Guide Base

Chapitre 10 Maintenance de Bases de Données

Droits d’auteur

Contributeurs

De cette édition

Pulkit Krishna

Des éditions précédentes

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

Traduction

Jean-Michel COSTE

Relecteurs

Patrick Auclair

Retour d’information

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

Note

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

Date de publication et version du logiciel

Publié en avril 2021. Basé sur LibreOffice 6.4..

Table des matières

Remarques générales sur la gestion des bases de données

La modification fréquente des données dans une base de données – en particulier la suppression fréquente des données – a deux effets. Premièrement, la base de données s'agrandit régulièrement, même si elle ne contient en fait pas plus de données. Deuxièmement, la clé primaire créée automatiquement continue de s'accroître, quelle que soit la valeur de la clé suivante nécessaire. Plusieurs importantes méthodes de maintenance sont décrites dans ce chapitre.

Compacter une base de données

Hsqldb a la particularité de continuer à fournir un espace de stockage même pour les données qui ont déjà été supprimées. Les bases de données qui étaient autrefois remplies de données, en particulier d'images, à des fins de tests, ont toujours la même taille, même si toutes ces données ont été supprimées.

Pour libérer à nouveau l'espace de stockage, les fichiers de la base de données doivent être réécrits (tables, descriptions de ces tables, etc.).

Directement sur l'interface de la Base, une commande simple peut être saisie directement via **Outils → SQL** qui est réservé à l'administrateur du système dans le cas des bases de données serveur :

```
SHUTDOWN COMPACT
```

La base de données est arrêtée et libérée de tous les anciens chargements. Toutefois, la base doit être redémarrée par la suite si l'on veut y accéder à nouveau.

Depuis la version LO 3.6, la compression est effectuée par défaut au plus tard à la fermeture de la base de données.

Réinitialisation des valeurs automatiques

Lorqu'une base de données est créée, toutes les fonctions possibles testées avec des exemples et des corrections peuvent être apportées jusqu'à ce que tout fonctionne. Alors, avant même qu'une base de données ne soit prête à être utilisée, il est possible que les valeurs de clé primaire prennent une valeur trop importante. Souvent, les clés primaires sont définies sur l'incrémentation automatique. Si les tables sont vidées en vue d'une utilisation normale ou avant de transmettre la base de données à une autre personne, la clé primaire continue à s'incrémenter à partir de sa position actuelle au lieu de se réinitialiser à zéro.

La commande SQL suivante, entrée via **Outils> SQL**, vous permet de réinitialiser la valeur initiale :

```
ALTER TABLE "Nom_Table" ALTER COLUMN "ID" RESTART WITH NouvelleValeur
```

Cela suppose que le champ de clé primaire a pour nom ID et a été défini comme un champ de valeur automatique. La nouvelle valeur doit être celle que vous souhaitez créer automatiquement pour le nouvel enregistrement suivant. Ainsi, par exemple, si les enregistrements actuels vont jusqu'à 4, la nouvelle valeur doit être 5 sans modifier le champ ID. La première valeur d'ID sera la NouvelleValeur dans l'instruction SQL ci-dessus.

Interroger les propriétés de la base de données

Toutes les informations sur les tables de la base de données sont stockées sous forme de table dans une partie distincte de HSQLDB. Cette zone distincte peut être atteinte en utilisant le nom **INFORMATION_SCHEMA**.

La requête suivante peut être utilisée pour trouver les noms de champ, les types de champ, les tailles de colonne et les valeurs par défaut. Voici un exemple de table nommée TableCherche.

```
SELECT "COLUMN_NAME",
```

```
    "TYPE_NAME",
    "COLUMN_SIZE",
    "COLUMN_DEF" AS "Valeur defaut"
FROM "INFORMATION_SCHEMA"."SYSTEM_COLUMNS"
WHERE "TABLE_NAME" = "TableCherche"
ORDER BY "ORDINAL_POSITION"
```

Toutes les tables spéciales de HSQLDB sont décrites dans l'annexe A de ce livre. Les informations sur le contenu de ces tables sont facilement obtenues par des requêtes directes :

```
SELECT * FROM "INFORMATION_SCHEMA"."SYSTEM_PRIMARYKEYS"
```

L'astérisque (*) garantit que l'ensemble des colonnes seront retournées La table explorée ci-dessus retourne les informations essentielles sur les clés primaires et les différentes tables.

