUM("Entree"-"Sortie") > 0 ;
```

[Expression-SQL]

Les expressions SQL sont combinées selon le schéma suivant :

```
[NOT] condition [{ OR | AND } condition]
```

Exemple :

```
SELECT *
FROM "Nom_Table"
WHERE NOT "Date_Retour" IS NULL AND "ID_Lecteur" = 2 ;
```

Les enregistrements lus à partir de la table sont ceux pour lesquels une Date_Retour a été entré et le ID-Lecteur est 2. En pratique, cela signifie que tous les supports prêtés à une personne spécifique et retournés peuvent être récupérés. Les conditions ne sont liées qu'à AND. Le NOT se réfère uniquement à la première condition.

```
SELECT *
FROM "Nom_Table"
WHERE NOT ("Date_Retour" IS NULL AND "ID_Lecteur" = 2) ;
```

Les parenthèses autour de la condition, avec NOT à l'extérieur, montrent uniquement les enregistrements qui ne remplissent pas complètement la condition entre parenthèses. Cela couvrirait tous les enregistrements, à l'exception de ceux du ID_Lecteur numéro 2, qui n'ont pas encore été renvoyés.

[Expression-SQL] : conditions

```
{ valeur [|| valeur]}
```

Une valeur peut être une ou plusieurs valeurs concaténées par deux lignes verticales ||. Naturellement, cela s'applique également au contenu des champs.

```
SELECT "NomFamille" || ', ' || "Prenom" AS "Identite"
FROM "Nom_Table"
```

Le contenu des champs NomFamille et Prenom est affiché ensemble dans un champ appelé Identite. Notez qu'une virgule et un espace sont insérés entre le nom et le prénom.

```
| valeur { = | < | <= | > | >= | <> |!= } valeur
```

Ces signes correspondent aux opérateurs mathématiques bien connus:
`{Égal à | Inférieur à | Inférieur ou égal à | Supérieur à | Supérieur ou égal à | Différent de | Différent de}`

```
| valeur IS [NOT] NULL
```

Le champ correspondant n'a pas de contenu, car rien n'y a été écrit. Cela ne peut pas être déterminé sans ambiguïté dans l'interface graphique, car un champ de texte visuellement vide ne signifie pas que le champ est complètement sans contenu. Cependant, la configuration par défaut dans Base est que les champs vides de la base de données sont définis sur NULL.

```
| EXISTS(Query_result)
```

Exemple :

```
SELECT "Nom"
FROM "Table1"
WHERE EXISTS
   (SELECT "Prenom"
   FROM "Table2"
   WHERE "Table2"."Prenom" = "Table1"."Nom")
```

Les noms de Table1 sont affichés pour lesquels les prénoms sont donnés dans Table2.

```
| Valeur BETWEEN Valeur AND Valeur
```

`BETWEEN valeur1 AND valeur2` renvoie toutes les valeurs comprises entre valeur1 et valeur2 (incluse). Si les valeurs sont des lettres, un tri alphabétique est utilisé dans lequel les lettres minuscules ont la même valeur que les majuscules correspondantes.

```
SELECT "Nom"
FROM "Nom_Table"
WHERE "Nom" BETWEEN 'A' AND 'E';
```

Cette requête renvoie tous les noms commençant par A, B, C ou D (ainsi que par les lettres minuscules correspondantes). Comme E est défini comme limite supérieure, les noms commençant par E ne sont pas inclus. La lettre E seule apparaît juste *avant* les noms commençant par E.

```
| valeur [NOT] IN ({valeur [,…] | Query resultat })
```

Cela nécessite soit une liste de valeurs, soit une requête. La condition est remplie si la valeur est incluse dans la liste de valeurs ou dans le résultat de la requête.

```
| valeur [NOT] LIKE valeur [ESCAPE] valeur }
```

L'opérateur LIKE est nécessaire dans de nombreuses fonctions de recherche simples. La valeur est saisie selon le modèle suivant :

'%' représente n'importe quel nombre de caractères (y compris 0), "_" remplace exactement un caractère.

Pour rechercher "%" ou "_" lui-même, les caractères doivent immédiatement suivre un autre caractère "\" défini comme ESCAPE.

```
SELECT "Nom"
FROM "Nom_Table"
WHERE "Nom" LIKE '\_%' ESCAPE '\'
```

Cette requête affiche tous les noms commençant par un trait de soulignement. «\» est défini ici comme le caractère ESCAPE.

[Expression-SQL] : valeurs

```
[+ | -] { Expression [{ + | – | * | / | || } Expression]}
```

Les valeurs peuvent être précédées d'un signe. L'addition, la soustraction, la multiplication, la division et la concaténation d'expressions sont autorisées. Un exemple de concaténation :

```
SELECT "NomFamille"||', '||"Prenom"
FROM "Table"
```

De cette façon, les enregistrements sont affichés par la requête avec un champ contenant "Nom, Prénom". L'opérateur de concaténation peut être qualifié par les expressions suivantes.

```
| (Condition)
```

Pour cela consultez la section précédente [Expression-SQL] : conditions].

```
| Fonction ([Paramètre] [,…])
```

Voir la section Fonctions en annexe.

Les requêtes suivantes sont également appelées sous-requêtes (sous-sélections).

```
| Résultat de la requête qui donne exactement une réponse
```

Comme un enregistrement ne peut avoir qu'une seule valeur dans chaque champ, seule une requête qui produit précisément une valeur peut être affichée dans son intégralité.

```
| {ANY|ALL} (Résultat de la requête qui donne exactement une réponse
à partir d'une colonne entière)
```

Il y a souvent une condition qui compare une expression à tout un groupe de valeurs.

Combinée avec `ANY` cela signifie que l'expression doit se produire au moins une fois dans le groupe. Cela peut également être spécifié en utilisant la condition `IN`. `= ANY` donne le même résultat que `IN`.

Combinée avec `ALL` cela signifie que toutes les valeurs du groupe doivent correspondre à une seule expression.

[Expression – SQ] : Expression

```
{ 'Texte' | Nombre entier | Nombre à virgules flottante
| ["Table".]"Champ" | TRUE | FALSE | NULL }
```

Fondamentalement, les valeurs servent d'arguments pour diverses expressions, en fonction du format source. Pour rechercher le contenu des champs de texte, placez le contenu entre guillemets. Les nombres entiers sont écrits sans guillemets, tout comme les nombres à virgule flottante.

Les champs représentent les valeurs qui apparaissent dans ces champs de la table. Les champs sont généralement comparés entre eux ou avec des valeurs spécifiques. En SQL, les noms de champs doivent être placés entre guillemets, car ils pourraient ne pas être correctement reconnus autrement. En général, SQL suppose que le texte sans guillemets doubles est sans caractères spéciaux, c'est-à-dire un seul mot sans espaces et en majuscules. Si plusieurs tables sont contenues dans la requête, le nom de la table doit être donné en plus du nom du champ, séparé de ce dernier par un point.
TRUE et FALSE dérivent généralement des champs Oui/Non.

NULL signifie pas de contenu. Ce n'est pas la même chose que 0 mais correspond plutôt à "vide".

UNION [ALL | DISTINCT] Résultat_Requête

Cela lie les requêtes afin que le contenu de la deuxième requête soit écrit sous la première. Pour que cela fonctionne, tous les types des champs des deux requêtes doivent correspondre. Cette liaison de plusieurs requêtes fonctionne uniquement en mode commande SQL directe.

```
SELECT "Prenom"
FROM "Table1"
UNION DISTINCT
   SELECT "Prenom"
   FROM "Table2";
```

Cette requête renvoie tous les prénoms de table1 et table2 ; le terme supplémentaire DISTINCT signifie qu'aucun prénom en double ne sera affiché. DISTINCT est la valeur par défaut dans ce contexte. Par défaut, les prénoms sont triés par ordre alphabétique croissant. ALL provoque l'affichage de tous les prénoms dans Table1, suivis du prénom dans Table2. Dans ce cas, la valeur de tri par défaut est par la clé primaire.

L'utilisation de cette technique de requête permet de lister les valeurs d'un enregistrement directement les unes sous les autres dans une colonne. Supposons que vous ayez une table appelée Stock dans laquelle il y a des champs pour Prix_Vente, Prix_Rabais_1 et Prix_Rabais_2. À partir de là, vous souhaitez calculer un champ de combinaison qui listera ces prix directement les uns sous les autres.

```
SELECT
   "Prix_Vente"
FROM "Stock" WHERE "ID_Stock" = 1
   UNION
   SELECT
      "Prix_Rabais_1"
   FROM "Stock" WHERE "ID_Stock" = 1
   UNION
   SELECT
      "Prix_Rabais_2"
   FROM "Stock" WHERE "ID_Stock" = 1 ;
```

La clé primaire de la table Stock doit naturellement être définie à l'aide d'une macro, car le champ de combinaison aura une entrée correspondante.

MINUS [DISTINCT] | EXCEPT [DISTINCT] Résultat_Requête

```
SELECT "Prenom"
FROM "Table1"
EXCEPT
   SELECT "Prenom"
   FROM "Table2";
```

Affiche tous les prénoms de la table 1 à l'exception des prénoms contenus dans la table 2. `MINUS` et `EXCEPT` conduisent au même résultat. Le tri est alphabétique.

INTERSECT [DISTINCT] Query_result

```
SELECT "Prenom"
FROM "Table1"
INTERSECT
   SELECT "Prenom"
   FROM "Table2";
```

Cela affiche les prénoms qui apparaissent dans les deux tables. Le tri est à nouveau alphabétique. À l'heure actuelle, cela ne fonctionne qu'en mode de commande SQL directe.

[ORDER BY Expression-Tri[,…]]

L'expression peut être un nom de champ, un numéro de colonne (commençant par 1 à gauche), un alias (formulé avec AS par exemple) ou une expression de valeur composite (voir

[Expression SQL] : valeurs). L'ordre de tri est généralement croissant (ASC). Si vous souhaitez un tri décroissant, vous devez spécifier DESC explicitement.

```
SELECT "Prenom", "NomFamille" AS "Nom"
FROM "Table1"
ORDER BY "NomFamille";
```

est identique à

```
SELECT "Prenom", "NomFamille" AS "Nom"
FROM "Table1"
ORDER BY 2 ;
```

est identique à

```
SELECT "Prenom", "NomFamille" AS "Nom"
FROM "Table1"
ORDER BY "Nom";
```

Utilisation d'un alias dans une requête

Les requêtes peuvent reproduire des champs avec des noms modifiés.

