Ch. 5. Problèmes liés au déploiement

De Apache OpenOffice Wiki
< FR | Documentation | HSQLDB Guide

Chapitre 5. Problèmes liés au déploiement

(Deployment Issues)

Fred Toussi

HSQLDB Development Group

<ft@cluedup.com>

$Date: 2005/07/02 09:11:39 $

Sommaire

But de ce document

Beaucoup de questions fréquemment posées dans les forums et mailing-lists trouvent leurs réponses dans ce guide. Si vous voulez utiliser HSQLDB avec votre application, vous devez lire ce guide. Ce document couvre les problèmes relatifs au système. Pour les problèmes relatifs au SQL voyez le chapitre : Problèmes liés au SQL.

Mode opératoire et tables

(Mode of Operation and Tables)

HSQLDB possède des modes opératoires et des fonctionnalités qui lui permette d'être utilisé dans des scénarios très différents. Les niveaux de l'utilisation de la mémoire, la vitesse et l'accessibilité sont influencés par la manière dont HSQLDB est déployé.

Mode opératoire

(Mode of Operation)

La décision d'exécuter HSQLDB comme un processus de serveur à part ou comme une base de données "in-process" doit être basée sur les faits suivants

- Quand HSQLDB est exécuté comme un serveur sur une machine séparée, il est isolé des pannes matérielles et plantages de l'hôte qui exécute l'application.
- Quand HSQLDB est exécuté comme un serveur sur la même machine, il est isolé des plantages de l'application et des fuites (leaks) de mémoire.
- Les connexions au serveur sont plus lentes que les connexions "in-process" à cause de la surcharge du flux de données pour chaque appel JDBC.

Tables

Les tables texte sont conçues (désignées) pour des applications spéciales où les données doivent être dans un format interchangeable, tel le format CSV. Les tables texte ne devraient pas être utilisées pour un stockage routinier des données.

Les tables mémoire et en cache sont généralement utilisées pour le stockage des données. Les différences entre ces deux types sont énumérées ci-dessous :

- Pour les tables mémoire, les données sont lues depuis le fichier .script quand la base de données est démarrée et chargée en mémoire. Au contraire des tables en cache pour lesquelles les données ne sont chargées en mémoire qu'après un accès à la table. De plus pour les tables en cache seulement une partie des données est chargée en mémoire, ce qui permet aux tables de dépasser la mémoire totale allouée.
- Quand la base de données est normalement fermée, toutes les données des tables mémoire sont écrites sur disque. Par comparaison, seules les données modifiées seront écrites dans les tables en cache à la fermeture, plus une sauvegarde compressée de l'intégralité des données pour toutes les tables en cache.
- La taille et la capacité du cache des données est configurable pour toutes les tables en cache. Ceci permet d'allouer au cache mémoire toutes les données des tables en cache. Dans ce cas, la vitesse d'accès est bonne, bien que légèrement plus lente qu'avec les tables mémoire.

- Pour la plupart des applications il est recommandé d'utiliser les tables mémoire pour les petites quantités de données, en réservant les tables en cache pour les grands jeux de données. Pour certaines applications spéciales dans laquelle la vitesse joue un rôle de première importance et si une grande quantité de mémoire libre est disponible, on peut utiliser aussi bien les tables mémoire pour de grandes tables.

Les grands objets

(Large Objects)

- Voir aussi CLOB (en) (http://en.wikipedia.org/wiki/CLOB)
- Voir aussi BLOB (en) (http://en.wikipedia.org/wiki/BLOB)

Les "types" CLOB (fr) (http://fr.wikipedia.org/wiki/CLOB) JDBC sont pris en charge par les colonnes de type LONGVARCHAR. Les "types" BLOB (fr) (http://fr.wikipedia.org/wiki/BLOB) JDBC sont eux supportés par les colonnes de type LONGVARBINARY. Si de grands objets (LONGVARCHAR, LONGVARBINARY, OBJECT) sont stockés dans des définitions d'une table qui contient plusieurs autres champs normaux, il serait en fait préférable de scinder cette table en deux. La première table contenant les champs normaux et la seconde contenant les grands objets plus un champ clé primaire. Utiliser cette méthode a deux avantages. (a) La première table peut être généralement créée comme une table mémoire alors que seulement la seconde est de type CACHED. (b) Les grands objets peuvent être retrouvés individuellement par l'utilisation de leur clé primaire, plutôt que d'avoir à charger les données en mémoire pour trouver les lignes durant l'exécution de la requête. Un exemple de deux tables et une requête sélection qui exploite la séparation entre les deux :

```
CREATE MEMORY TABLE MAINTABLE(MAINID INTEGER, ......);

CREATE CACHED TABLE LOBTABLE(LOBID INTEGER, LOBDATA LONGVARBINARY);

SELECT * FROM (SELECT * FROM MAINTABLE <join any other table> WHERE <various conditions apply>) JOIN
```

La commande SELECT trouve les lignes demandées sans faire référence à LOBTABLE, et lorsque toutes les lignes sont trouvées, récupère les grands objets associés de LOBTABLE.