Ces informations sont surtout utiles pour les macros. Au lieu d'avoir à fournir des informations détaillées sur chaque table ou base de données nouvellement créée, des procédures sont écrites pour extraire ces informations directement de la base de données et sont donc universellement applicables. L'exemple de base de données le montre, entre autres, dans l'un des modules de maintenance, où les clés étrangères sont déterminées. Firebird contient des informations correspondantes, mais malheureusement un peu plus dispersées dans les bases de données de son système :

```
SELECT "a".RDB$RELATION_NAME AS "Tables",
   "a".RDB$FIELD_NAME AS "Champs",
   "c".RDB$TYPE_NAME AS "Types",
   "a".RDB$Field_POSITION AS "Position_Champs",
   "a".RDB$NULL_FLAG AS "Flag_Null",
   "b".RDB$FIELD_LENGTH AS "Longueur_Champs"
FROM RDB$RELATION_FIELDS AS "a", RDB$FIELDS AS "b", RDB$TYPES AS "c"
WHERE "a".RDB$FIELD_SOURCE = "b".RDB$FIELD_NAME
   AND "b".RDB$FIELD_TYPE = "c".RDB$TYPE
   AND "c".RDB$FIELD_NAME = 'RDB$FIELD_TYPE'
   AND "a".RDB$RELATION_NAME = 'Nom_Relation'
ORDER BY "Tables", "Position_Champs"
```

Exporter des données

Il existe une méthode beaucoup plus simple d'exportation de données que la méthode standard, en ouvrant le fichier *.odb. Directement sur l'interface de base, vous pouvez utiliser **Outils> SQL** pour saisir une commande simple. Dans les bases de données serveur, celle-ci est réservée à l'administrateur système.

```
SCRIPT 'Nom_de_Fichier'
```

'Nom_De_Fichier' doit être de la forme : Chemin\ Nom_De_Fichier

Exemple (sous windows) : C:\Users\...\Exemple_Cherche_et_Filtre.SQL ou
sous Linux *:* /home/.../ *E*xemple_Cherche_et_Filtre.SQL

Cela crée une extraction SQL complète de la base de données avec toutes les définitions de table, les relations entre les tables et les données. Les requêtes et les formulaires ne sont pas extraits, car ils ont été créés dans l'interface utilisateur et ne sont pas stockés dans la base de données interne. Cependant, toutes les vues sont incluses.

Note

Cette procédure peut être utilisée pour mettre à jour une base de données intégrée pour la connexion à la base de données externe avec HSQLDB 2.50. Là encore, les requêtes et les formulaires doivent être remplacés.

Par défaut, le fichier exporté est un fichier texte normal,

Le fichier peut également être fourni sous forme binaire ou compressé (sous forme zippée), ce qui est utile pour les bases de données volumineuses. Cependant, cela rend sa réimportation dans LibreOffice Base un peu plus compliquée.

Le format du fichier exporté peut être modifié en utilisant :

```
SET SCRIPTFORMAT {TEXT | BINARY | COMPRESSED}
```

Pour exporter le fichier, vous devez utiliser ce code SQL une ligne à la fois : :

```
SCRIPT 'Nom_De_Fichier'
SET SCRIPTFORMAT {TEXT | BINARY | COMPRESSED}
CHECKPOINT
SHUTDOWN SCRIPT
```

La documentation concernant ces commandes se trouvent dans le guide (en anglais) pour HSQLDB 1.8 à l'adresse : http://www.hsqldb.org/doc/1.8/guide/guide.html ou guide.pdf.