```
SELECT "Prenom", "NomFamille" AS "Nom"
FROM "Table1"
```

Le champ *NomFamille* s'appelle *Nom* dans l'affichage.

Si une requête implique deux tables, chaque nom de champ doit être précédé du nom de la table :

```
SELECT "Table1"."Prenom", "Table1"."NomFamille" AS "Nom", "Table2"."Classe"
FROM "Table1", "Table2"
WHERE "Table1"."ID_Classe" = "Table2"."ID"
```

Le nom de la table peut également recevoir un alias, mais celui-ci ne sera pas reproduit en vue table. Si un tel alias est défini, tous les noms de table de la requête doivent être modifiés en conséquence :

```
SELECT "a"."Prenom", "a"."NomFamille" AS "Nom", "b"."Classe"
FROM "Table1" AS "a", "Table2" AS "b"
WHERE "a"."ID_Classe" = "b"."ID"
```

L'affectation d'un alias pour une table peut être effectuée plus brièvement sans utiliser le terme AS :

```
SELECT "a"."Prenom", "a"."NomFamille" "Nom", "b"."Classe"
FROM "Table1" "a", "Table2" "b"
WHERE "a"."ID_Class" = "b"."ID"
```

Cela rend cependant le code moins lisible. Pour cette raison, la forme abrégée ne doit être utilisée que dans des circonstances exceptionnelles.

Note

Malheureusement, le code de l'éditeur de requête dans l'interface utilisateur graphique a été récemment modifié afin que les désignations d'alias soient créées sans utiliser le préfixe AS. En effet, lorsque des bases de données externes sont utilisées, dont la méthode de spécification des alias ne peut être prédite, l'inclusion d'AS a conduit à des messages d'erreur.

Si le code de requête est ouvert pour modification, pas directement dans SQL mais à l'aide de l'interface graphique, les alias existants perdent leur préfixe AS. Si cela est important pour vous, vous devez toujours **modifier les requêtes dans la vue SQL**.

Un nom d'alias permet également d'utiliser une table avec le filtrage correspondant plus d'une fois dans une requête :

```
SELECT "PaiementCompte"."Balance", "Compte"."Date",
   "a"."Balance" AS "Actuel",
   "b"."Balance" AS "Prevu"
FROM "Compte"
   LEFT JOIN "Compte" AS "a"
   ON "Compte"."ID" = "a"."ID" AND "a"."Balance" >= 0
   LEFT JOIN "Compte" AS "b"
   ON "Compte"."ID" = "b"."ID" AND "b"."Balance" < 0
```

Requêtes pour la création de champs de list box

Les champs de zone de liste affichent une valeur qui ne correspond pas au contenu de la table sous-jacente. Ils sont utilisés pour afficher la valeur attribuée par un utilisateur à une clé étrangère plutôt que la clé elle-même. La valeur qui est finalement enregistrée dans le formulaire ne doit pas apparaître dans la première position du champ de zone de liste.

```
SELECT "Prenom", "ID"
FROM "Table1";
```

Cette requête afficherait tous les prénoms et les valeurs de clé primaire "ID" fournis par la table sous-jacente du formulaire. Bien sûr, ce n'est pas encore optimal. Les prénoms n'apparaissent pas triés et, dans le cas de prénoms identiques, il est impossible de déterminer de quelle personne il s'agit.

```
SELECT "Prenom"||' '||"Nom", "ID"
FROM "Table1"
ORDER BY "Prenom"||' '||"Nom";
```

Maintenant, le prénom et le nom apparaissent tous les deux, séparés par un espace. Les noms se distinguent et sont également triés. Mais le tri suit la logique habituelle de commencer par la première lettre de la chaîne, donc il trie par prénom et seulement ensuite par Nom. Un ordre de tri différent de celui dans lequel les champs sont affichés serait certainement déroutant.

```
SELECT "Nom"||', '||"Prenom", "ID"
FROM "Table1"
ORDER BY "Nom"||', '||"Prenom";
```

Cela conduit désormais à un tri qui correspond mieux à la coutume normale. Les membres de la famille apparaissent ensemble, les uns sous les autres ; cependant différentes familles avec le même nom de famille seraient imbriquées. Pour les distinguer, nous aurions besoin de les regrouper différemment dans le tableau.

Il y a un dernier problème : si deux personnes ont le même nom de famille et le même prénom, elles ne seront toujours pas distinguables. Une solution pourrait être d'utiliser un suffixe de nom. Mais imaginez à quoi cela ressemblerait si une civilité se lisait M. "Dupond II"!

```
SELECT "Nom"||', '||"Prenom"||' – ID:'||"ID", "ID"
FROM "Table1"
ORDER BY "Nom"||', '||"Prenom"||"ID";
```

Ici, tous les enregistrements sont apparents. Ce qui est réellement affiché est "Nom, Prenom – ID : valeur ID".

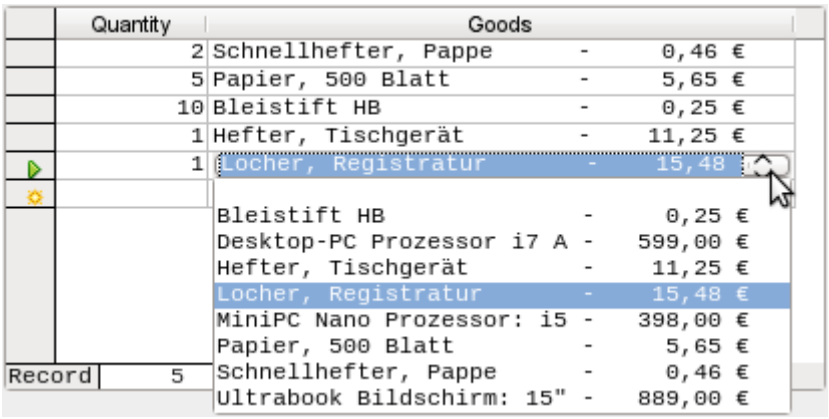
Dans le formulaire de prêt, il y a une zone de liste qui montre uniquement les médias qui n'ont pas encore été prêtés. Il est créé à l'aide de la formule SQL suivante :

```
SELECT "Titre" || ' – No. ' || "ID", "ID"
FROM "Medias"
WHERE "ID" NOT IN
   (SELECT "ID_Media"
   FROM "Prets"
```

```
   WHERE "Date_Retour" IS NULL)
ORDER BY "Titre" || ' – No. ' || "ID" ASC
```

Il est important que ce champ de liste soit toujours mis à jour si un média qu'il contient est prêté.

Le champ de liste suivant montre le contenu de plusieurs champs sous forme de tableau afin que les éléments qui appartiennent au même ensemble soient directement les uns sous les autres.



Pour qu'une telle représentation fonctionne, vous devez d'abord choisir une police de largeur fixe appropriée. Courier ou n'importe quelle mono-police telle que Liberation Mono peut être utilisée ici. Vous créez le formulaire tabulaire à l'aide du code SQL :

```
SELECT
   LEFT("Stock"||SPACE(25),25) || ' – ' ||
      RIGHT(SPACE(8)||"Prix",8) || ' €',
   "ID"
FROM "Stock"
ORDER BY ("Stock" || ' – ' || "Prix" || ' $') ASC
```

Le contenu du champ Stock a été rempli d'espaces afin que la chaîne entière ait une longueur minimale de 25 caractères. Ensuite, les 25 premières lettres y sont placées et le surplus est coupé.

Cela devient plus compliqué lorsque le contenu du champ de liste contient des caractères non imprimables comme des retours à la ligne. Ensuite, le code doit être personnalisé pour s'adapter :

```
SELECT
   LEFT(REPLACE("Stock", CHAR(10),' ')||SPACE(25), 25) || ' – ' ||…
```

Cela remplace une fin de ligne sous Linux par un espace. Sous Windows, vous devez en outre supprimer le retour chariot (CHAR (13)).

Le nombre d'espaces nécessaires peut également être déterminé sur une base par requête. Cela évite que la valeur de « Stock » ne soit accidentellement tronquée.

```
SELECT
   LEFT("Stock"||SPACE((SELECT MAX(LENGTH("Stock")) FROM "Stock")),
      (SELECT MAX(LENGTH("Stock")) FROM "Stock"))|| ' – ' ||
      RIGHT(' '||"Prix",8) || ' €',
   "ID"
FROM "Stock"
ORDER BY ("Stock" || ' – ' || "Prix" || ' $') ASC
```

Comme le prix doit être affiché justifié à droite, il est rempli d'espaces à gauche et placé au maximum huit caractères à partir de la droite. La représentation choisie fonctionnera pour tous les prix jusqu'à 99999,99 €.

Si vous souhaitez remplacer le point décimal par une virgule dans SQL, vous aurez besoin d'un code supplémentaire :

```
REPLACE(RIGHT(' '||"Prix",8),'.',',')
```

Requêtes comme base d'informations supplémentaires dans les formulaires

Si vous souhaitez qu'un formulaire affiche des informations supplémentaires qui autrement ne seraient pas visibles, il existe différentes possibilités de requête. Le plus simple est de récupérer ces informations avec des requêtes indépendantes et d'insérer les résultats dans le formulaire. L'inconvénient de cette méthode est que les modifications des enregistrements peuvent affecter le résultat de la requête, mais malheureusement ces modifications ne sont pas affichées automatiquement.

Voici un exemple dans le domaine du contrôle des stocks pour une simple vérification.

La table de vérification contient les totaux et les clés étrangères pour les articles en stock, ainsi qu'un numéro de reçu. L'acheteur a très peu d'informations s'il n'y a pas de résultat de requête supplémentaire imprimé sur le reçu. Après tout, les articles ne sont identifiés que par la lecture d'un code-barres. Sans requête, le formulaire affiche uniquement :

Total	*CodeBarre*
3	17
2	24

Ce qui est caché derrière les nombres ne peut pas être rendu visible à l'aide d'une zone de liste, car la clé étrangère est saisie directement à l'aide du code-barres. De la même manière, il est impossible d'utiliser une zone de liste à côté de l'article pour afficher au moins le prix unitaire.

Ici, une requête peut vous aider.

```
SELECT "Verif"."ID_recu", "Verif"."Total", "Stock"."Element",
"Stock"."Prix_Unitaire", "Verif"."Total"*"Stock"."Prix_Unitaire" AS
"Prix_Total"
FROM "Verif", "Stock"
WHERE "Stock"."ID" = "Verif"."ID_Element";
```

Maintenant, au moins après la saisie des informations, nous savons combien il faut payer pour 3 * Item'17 '. De plus, seules les informations relatives au ID_recu correspondant doivent être filtrées dans le formulaire. Ce qui manque encore, c'est ce que le client doit payer globalement.

```
SELECT "Verif"."ID_Recu", SUM("Verif"."Total"*"Stock"."Prix_Unitaire") AS
"Sum"
FROM "Verif", "Stock"
WHERE "Stock"."ID" = "Verif"."ID_Element"
GROUP BY "Verif"."ID_Recu";
```

Concevez le formulaire pour afficher un enregistrement de la requête à la fois. Étant donné que la requête est regroupée par ID_Recu, le formulaire affiche des informations sur un client à la fois.

Conseil

Si un formulaire doit afficher des valeurs de date qui dépendent d'une autre date (par exemple, une période de prêt pour un support peut être de 21 jours, quelle est la date de retour?), Vous ne pouvez pas utiliser les fonctions intégrées de HSQLDB. Il n'y a pas de fonction « DATEADD ».

La requête

```
SELECT "Date", DATEDIFF('dd','1899-12-30',"Date")+21 AS "Date_Retour" FROM
"Table"
```

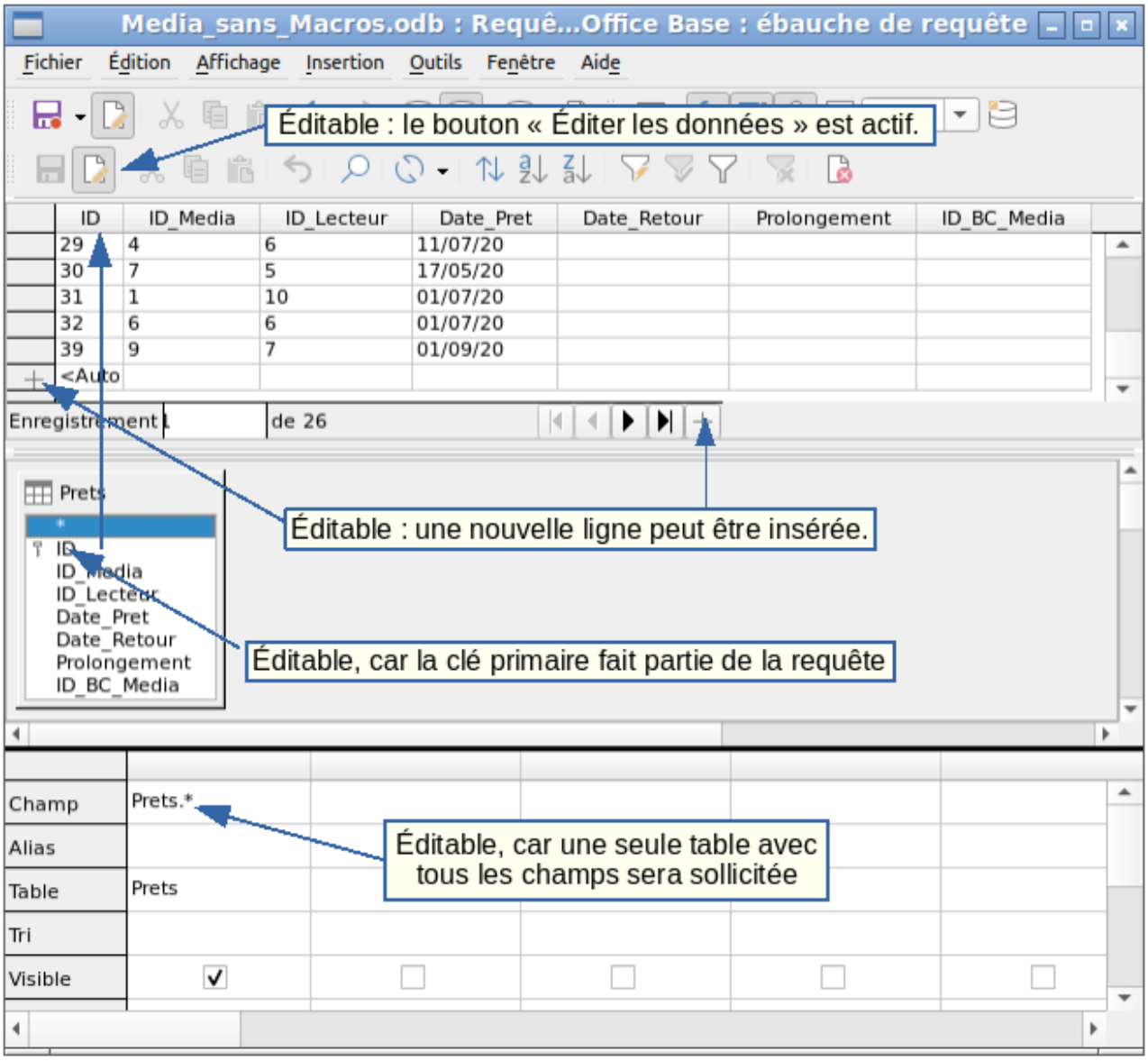
donnera la date cible correcte pour le retour dans un formulaire. Cette requête compte les jours à partir du 30.12.1899. Il s'agit de la date par défaut, que Calc utilise également comme valeur zéro.

Cependant, la valeur renvoyée n'est qu'un nombre et non une date qui peut être utilisée dans une autre requête.

Le nombre renvoyé ne convient pas pour une utilisation dans les requêtes car la mise en forme des requêtes n'est pas enregistrée. Vous devez plutôt créer une vue.

Possibilités de saisie de données dans les requêtes

Pour créer des saisies dans une requête, la clé primaire de la table sous-jacente à la requête doit être présente. Cela s'applique également à une requête qui relie plusieurs tables entre elles.



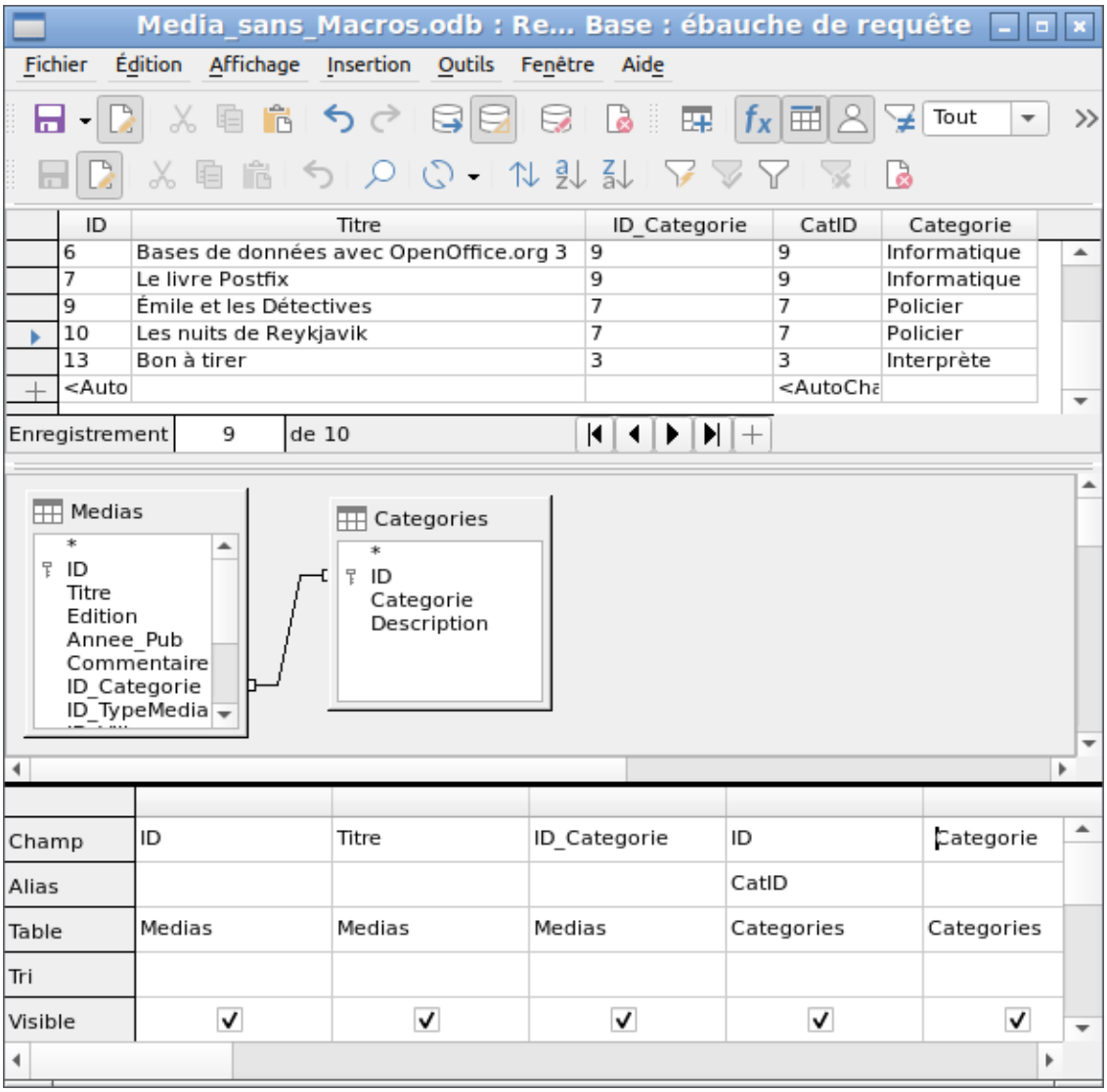
Lors du prêt de supports, cela n'a aucun sens d'afficher pour un lecteur des éléments qui ont déjà été retournés il y a quelque temps.