Le fichier peut être importé en utilisant **Outils> SQL** et en créant une nouvelle base de données avec les mêmes données. Dans le cas d'une base de données interne, les lignes suivantes doivent être supprimées avant l'importation :

```
CREATE SCHEMA PUBLIC AUTHORIZATION DBA
CREATE USER SA PASSWORD ""
GRANT DBA TO SA
SET WRITE_DELAY 60
SET SCHEMA PUBLIC
```

Ces entrées traitent du profil utilisateur et d'autres paramètres par défaut, qui sont déjà définis pour les bases de données internes de LibreOffice. Par conséquent, un message d'erreur apparaît si l'une de ces lignes est présente. Ils se trouvent directement avant le contenu qui sera inséré dans les tables à l'aide de la commande INSERT.

Pour importer ce fichier, son contenu doit être divisé en plusieurs fichiers texte créés par un simple programme d'édition de texte. Le premier fichier doit contenir toutes les tables et vues de création. Copiez toutes les lignes depuis le début avec CREATE TABLE et en arrêtant une ligne au-dessus de la ligne contenant INSERT INTO. Collez-le dans le premier fichier. Copiez et collez les lignes restantes dans le deuxième fichier.

Il y a une limite à la taille du deuxième fichier : il doit être inférieur à 65 Ko. S'il est plus grand que cela, il doit également être divisé en fichiers texte plus petits en coupant et en collant. Assurez-vous simplement que la première ligne de chacun de ces nouveaux fichiers commence par INSERT INTO. Une façon de faire est de couper à partir du bas jusqu'à une telle ligne.

Tester les table pour les entrées inutiles

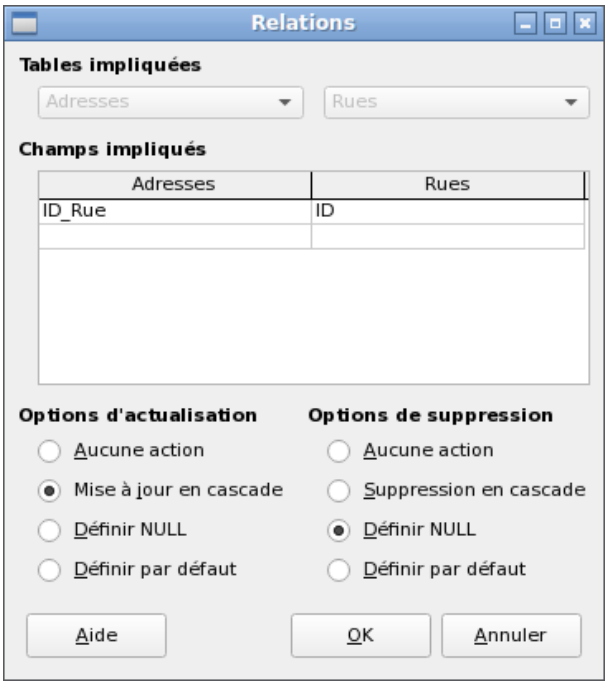
Une base de données se compose d'une ou plusieurs tables principales, qui contiennent des clés étrangères d'autres tables. Dans l'exemple de base de données, il s'agit des tables Medias et Adresses. Dans la table d'adresses, la clé primaire du code postal apparaît comme une clé étrangère. Si une personne déménage dans une nouvelle maison, l'adresse est modifiée. Le résultat peut être qu'aucune clé étrangère **ID_CodePostal** correspondant à ce code postal n'existe plus. En principe, le code postal lui-même pourrait donc être supprimé. Cependant, lors d'une utilisation normale, il n'est pas évident que l'enregistrement n'est plus nécessaire. Il existe différentes manières d'éviter ce genre de problème.

Test des entrées à l'aide de la définition de relation

L'intégrité des données peut être garantie lors de la définition des relations. En d'autres termes, vous pouvez empêcher la suppression ou la modification de clés de conduire à des erreurs dans la

base de données. La boîte de dialogue suivante est accessible via **Outils> Relations**, suivi d'un clic droit sur le connecteur entre deux tables.