```
SELECT "ID", "ID_Lecteur", "ID_Media", "Date_Pret"
FROM "Prets"
WHERE "Date_Retour" IS NULL ;
```

De cette manière, un formulaire peut afficher dans un champ de contrôle de table tout ce qu'un lecteur particulier a emprunté au fil du temps. Ici aussi, la requête doit filtrer en utilisant la structure

de formulaire appropriée (lecteur dans le formulaire principal, requête dans le sous-formulaire), de sorte que seuls les médias réellement prêtés soient affichés (et non les prêts anciens revenus). La requête convient à la saisie de données car la clé primaire est incluse dans la requête.

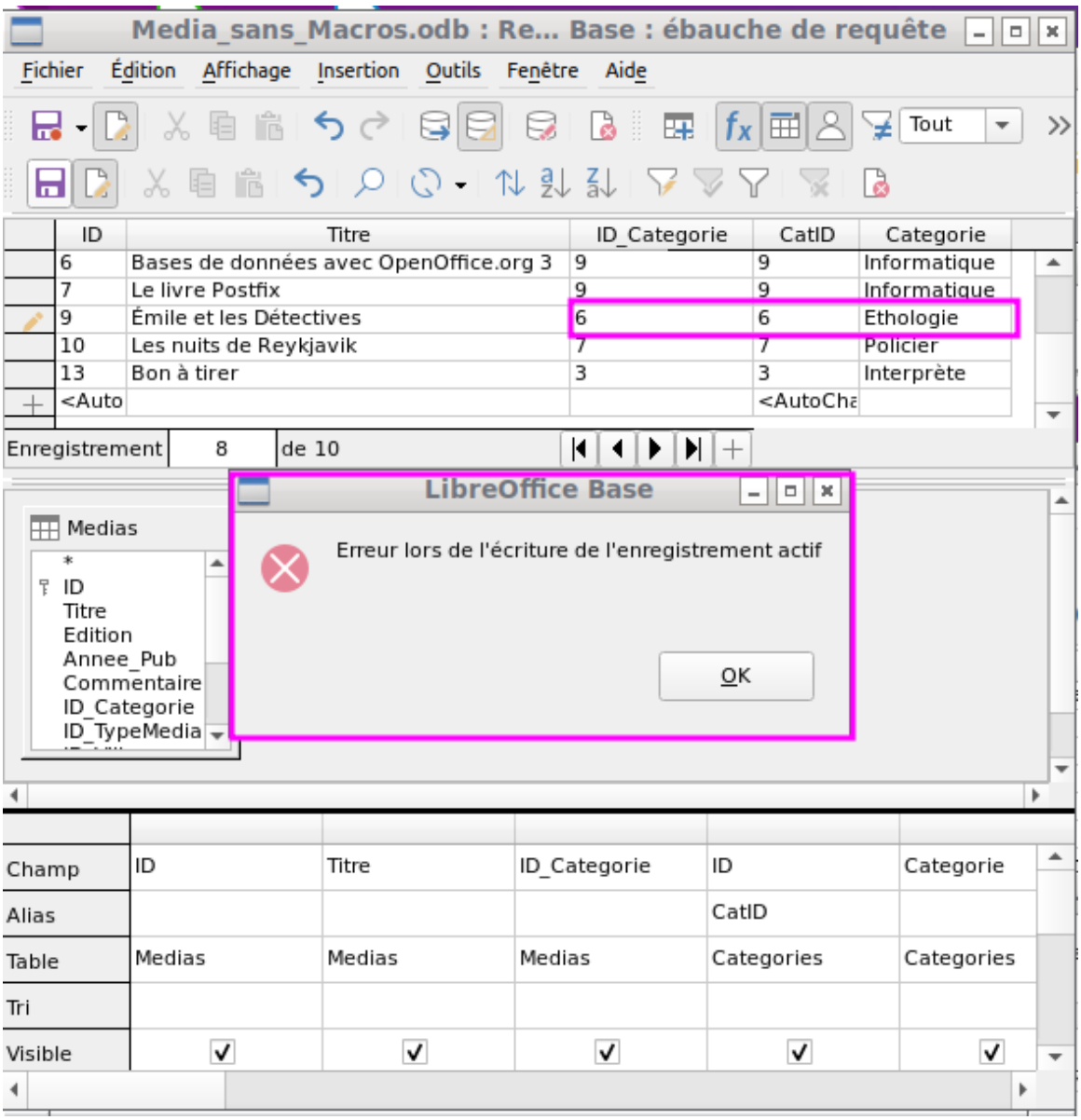
La requête cesse d'être modifiable si elle se compose de plus d'une table et que les tables sont accessibles via un alias. Cela ne fait aucune différence dans ce cas que la clé primaire soit présente ou non dans la requête.




```
SELECT "Medias"."ID", "Medias"."Titre", "Medias"."ID_Categorie",
"Categorie"."ID" AS "CatID", "Categorie"."Categorie"
FROM "Medias", "Categorie"
WHERE "Medias"."ID_Categorie" = "Categorie"."ID";
```

Cette requête est modifiable, car les deux clés primaires sont incluses et sont accessibles dans les tables sans utiliser d'alias. Une requête qui lie plusieurs tables ensemble nécessite également que toutes les clés primaires soient présentes.

Dans une requête qui implique plusieurs tables, il n'est pas possible de modifier un champ de clé étrangère dans une table qui fait référence à un enregistrement dans une autre table. Dans l'enregistrement ci-dessous, une tentative a été faite pour changer la catégorie du titre « Emilie et les détectives ». Le champ ID_Categorie est passé de 7 à 6. Le changement a semblé avoir lieu et la nouvelle catégorie est apparue dans l'enregistrement. Cependant, il s'est avéré impossible de l'enregistrer.

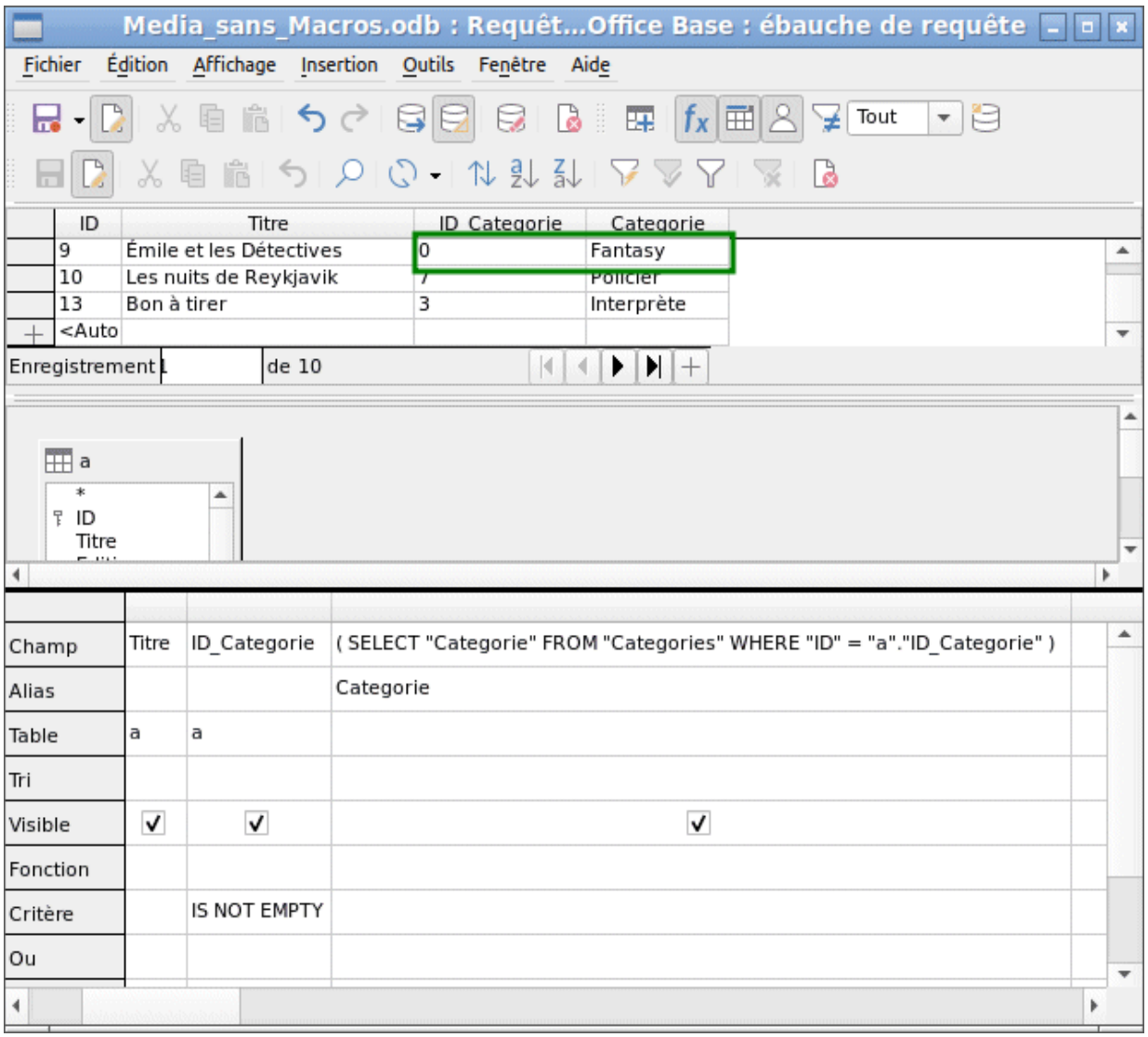


Cependant, il est possible d'éditer le contenu de l'enregistrement de la catégorie correspondante, par exemple pour remplacer « Policier » par « Suspense ». Le nom de la catégorie sera alors modifié dans la table Categories, donc pour tous les enregistrements liés à cette catégorie.

```
SELECT "m"."ID", "m"."Titre", "Categories"."Categorie", "Categorie"."ID" AS
"CatID"
FROM "Medias" AS "m", "Categories"
WHERE "m"."ID_Categorie" = "Categories"."ID";
```

Dans cette requête, la table Medias est accessible à l'aide d'un alias. La requête ne peut pas être modifiée.

Dans l'exemple ci-dessus, ce problème est facilement évité. Si, toutefois, une sous-requête corrélée (voir page 40)est utilisée, vous devez utiliser un alias de table. Une requête n'est modifiable dans ce cas que si elle ne contient qu'une seule table dans la requête principale.



En mode Ébauche, un seul tableau apparaît. La table Medias reçoit un alias afin que le contenu du champ ID_ Categorie soit accessible à l'aide d'une sous-requête corrélée.

```
SELECT "ID", "Titre", "ID_Categorie", (SELECT "Categorie" FROM "Categories"
WHERE "ID" = "a"."ID_Categorie") AS "Categorie" FROM "Medias" AS "a" WHERE
"ID_Categorie" IS NOT NULL
```

Dans une requête comme celle-ci, il est désormais possible de changer le champ de clé étrangère ID_Categorie en une autre catégorie. Dans l'exemple ci-dessus, le champ ID_Categorie passe de 7 à 0. Le titre « Emilie et les détectives » est ainsi affecté à la catégorie "Fantasy".

Cependant, il n'est plus possible de modifier une valeur dans le champ qui a reçu son contenu via une sous-requête corrélée. Une tentative de changement de catégorie de "Fantasy" à "Suspense" est montrée. Cette modification n'est pas enregistrée et ne peut pas non plus être enregistrée, puisque dans le tableau que la vue de conception affiche, le champ Categorie n'est pas présent, mais seulement un alias.

Utilisation des paramètres dans les requêtes

Si vous utilisez souvent la même requête de base mais avec des valeurs différentes à chaque fois, des requêtes avec des paramètres peuvent être utilisées. En principe, les requêtes avec paramètres fonctionnent comme les requêtes pour un sous-formulaire :

```
SELECT "ID", "ID_Lecteur", "ID_Media", "Date_Pret"
FROM "Prets"
WHERE "Date_Retour" IS NULL AND "ID_Lecteur"=2 ;
```

Cette requête n'affiche que les médias prêtés au lecteur avec le numéro 2.