Ici, les tables **Adresses** et **Rues** sont considérées. Toutes les actions spécifiées s'appliquent à la table Adresses, qui contient la clé étrangère **ID_Rue**. Les options de mise à jour font référence à une mise à jour du champ **ID** dans la table **Rues**. Si la clé numérique dans le champ **"Rues"."ID"** est modifiée, **Aucune action** signifie que la base de données n'autorise pas ce changement si un **"Rues"."ID"** avec ce numéro de clé apparaît comme clé étrangère dans la table d'adresses.



La **Mise à jour en cascade** signifie que le numéro de clé est simplement reporté. Si la rue 'de la fontaine' dans la table Rues a l'ID '3' et est également représentée dans "Adresses"."ID_Rue", l'ID peut être modifié en toute sécurité. Par exemple, si elle est remplacée par "67", les valeurs "Adresses"."ID_Rue" correspondantes seront automatiquement remplacées par "67".

Si **Définir Null** est choisi, la modification de l'ID fait de "Adresses"."ID_Rue" un champ vide.

Les options de suppression sont gérées de la même manière.

Pour les deux options, l'interface graphique actuelle n'autorise pas la possibilité **Définir par défaut** (même si les boutons sont présents) car les paramètres par défaut de l'interface graphique sont différents de ceux de la base de données. Voir le chapitre 3, Tables.

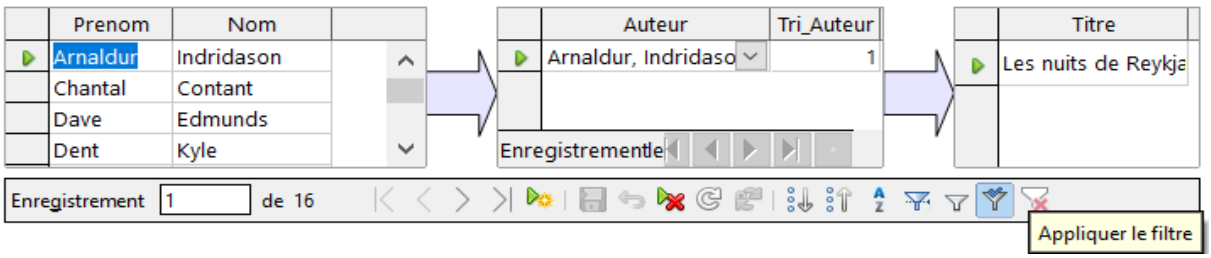
La définition de relations permet de garder les relations elles-mêmes propres, mais cela ne supprime pas les enregistrements inutiles qui fournissent leur clé primaire en tant que clé étrangère dans la relation. Il peut y avoir n'importe quel nombre de rues sans adresses correspondantes.

Modification d'entrées à l'aide de formulaires et de sous-formulaires

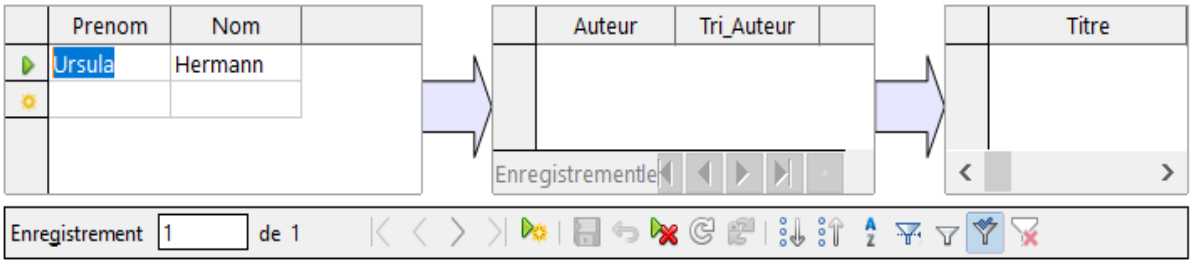
En principe, toute l'interrelation entre les tables peut être affichée dans des formulaires. Ceci est bien sûr plus simple lorsqu'une table est liée à une seule autre table. Ainsi, dans l'exemple suivant, la clé primaire de l'**auteur** devient la clé étrangère dans la table **relation_Media_Auteur**. rel_Media_Auteur contient également une clé étrangère de **Medias**, de sorte que la disposition suivante montre une relation **n:m** avec trois formes. Chacune est présentée à travers une table.