```
SELECT "ID", "ID_Lecteur", "ID_Media", "Date_Pret"
FROM "Prets"
WHERE "Date_Retour" IS NULL AND "ID_Lecteur"= :NumeroLecteur ;
```

Désormais, lorsque vous exécutez la requête, un champ de saisie apparaît. Il vous invite à entrer un numéro de lecteur. Quelle que soit la valeur que vous entrez ici, les médias actuellement prêtés à ce lecteur seront affichés.

```
SELECT
   "Pret"."ID",
   "Lecteurs"."Nom"||', '||"Lecteurs"."Prenom",
   "Prets"."ID_Media",
   "Prets"."Date_Pret"
FROM "Prets", "Lecteurs"
WHERE "Prets"."Date_Retour" IS NULL
   AND "Lecteurs"."ID" = "Prets"."ID_Lecteur"
   AND "Lecteurs"."Nom" LIKE '%' ||: NomLecteur || '%'
ORDER BY "Lecteurs"."Nom"||', '||"Lecteurs"."Prenom" ASC ;
```

Cette requête est clairement plus conviviale que la précédente. Il n'est plus nécessaire de connaître le numéro du lecteur. Tout ce dont vous avez besoin est de saisir une partie du nom et tous les médias prêtés aux lecteurs correspondants sont affichés. La casse est importante.

Si vous remplacez

```
"Lecteurs"."Nom" LIKE '%' ||: NomLecteur || '%'
```

par

```
LOWER("Lecteurs"."Nom") LIKE '%' || LOWER(: NomLecteur) || '%'
```

Peu importe que le nom soit saisi en majuscules ou en minuscules.

Si le paramètre dans la requête ci-dessus est laissé vide, toutes les versions de LibreOffice jusqu'à 4.4 afficheront tous les lecteurs, car ce n'est que dans la version 4.4 qu'un champ de paramètre vide se lit comme NULL plutôt que comme une chaîne vide. Si vous ne voulez pas ce comportement, vous devez utiliser une astuce pour l'empêcher :

```
LOWER ("Lecteurs"."Nom") LIKE '%' || IFNULL(NULLIF (LOWER (: NomLecteur), ''),
'§§') || '%'
```

Le champ de paramètre vide renvoie une chaîne vide et non une valeur NULL à la requête. Par conséquent, le champ de paramètre vide doit être affecté à la propriété NULL à l'aide de NULLIF. Ensuite, puisque l'entrée de paramètre donne maintenant NULL, elle peut être réinitialisée à une valeur qui ne se produit normalement dans aucun enregistrement. Dans l'exemple ci-dessus, il s'agit de «§§». Cette valeur ne sera bien sûr pas trouvée dans la recherche.

Depuis la version 4.4, des adaptations à cette technique de requête sont nécessaires :

```
LOWER ("Lecteurs"."Nom") LIKE '%' || LOWER (: NomLecteur) || '%'
```

doit, en l'absence d'entrée, obligatoirement donner pour la combinaison :

```
'%' || LOWER (: NomLecteur) || '%' la valeur NULL.
```

Pour éviter cela, ajoutez une condition supplémentaire, à savoir que pour un champ vide, toutes les valeurs sont réellement affichées :

```
(LOWER ("Lecteurs"."Nom") LIKE '%' || LOWER (: NomLecteur) || '%' OR :
NomLecteur IS NULL)
```

Le tout doit être mis entre parenthèses. Ensuite, soit un nom est recherché, soit, si le champ est vide (NULL de LibreOffice 4.4), la deuxième condition s'appliquera.

Lors de l'utilisation de formulaires, le paramètre peut être passé du formulaire principal à un sous-formulaire. Cependant, il arrive parfois que les requêtes utilisant des paramètres dans les sous-formulaires ne soient pas mises à jour, si les données sont modifiées ou nouvellement saisies.

Souvent, il serait bien de modifier le contenu des zones de liste en utilisant les paramètres du formulaire principal. Ainsi, par exemple, nous pourrions empêcher le prêt de supports de bibliothèque à des personnes actuellement interdites d'emprunter des supports. Malheureusement, il n'est pas possible de contrôler les paramètres de la zone de liste de cette manière personnalisée à l'aide de paramètres.

Sous-requêtes

Les sous-requêtes intégrées aux champs ne renvoient toujours qu'un seul enregistrement. Le champ ne renvoie également qu'une seule valeur.

```
SELECT "ID", "Revenu", "Depense",
   (SELECT SUM("Revenu") – SUM("Depense")
      FROM "Caisse") AS "Balance"
FROM "Caisse";
```

Cette requête permet la saisie de données (clé primaire incluse). La sous-requête renvoie précisément une valeur, à savoir le solde total. Cela permet de lire le solde de la caisse après chaque entrée. Ce n'est toujours pas comparable avec le formulaire de paiement du supermarché décrit dans "Requêtes comme base d'informations supplémentaires dans les formulaires" en page 33. Naturellement, il manque les calculs individuels de Total * Prix_Unitaire, mais aussi la présence du numéro de reçu. Seule la somme totale est donnée. Au moins le numéro de reçu peut être inclus à l'aide d'un paramètre de requête :

```
SELECT "ID", "Revenu", "Depense",
   (SELECT SUM("Revenu") – SUM("Depense")
      FROM "Caisse")
      WHERE "ID_Recu" = : Numero_Recu) AS "Balance"
FROM "Caisse" WHERE "ID_Recu" =: Numero_Recu ;
```

Dans une requête avec paramètres, le paramètre doit être le même dans les deux instructions de requête s'il doit être reconnu comme paramètre.

Pour les sous-formulaires, ces paramètres peuvent être inclus. Le sous-formulaire reçoit alors, au lieu d'un nom de champ, le nom de paramètre correspondant. Ce lien ne peut être saisi que dans les propriétés du sous-formulaire, et non lors de l'utilisation de l'assistant.

Note

Les sous-formulaires basés sur des requêtes ne sont pas automatiquement mis à jour en fonction de leurs paramètres. Il est plus approprié de transmettre le paramètre directement depuis le formulaire principal.

Sous-requêtes corrélées

En utilisant une requête encore plus raffinée, une requête modifiable vous permet même de porter le solde courant de la caisse :

```
SELECT "ID", "Revenu", "Depense",
(SELECT SUM("Revenu") - SUM("Depense")
   FROM "Caisse"
   WHERE "ID" <= "a"."ID") AS "Balance"
FROM "Caisse" AS "a"
ORDER BY "ID" ASC
```

La table de paiement est le même que la table « a ». "a" ne donne cependant que la relation avec les valeurs actuelles de cet enregistrement. De cette manière, la valeur actuelle de l'ID de la requête externe peut être évaluée dans la sous-requête. Ainsi, en fonction de l'ID, le solde précédent au moment correspondant est déterminé, si vous partez du fait que l'ID, qui est une valeur automatique, s'incrémente de lui-même.

Note

Si la sous-requête doit être filtrée pour le contenu à l'aide de la fonction de filtre de l'éditeur de requête, cela ne fonctionne actuellement que si vous utilisez des parenthèse doubles au lieu de simples au début et à la fin de la sous-requête : `((SELECT…)) AS "Saldo"`

Requêtes comme tables source pour les requêtes

Une requête est nécessaire pour verrouiller tous les lecteurs qui ont reçu un troisième avis de retard pour un support.

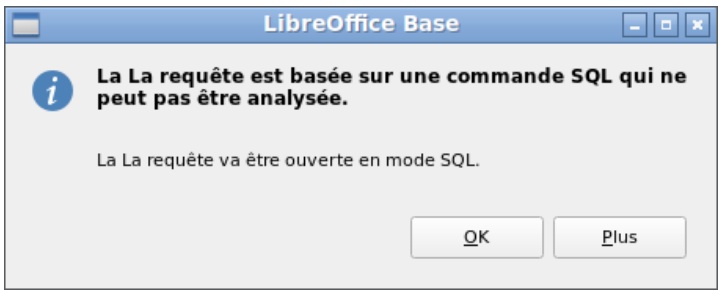
```
SELECT "Prets"."ID_Lecteur", '3e retard – le lecteur est sur la liste noire'
AS "Verrou"
FROM
   (SELECT COUNT("Date") AS "Compte_Total", "ID_Pret"
      FROM "Rappels" GROUP BY "ID_Pret") AS "a",
   "Prets"
WHERE "a"."ID_Pret" = "Prets"."ID" AND "a"."Compte_Total" > 2
```

Examinons d'abord la requête interne à laquelle se rapporte la requête externe. Dans cette requête, le nombre d'entrées de date regroupées par la clé étrangère ID_Pret est déterminé. Cela ne doit pas dépendre de l'ID_Lecteur, car cela entraînerait non seulement trois avis en retard pour un seul support, mais aussi trois médias avec un avis en retard chacun à compter. La requête interne reçoit un alias afin qu'elle puisse être liée à l'ID_Lecteur à partir de la requête externe.

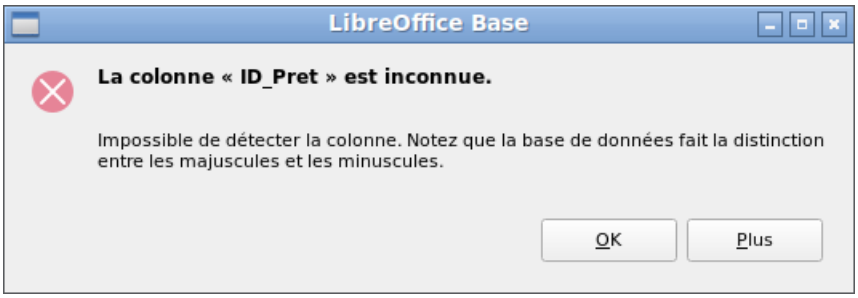
La requête externe concerne dans ce cas uniquement la formule conditionnelle de la requête interne. Il affiche uniquement un ID_Lecteur et le texte du champ Verrou lorsque le "Pret". "ID" et "a". "ID_Pret" sont égaux et "a". "Compte_Total"> 2.

En principe, tous les champs de la requête interne sont disponibles pour la requête externe. Ainsi, par exemple, la somme "a". "Compte_Total" peut être fusionnée dans la requête externe pour donner le total réel des amendes.

Cependant, il peut arriver, dans la boîte de dialogue Ébauche de requête, que le mode Ébauche ne fonctionne plus après une telle construction. Si vous essayez à nouveau d'ouvrir la requête pour la modifier, vous obtenez ce qui suit warning :



Si vous ouvrez ensuite la requête pour la modifier en mode SQL et que vous essayez de passer de là en mode Ébauche, vous obtenez le message d'erreur :



Le mode d'affichage de conception ne peut pas trouver le champ contenu dans la requête interne "ID_Pret", qui régit la relation entre les requêtes internes et externes. Lorsque la requête est exécutée en mode SQL, le contenu correspondant de la sous-requête est reproduit sans erreur. Par conséquent, vous n'êtes pas obligé d'utiliser le mode SQL direct dans ce cas.

La requête externe a utilisé les résultats de la requête interne pour produire les résultats finaux. Voici une liste des valeurs "ID_Pret" qui doivent être verrouillées et pourquoi. Si vous souhaitez limiter davantage les résultats finaux, utilisez les fonctions de tri et de filtrage de l'interface utilisateur graphique.

Les captures d'écran suivantes montrent comment les résultats d'une requête peuvent être obtenus de différentes manières avec des sous-requêtes. Ici, une requête vers une base de données de stock tente de déterminer ce que le client doit payer à la caisse. Les prix individuels sont multipliés par le nombre d'articles achetés donnant un sous-total. Ensuite, la somme de ces sous-totaux doit être déterminée. Tout cela doit être modifiable pour que la requête puisse être utilisée comme base d'un formulaire.

	ID	Quantite	ID_Article	IDArticle	Prix	SousTotal
	0	1	6	6	398,00 €	398,00 €
▶	1	10	2	2	0,46 €	4,60 €
	2	1	5	5	899,00 €	899,00 €
	3	1	6	6	398,00 €	398,00 €
	4	1	7	7	505,00 €	505,00 €
*	<Auto			<AutoChamp		

Enregistrement 2 de 5

```
SELECT
   "Ventes"."ID",
   "Ventes"."Quantite",
   "Ventes"."ID_Article",
    "Articles"."ID" AS "IDArticle",
   "Articles"."Prix",
   "Quantite" * "Prix" AS"SousTotal"
FROM "Ventes",
      "Articles"
WHERE "Ventes"."ID_Article" = "Articles"."ID"
```

Figure 6: Requête utilisant deux tables. Pour la rendre modifiable, les deux clés primaires doivent être incluses.

Note

En raison du bogue 61871, Base ne met pas à jour automatiquement le résultat partiel.

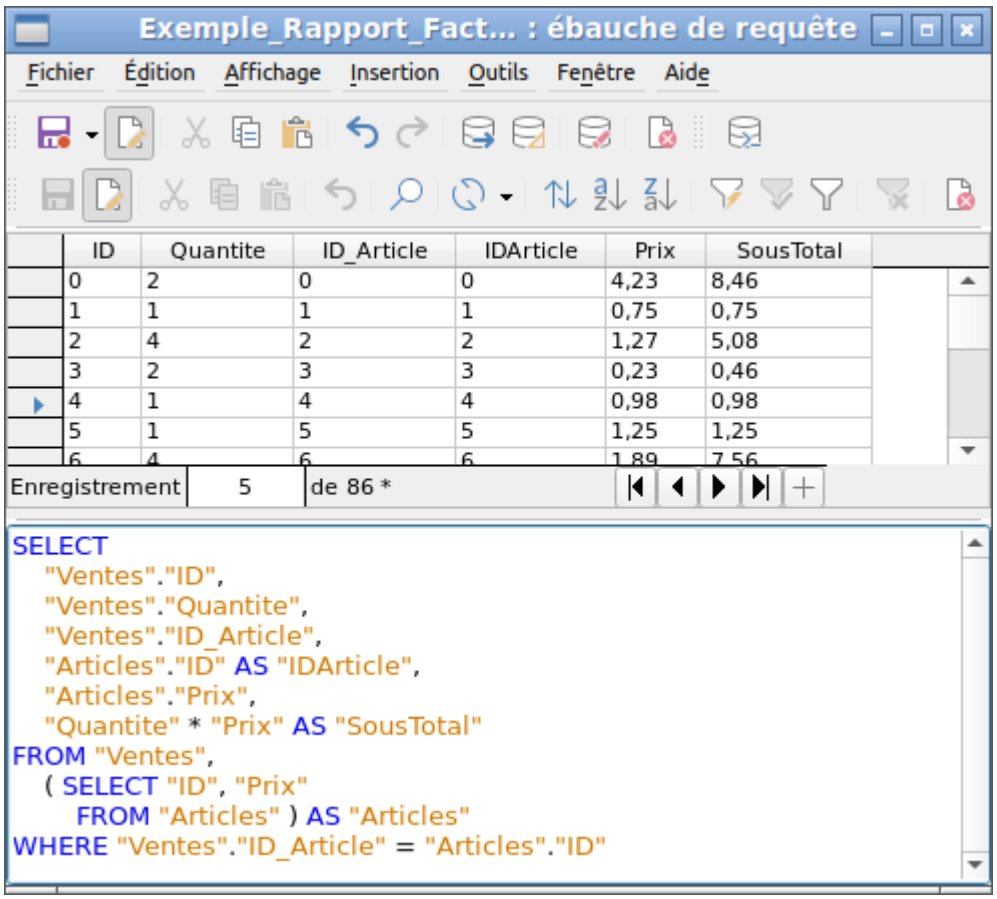


Figure 7: La table Articles est déplacée dans une sous-requête, qui est créée dans la zone de table (après le terme « FROM ») et reçoit un alias. Désormais, la clé primaire de la table Articles n'est plus strictement nécessaire pour rendre la requête modifiable.

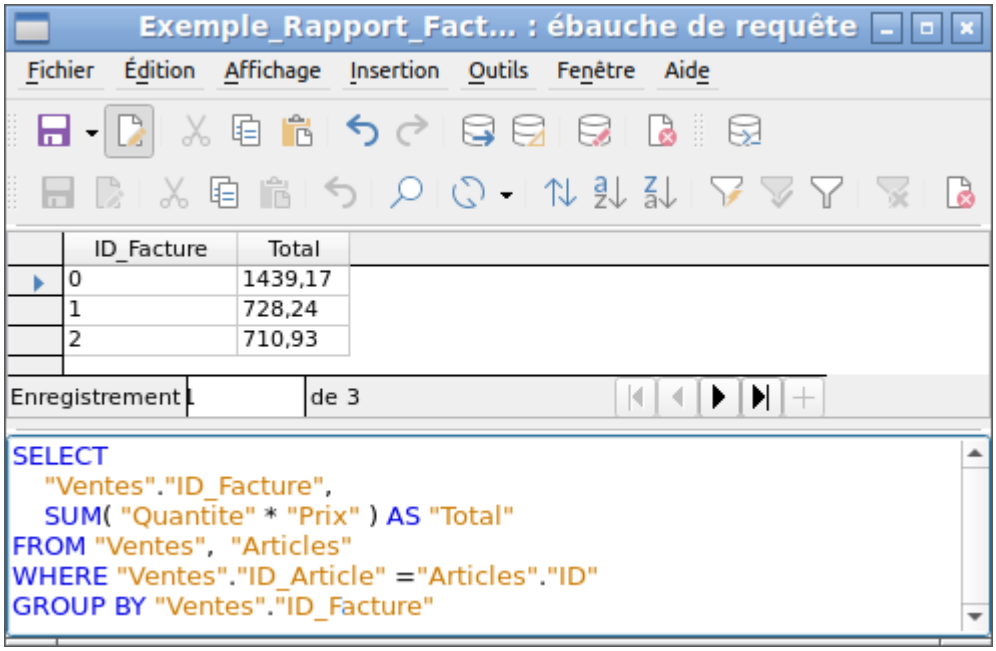


Figure 8: La somme calculée doit maintenant apparaître dans la requête. Déjà, la requête simple pour la somme de calcul n'est pas modifiable, elle est donc regroupée et additionnée ici.

ID	Quantite	ID_Article	Prix	SousTotal	Total
51	1	27	78,89	78,89	1439,17
52	2	28	1,25	2,5	1439,17
53	1	29	0,85	0,85	1439,17
54	2	30	0,65	1,3	1439,17
55	2	0	4,23	8,46	728,24
56	1	1	0,75	0,75	728,24
57	4	2	1,27	5,08	728,24
58	2	3	0,23	0,46	728,24
59	1	4	0,98	0,98	728,24

Enregistrement 1 de 110

```
SELECT
   "Ventes"."ID",
   "Ventes"."Quantite",
   "Ventes"."ID_Article",
   "Articles"."Prix",
   "Quantite" * "Prix" AS "SousTotal",
   "TotalFacture"."Total"
FROM "Ventes",
      ( SELECT
         "ID",
         "Prix"
      FROM "Articles" )
   AS "Articles",
      ( SELECT
         "Ventes"."ID_Facture",
         SUM( "Quantite" * "Prix" ) AS "Total"
      FROM "Ventes",
         "Articles"
      WHERE "Ventes"."ID_Article" = "Articles"."ID"
      GROUP BY "Ventes"."ID_Facture" )
   AS "TotalFacture"
WHERE "Ventes"."ID_Article" = "Articles"."ID"
   AND "Ventes"."ID_Facture" = "TotalFacture"."ID_Facture"
ORDER BY "Ventes"."ID_Facture", "Ventes"."ID"
```

Figure 9: Avec la deuxième sous-requête, l'impossibilité apparente devient possible. La requête précédente est insérée en tant que sous-requête dans la définition de table de la requête principale (après « FROM »). Par conséquent, l'ensemble de la requête reste modifiable. Dans ce cas, les entrées ne sont possibles que dans les colonnes « Quantite » et « ID_Article ». Ceci est ensuite précisé dans le formulaire de requête.

Récapituler les données avec des requêtes

Lorsque des données sont recherchées sur toute une base de données, l'utilisation de fonctions de formulaire simples pose souvent des problèmes. Un formulaire se réfère après tout à une seule table et la fonction de recherche se déplace uniquement dans les enregistrements sous-jacents de ce formulaire.

Obtenir toutes les données est plus simple lorsque vous utilisez des requêtes, qui peuvent fournir une image de tous les enregistrements. La section sur “Définition de la relation dans la requête” suggère une telle construction de requête. Ceci est construit pour l'exemple de base de données comme suit :

```
SELECT "Medias"."Titre", "SousTitre"."SousTitre", "Auteurs"."Nom"
FROM "Medias"
   LEFT JOIN "SousTitre"
```

```
   ON "Medias"."ID" = "SousTitre"."ID_Media"
LEFT JOIN "relation_Media_Auteur"
   ON "Medias"."ID" = "relation_Media_Auteur"."ID_Media"
LEFT JOIN "Auteurs"
   ON "relation_Media_Auteur"."ID_Auteur" = "Auteurs"."ID"
```

Ici, tous les titres, sous-titres et auteurs sont affichés ensemble.

La table Medias contient un total de 13 titres. Pour deux de ces titres, il y a un total de 10 sous-titres. Sans une jointure à gauche (LEFT **JOIN)**, les deux tables affichées ensemble ne donnent que 10 enregistrements. Pour chaque sous-titre, le titre correspondant est recherché, et c'est la fin de la requête. Les titres sans sous-titres ne sont pas affichés.

Maintenant, pour afficher tous les médias, y compris ceux sans sous-titre : le média est sur le côté gauche de l'affectation, sous-titre sur le côté droit. Une jointure à gauche affichera tous les titres des médias, mais seulement un sous-titre pour ceux qui ont un titre. Le média devient la table décisive pour déterminer quels enregistrements doivent être affichés. Cela avait déjà été prévu lors de la construction de la table (voir Chapitre 3, Tables). Comme des sous-titres existent pour deux des treize titres, la requête affiche désormais 13+ 9 = 22 enregistrements.

Note

La liaison normale des tables, une fois que toutes les tables ont été répertoriées, suit le mot-clé WHERE.