Les illustrations suivantes proviennent de captures d'écran du formulaire : **Maintenance**, dans la base : **Exemple_Media_Sans_Macro.odb**.

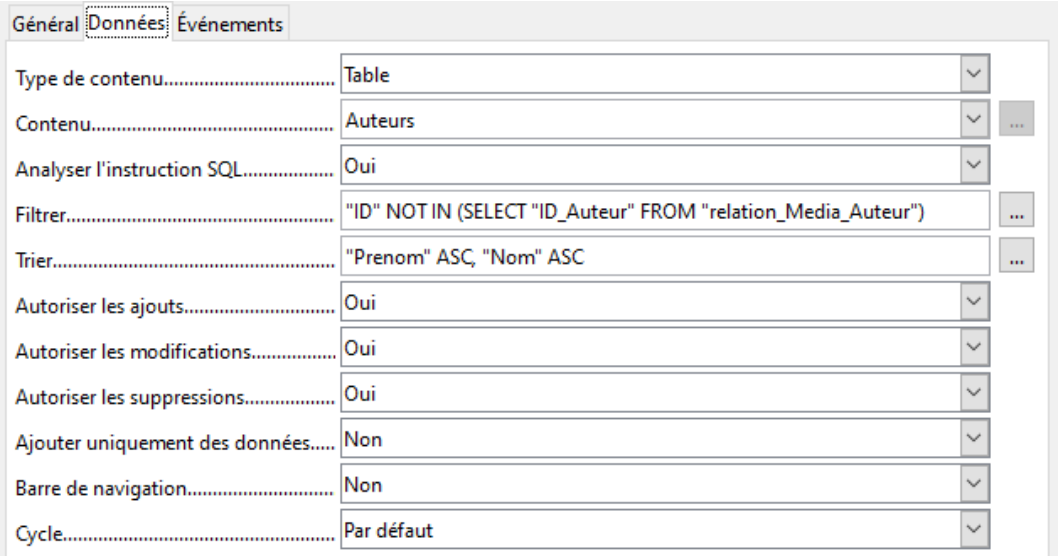
La première figure montre que le titre *Les nuits de Reykjavik* appartient à l'auteur *Arnaldur Indridason*. Donc cet auteur ne doit pas être supprimé – sinon les informations requises pour le média *Les nuits de Reykjavik* sera manquant. Cependant, la zone de liste vous permet de choisir un enregistrement différent à la place de *Arnaldur Indridason.*



Dans le formulaire, il y a un filtre intégré dont l'activation peut vous dire quelles catégories ne sont pas nécessaires dans la table **Medias**. Dans le cas qui vient d'être décrit, presque tous les auteurs sont lié à un média. Seul l'enregistrement Ursula Hermann peut être supprimé sans aucune conséquence pour tout autre enregistrement dans Medias. En effet il n'est relié à aucun média.



Le filtre est codé en dur dans ce cas. Il se trouve dans les propriétés du formulaire.



Un tel filtre est activé automatiquement au lancement du formulaire. Il peut être désactivé et activé. S'il est supprimé, il est à nouveau accessible par un rechargement complet du formulaire. Cela signifie plus qu'une simple mise à jour des données l'ensemble du formulaire doit être fermé puis rouvert.

Requêtes pour rechercher des entrées orphelines

Le filtre ci-dessus fait partie d'une requête qui peut être utilisée pour rechercher des entrées orphelines.

```
SELECT "Nom", "Prenom" FROM "Auteurs" WHERE "ID" NOT IN (SELECT "ID_Auteur"
FROM "relation_Media_Auteur")
```

Si une table contient des clés étrangères de plusieurs autres tables, la requête doit être étendue en conséquence. Cela affecte, par exemple, la table **Villes**, qui a des clés étrangères à la fois dans la table **Medias** et dans la table **CodePostal**. Par conséquent, les enregistrements de la table **Villes** qui doivent être supprimés ne doivent être référencés dans aucune de ces tables. Ceci est déterminé par la requête suivante : :

```
SELECT "Ville" FROM "Villes" WHERE "ID" NOT IN (SELECT "ID_Ville" FROM
"Medias") AND "ID" NOT IN (SELECT "ID_Ville" FROM "CodePostal")
```

Les entrées orphelines peuvent ensuite être supprimées en sélectionnant toutes les entrées qui passent le filtre défini et en utilisant l'option Supprimer du menu contextuel du pointeur d'enregistrement, appelée par un clic droit.

Vitesse de recherche dans la base de données

Effet des requêtes

Ce ne sont que ces requêtes, utilisées dans la section précédente pour filtrer les données, qui s'avèrent insatisfaisantes au regard de la vitesse maximale de recherche dans une base de données. Le problème est que dans les grandes bases de données, la sous-requête récupère une quantité proportionnellement importante de données avec lesquelles chaque enregistrement affichable unique doit être comparé. Seules les comparaisons avec la relation 1-n permettent de comparer une seule valeur avec un ensemble de valeurs. La requête

```
… WHERE "ID" NOT IN (SELECT "ID_Auteur" FROM "relation_Media_Auteur")
```

peut contenir un grand nombre de clés étrangères possibles de la table **relation_Media_Auteur,** qui doivent d'abord être comparées aux clés primaires de la table **Auteurs** pour chaque enregistrement de cette table. Une telle requête n'est donc pas adaptée à une utilisation quotidienne mais peut être nécessaire pour la maintenance de la base de données

Les fonctions de recherche doivent être structurées différemment afin que la recherche de données ne prenne pas un temps interminable et ne gâche pas le travail avec la base de données dans les opérations quotidiennes.

Effet des zones de liste et des zones combinées

Plus un formulaire contient de zones de liste et plus le contenu qu'elles doivent fournir est important, et de ce fait plus la construction du formulaire prend du temps.

Les listes de sélection sont créées par des requêtes qui sont exécutées, pour chaque liste de sélection, lorsque le formulaire est lancé.

Plus Base met à disposition, en priorité, un affichage complet pour l'utilisateur, et ne lit que partiellement les zones de liste, moins la charge correspondante est perceptible.

La même structure de requête, pour davantage de zones de listes, est plus performante en utilisant une vue commune, au lieu de créer à plusieurs reprises des champs avec la même syntaxe en utilisant des commandes SQL stockées dans les zones de listes.

Les vues sont avant tout préférables pour les systèmes de bases de données externes, car dans ce cas le serveur s'exécute beaucoup plus rapidement qu'une requête qui doit être rassemblée par l'interface graphique et envoyée au serveur à chaque fois. Le serveur traite les vues comme des requêtes locales complètes.

Influence du système de base de données utilisé

Le SGBD (Système de Gestion de Base de Données) interne HSQLDB est conçu pour garantir que Base et Java fonctionnent bien ensemble.

Lorsque Base utilise HSQLDB, la taille et la réactivité de votre base sont limitées par rapport à l'utilisation d'un moteur de base de données externe. En particulier lorsque ce serveur de base de données est exécuté sur un ordinateur distinct.

Si les fonctions de votre base de données commencent à ralentir, suivez d'abord les étapes de ce guide pour nettoyer les espaces vides, les données supprimées ou temporaires et vérifiez que vous utilisez des index là où ils ont du sens. Si la réactivité ne revient pas, envisagez de déplacer vos données du fichier de base vers un serveur de base de données externe.

Les bases de données externes s'exécutent beaucoup plus rapidement. En termes de vitesse, les connexions directes à MySQL ou PostgreSQL et les connexions via ODBC (Open DataBase Connectivity) sont presque équivalentes. JDBC (Java DataBase Connectivity) dépend également de l'interaction avec Java, mais fonctionne beaucoup plus rapidement qu'une connexion interne à HSQLDB.