```
S'il existe une jointure à gauche ou une jointure à droite, l'affectation est définie
directement après les deux noms de table à l'aide de ON. La séquence est donc
toujours
Table1 LEFT JOIN Table2 ON Table1.Champ1 = Table2.Champ1 LEFT JOIN Table3 ON
Table2.Champ1 = Table3.Champ1…
```

Deux titres de la table Medias n'ont pas encore d'entrée d'auteur ou de sous-titre. En même temps, un titre a un total de trois auteurs. Si la table Auteurs est liée sans une jointure à gauche, les médias sans auteur ne seront pas affichés. Mais comme un média a trois auteurs au lieu d'un, le nombre total d'enregistrements affichés sera de 23.

Ce n'est qu'en utilisant LEFT JOIN que la requête sera invitée à utiliser la table Medias pour déterminer les enregistrements à afficher. Maintenant, les enregistrements sans sous-titre ni auteur réapparaissent, ce qui donne un total de 25 enregistrements.

L'utilisation de jointures appropriées augmente généralement la quantité de données affichées. Mais cet ensemble de données élargi peut être facilement examinés, car les auteurs et les sous-titres sont affichés en plus des titres. Dans l'exemple de base de données, toutes les tables dépendantes du support sont accessibles.

Accès plus rapide aux requêtes à l'aide des vues de table

Les vues en SQL sont plus rapides que les requêtes, en particulier pour les bases de données externes, car elles sont directement ancrées dans la base de données et le serveur ne renvoie que les résultats. En revanche, les requêtes sont d'abord envoyées au serveur et y sont traitées.

Si une nouvelle requête est liée à une autre requête, la vue SQL de Base fait que l'autre requête ressemble à une table. Si vous créez une vue à partir de celle-ci, vous pouvez voir que vous travaillez réellement avec une sous-requête (Select utilisé dans un autre Select). Pour cette raison, une requête 2 qui se rapporte à une autre requête 1 ne peut pas être exécutée (en mode vue SQL) à l'aide de la commande **Édition> Exécuter directement l'instruction SQL**, car seule l'interface utilisateur graphique et non la base de données elle-même connaît la requête 1.

La base de données ne vous donne aucun accès direct aux requêtes. Cela s'applique également à l'accès à l'aide de macros. Les vues, en revanche, sont accessibles à partir de macros et de

tables. Cependant, aucun enregistrement ne peut être modifié dans une vue. (Ils doivent être modifiés dans un tableau ou un formulaire.)

Conseils

Une requête créée à l'aide de Créer une requête en mode SQL présente l'inconvénient de ne pas pouvoir être triée ou filtrée à l'aide de l'interface graphique. Il y a donc des limites à son utilisation.

Une vue en revanche peut être gérée dans Base comme une table normale – à l'exception qu'aucune modification des données n'est possible. Ici donc, même dans les commandes SQL directes, toutes les possibilités de tri et de filtrage sont disponibles.

De plus, la mise en forme des colonnes dans une vue est conservée lorsque la base de données est fermée, contrairement aux colonnes d'une requête.

Les vues sont une solution pour de nombreuses requêtes, si vous souhaitez obtenir des résultats. Si, par exemple, une sous-sélection doit être utilisée sur les résultats d'une requête, créez une vue qui vous donne ces résultats. Ensuite, utilisez la sous-sélection sur la vue. Des exemples correspondants se trouvent dans le chapitre 8, Trucs et Astuces.

La création d'une vue à partir d'une requête est plutôt simple et directe.

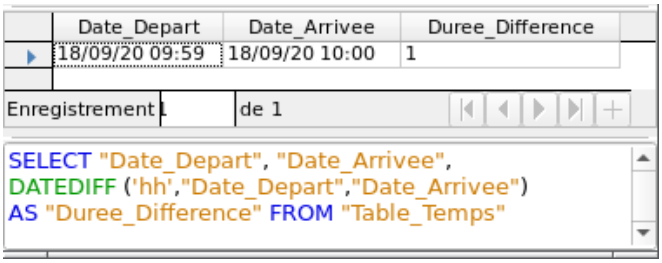
Clic droit sur la requête, puis **Créer en tant que Vue.**

Erreurs de calcul dans les requêtes

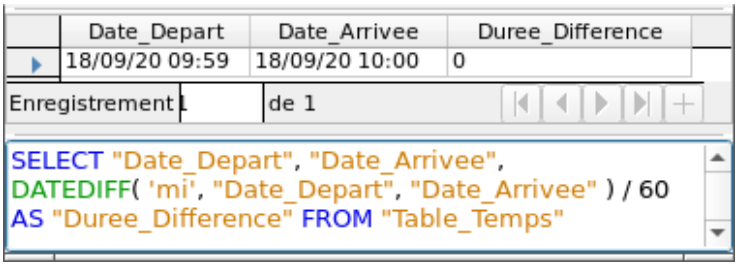
Les requêtes sont également utilisées pour calculer les valeurs. Parfois, la base de données interne HSQLDB produit des erreurs apparentes qui, à un examen plus approfondi, se révèlent être des interprétations logiquement correctes des données. Il existe également des problèmes d'arrondi, qui peuvent facilement prêter à confusion.

Les heures dans HSQLDB ne sont formatées correctement que jusqu'à une différence de 23:59:59 heures. Si plusieurs durées doivent être ajoutées, par exemple pour calculer les heures travaillées, il faut trouver un autre moyen. Ici, il existe plusieurs approches compliquées :

- Le temps est directement exprimé uniquement comme un total de minutes ou même de secondes. Avantage : les valeurs permettent un calcul ultérieur sans problème.
- Le temps est divisé en heures, minutes et secondes et réassemblé sous forme de texte en utilisant «:» comme séparateur. Avantage : le texte apparaît dans les requêtes comme une heure correctement formatée depuis le début.
- L'heure est créée sous forme de nombre décimal. Un jour vaut 1, une heure correspond à 1/24 et ainsi de suite. Avantage : les valeurs peuvent ensuite être reformatées en temps dans la requête et présentées sous forme de champ de formulaire formatable.

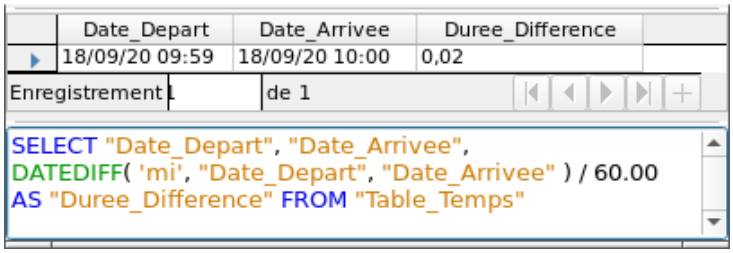


DATEDIFF permet de déterminer les intervalles de temps. Il demande explicitement la différence à déterminer, dans le cas présent les heures ('hh'). . Si, par exemple, le champ Date_Arrivee est défini sur 19/09/20 10:00, cela est calculé comme 25 heures.



Si au contraire l'intervalle de temps est calculé en minutes puis divisé par 60, le décalage horaire en heures devient nul. Cela ressemble plus à la valeur correcte (puiqu'il y a bien 0 heure de décalage) , sauf : où est passée cette minute ?

En utilisant `MOD(DATEDIFF('mi',"Date_Depart",Date_Arrivee"),60`, on obtient bien 1 minute.

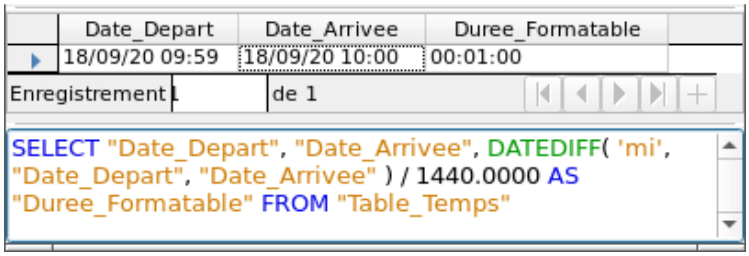


L'intervalle de temps a été donné sous forme de nombre entier. Un entier a été divisé par un entier. Le résultat de la requête dans de tels cas doit également être un entier et non un nombre décimal. Cela peut être facilement corrigé. Le décalage horaire en minutes est divisé par un nombre décimal avec deux décimales (60,00). Cela donne un résultat qui a également deux décimales. 0,02 heure n'est toujours pas exactement une minute, mais c'est beaucoup plus proche qu'avant. Le nombre de décimales peut être augmenté en utilisant plus de zéros. Une valeur de 0,016 est une approximation encore plus proche, mais des erreurs de calcul ultérieures ne peuvent pas toujours être exclues.

Au lieu d'avoir à travailler avec beaucoup de zéros ajoutés, le type de données de DATEDIFF peut être directement influencé. En utilisant `(CONVERT(DATEDIFF('mi',"Date_Depart", "Date_Arrivee"), DECIMAL(50,49)))/60` vous pouvez obtenir une précision de 49 décimales.

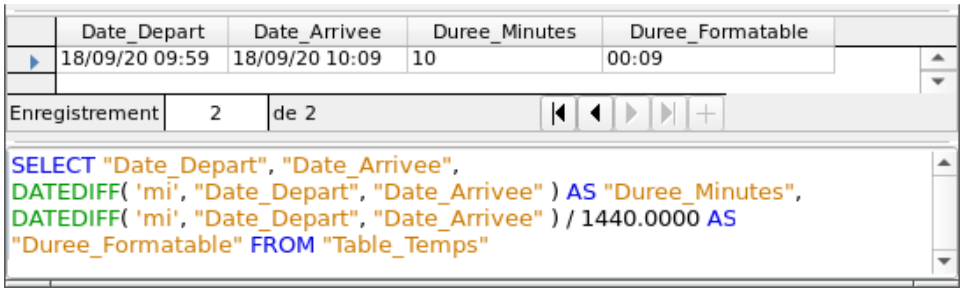
Lorsque vous utilisez des fonctions de calcul, vous devez toujours comprendre que les types de données dans HSQLDB n'ont qu'une précision limitée. Quel que soit le nombre de décimales que vous utilisez, il n'en reste pas moins que les résultats intermédiaires impliquant un décompte du temps ne peuvent être utilisés que dans une mesure limitée pour des calculs ultérieurs.

Si une valeur d'horodatage doit par la suite être utilisée dans un formulaire ou un état en tant qu'heure formatée, vous devez vous assurer que le jour est valide comme base du format d'heure.



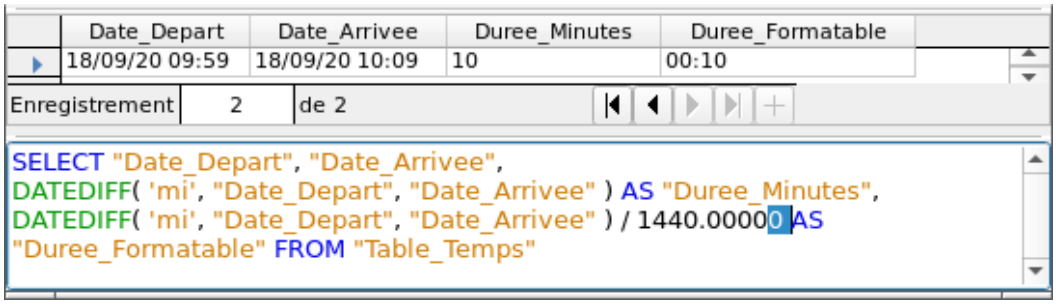
Ci-dessus, la différence est calculée en minutes. Le résultat est donné sous forme de fraction de jour. Un jour a 60 * 24 minutes. Si vous divisez simplement par 1440, le résultat serait zéro, donc encore une fois, vous devez indiquer explicitement les décimales. Il apparaît alors comme une heure formatée de 0 heure et 1 minute. Le formatage est effectué sur la colonne du tableau, après l'affichage.

Le code de format pour une durée supérieure à un jour est [HH] : MM. Si vous utilisez le mauvais format, un décalage horaire de 1 jour et 1 minute peut être affiché comme seulement 1 minute.



Le démon de l'erreur frappe à nouveau ! Un décalage horaire de 10 minutes ne doit pas apparaître comme 9 minutes lorsqu'il est correctement formaté. Pour savoir où se situe le problème, nous devons considérer exactement comment le calcul est effectué :

1) 10/1440 = 0,00694. Le résultat est arrondi à 0,0069 car seules quatre décimales ont été spécifiées.
2) 0,0069 * 1440 = 9,936 minutes, soit 9 minutes 56,16 secondes. Et les secondes ne sont pas affichées dans le format choisi !. Si on choisit le format hh : mm : ss on obtient 00:09:56.

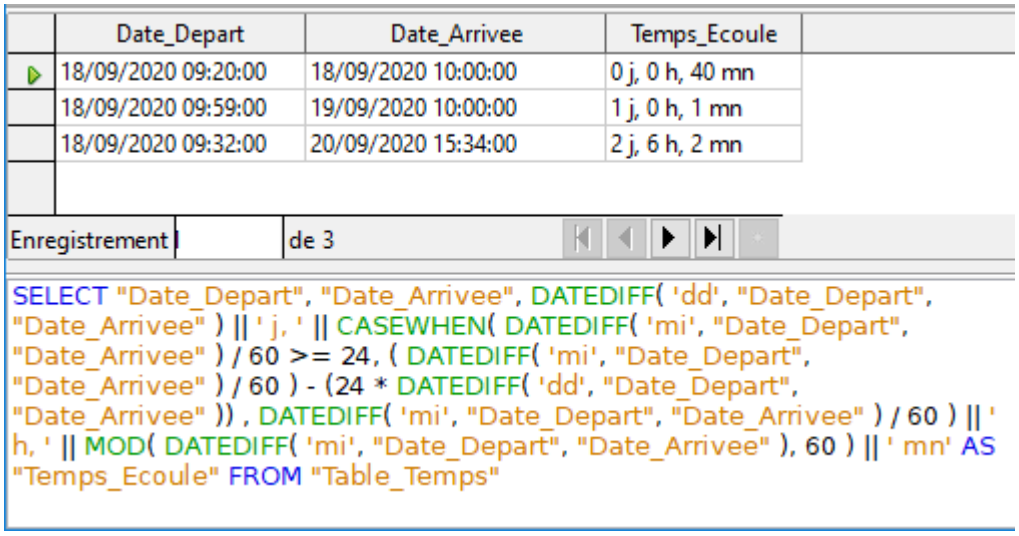


L'allongement du diviseur d'une seule décimale (de 1440,0000 à 1440,00000) corrige cette erreur. Maintenant, l'arrondi est à 0,00694. 0,00694 * 1440 donne 9,9936, soit 59,616 secondes. Le nombre de secondes est arrondi à 60 secondes, donc les 9 minutes ont 1 minute ajoutée, soit 10 minutes en tout.

Ici aussi, il pourrait y avoir d'autres problèmes. Peut-il y avoir d'autres décimales qui, une fois formatées, ne donnent pas 1 minute ? Pour résoudre ce problème, un court calcul utilisant Calc avec des chiffres arrondis de la même manière peut aider. La colonne A contient une séquence de nombres à partir de 1 (pour les minutes). La colonne B contient la formule = ROUND (A1/ 440 ; 4) et est formatée pour afficher les heures et les minutes. Si cela continue vers le bas, nous pouvons voir, à côté de 10 minutes dans la colonne A, 00:09 dans la colonne B. De même pendant 28 minutes, etc. Si vous arrondissez à 5 places, ces erreurs disparaissent.

Aussi agréable que cela puisse être d'avoir un affichage correctement formaté dans un formulaire, vous devez être conscient que vous avez affaire à des valeurs arrondies qui ne conviennent pas pour une utilisation ultérieure dans des calculs mécaniques aveugles. Si une valeur doit être utilisée pour un calcul ultérieur, il est préférable d'utiliser uniquement une représentation du décalage horaire dans les plus petites unités disponibles, dans ce cas les minutes.

Une alternative est fournie ci- dessous



Date_Depart	Date_Arrivee	Temps_Ecoule
18/09/2020 09:20:00	18/09/2020 10:00:00	0 j, 0 h, 40 mn
18/09/2020 09:59:00	19/09/2020 10:00:00	1 j, 0 h, 1 mn
18/09/2020 09:32:00	20/09/2020 15:34:00	2 j, 6 h, 2 mn

Enregistrement de 3

```
SELECT "Date_Depart", "Date_Arrivee", DATEDIFF( 'dd', "Date_Depart",
"Date_Arrivee" ) || ' j, ' || CASEWHEN( DATEDIFF( 'mi', "Date_Depart",
"Date_Arrivee" ) / 60 >= 24, ( DATEDIFF( 'mi', "Date_Depart",
"Date_Arrivee" ) / 60 ) - (24 * DATEDIFF( 'dd', "Date_Depart",
"Date_Arrivee" )) , DATEDIFF( 'mi', "Date_Depart", "Date_Arrivee" ) / 60 ) || '
h, ' || MOD( DATEDIFF( 'mi', "Date_Depart", "Date_Arrivee" ), 60 ) || ' mn' AS
"Temps_Ecoule" FROM "Table_Temps"
```

La requête affiche :

- les champs Date_Depart et Date_Arrivee. Ils sont de type DATE/Heure [TIMESTAMP] au format JJ/MM/AAAA HH:MM:SS ;
- Le temps écoulé qui est le résultat de la concaténation des calculs détaillés ci-dessous (Attention, c'est une chaîne de caractères).

  `DATEDIFF ('dd', "DATE_Depart", "Date_Arivee")`
  différence en jours entre les deux dates.

  `CASEWHEN(DATEDIFF('mi', "Date_Depart", "Date_Arivee")/60 >= 24).`
  Dans le cas ou la différence en minutes entre les deux dates, (divisée par 60 pour obtenir des heures) est suppérieure à 24.
  Alors, le résultat sera `(DATEDIFF('mi', "Date_Depart", "Date_Arivee")/60 – (24 * DATEDIFF ('dd', "DATE_Depart", "Date_Arivee"))`
  La différence en minutes, entre les deux dates, divisée par 60 (pour obtenir des heures) – 24 (heures) multiplié par la différence en jours entre les dates. Exemple (enregistrement n°3) : le résultat de `DATEDIFF('mi', "Date_Depart", "Date_Arivee")/60` donne 54 (heures) alors on obtiendra 54 – (24 * 2[jours]) = 6 heures.
  Sinon le résultat sera `DATEDIFF('mi', "Date_Depart", "Date_Arivee")`/60.

  `MOD(DATEDIFF('mi', "Date_Depart", "Date_Arivee"),60)`
  MOD (Modulo) donne le reste entier de la division de `DATEDIFF('mi', "Date_Depart", "Date_Arivee")`, par 60.
  (Nous le répétons : Attention, c'est une chaîne de caractères. Aucun calcul ne peut être réalisé à partir de ces résultats).

Pour plus de précisions sur la sujet, lire le Guide des fonctions intégrées à HSQLDB

Annexe : Notation BNF

BNF signifie Backus Naur Form (voir https://fr.wikipedia.org/wiki/Forme_de_Backus-Naur).

Tableau succinct des signes utilisés dans le présent document.

|	Ou
[ ]	Les éléments compris entre [et] sont facultatifs (optionnels).
{ }	Les accolades { } encadrent des éléments répétitifs.

||	Sert à concaténer plusieurs éléments en une seule chaîne de caractère. Les éléments de type numérique ou de type date, sont automatiquemet convertis en chaînes.

Traduction d'un exemple simple :

[+ | -] { expression [{ + | – | * | / | || } expression]}

+ ou (|) – peuvent facultativement [] se trouver devant ce qui suit ;

Le second élément est constitué à minima d'une expression (une chaîne de caractère, un nombre, un résultat).

Celle-ci peut optionnellement être suivie de :

- + ou - ou * ou / (opérateurs mathématiques pour des expressions constituées par des valeurs numériques) ;
- `ou par un opérateur de concaténation || (pour les expressions de type chaînes de caractères ;`
- et d'une nouvelle expression.

Et ceci autant de fois que souhaité.