Contexte de déploiement

(Deployment context)

Les fichiers nécessaires pour stocker les données de bases de données HSQLDB sont tous situés dans le même répertoire. Les nouveaux fichiers sont toujours créés et supprimés par le moteur de la base de données. Deux principes simples doivent être observés :

- Le processus Java exécutant HSQLDB doit avoir les pleins privilèges sur le dossier où les fichiers sont enregistrés. Y compris les droits de création et de suppression.
- Le système de fichiers doit avoir assez de place disponible à la fois pour les fichiers permanents et temporaires. La taille par défaut maximum du fichier .log est de 200 Mo. Le fichier .data peut quant à lui occuper jusqu'à 8 Go. Le fichier .backup peut atteindre jusqu'à 50% du fichier .data. Le fichier temporaire créé au moment d'un SHUTDOWN COMPACT peut être égal en taille au fichier .data.

Utilisation de la mémoire et du disque

(Memory and Disk Use)

La mémoire utilisée par le programme peut être vue comme deux groupes distincts : La mémoire utilisée pour les données de tables, et la mémoire utilisée pour construire les jeux de résultats et autres opérations internes. De plus, lors de l'exécution de transactions, la mémoire est utilisée pour stocker l'information nécessaire à une annulation (rollback).

Depuis la version 1.7.1, l'utilisation de la mémoire a été sensiblement améliorée comparativement aux versions précédentes. La mémoire utilisée pour une table mémoire est la somme de la mémoire utilisée par chaque ligne. Chaque ligne de table MEMORY est un objet Java comprenant 2 variables int ou de référence. Il contient un tableau d'objets pour les champs de la ligne. Chaque champ est un objet tel que Integer, Long, String, etc. De plus chaque index de la table ajoute un objet nœud à la ligne. Chaque objet nœud a 6 variables int ou de référence. Finalement, une table d'une colonne de type INTEGER aura quatre objets par ligne, pour un total de dix variables de quatre octets chacune - occupant donc couramment jusqu'à quatre-vingt octets par ligne. Bien sur, chaque colonne supplémentaire ajoute au moins quelques octets à la taille de chaque ligne.

La mémoire utilisée pour renvoyer un jeu de lignes (result set row) est moins gourmande (moins de variables et pas de nœuds d'index) mais utilise toujours beaucoup de mémoire. Toutes les lignes du jeu de résultat sont construites en mémoire, et donc de très grands jeux de résultats peuvent ne pas être possible. Dans les bases de données en mode serveur, le jeu de résultats en mémoire est envoyé par le serveur une fois que le serveur de la base de données a lui-même renvoyé le jeu de résultats. Les bases de données "in-process" libèrent la mémoire quand l'application renvoie l'objet java.sql.ResultSet. Les modes serveur requièrent de la mémoire additionnelle pour retourner les jeux de résultats, puisqu'ils convertissent l'intégralité du jeu de résultats en un tableau d'octets qui est ensuite transmit au client.

Quand des requêtes UPDATE et DELETE sont exécutées sur des tables CACHED, tout le jeu de lignes affectées, incluant celles affectées suite aux actions ON UPDATE, est retenu en mémoire pour la durée de l'opération. Ce qui signifie qu'il peut être impossible d'exécuter des suppressions ou mises à

jour invoquant de très grands nombres de lignes pour les tables CACHED. De telles opérations devront être effectuées en plusieurs plus petits jeux de lignes.

Quand la prise en charge des transactions est disponible avec SET AUTOCOMMIT OFF, des listes de toutes les insertions, opérations de suppressions ou de mises à jour sont stockées en mémoire de façon a pouvoir être annulées lors de l'envoi d'une commande ROLLBACK. Des transactions englobant des centaines de modifications des données prendront beaucoup de mémoire jusqu'à ce qu'un prochain COMMIT ou ROLLBACK nettoie ces listes.

La plupart des implémentations de machines virtuelles Java allouent un maximum de mémoire (généralement 64 Mo par défaut). Cette quantité n'est généralement pas suffisante lors de l'utilisation de grandes tables mémoire, ou quand la taille moyenne des lignes des tables en cache dépasse quelques centaines d'octets. La quantité maximum de mémoire allouée peut être définie sur la ligne de commande Java utiliser pour exécuter HSQLDB. Par exemple, avec une JVM Sun version 1.3.0 le paramètre -Xmx256m augmente la quantité de mémoire allouée à 256 Mo.

La version 1.8.0 utilise un cache rapide pour des objets tels que Integer ou String qui sont stockés dans la base de données. Dans la plupart des cas, ceci réduit encore plus la quantité de mémoire non rafraichie puisque moins de copies des objets les plus fréquemment utilisés sont conservées en mémoire.(In most circumstances, this reduces the memory footprint still further as fewer copies of the most frequently-used objects are kept in memory.)

Allocation du cache mémoire

(Cache Memory Allocation)

Avec les tables en cache, les données sont stockées sur le disque et il y a une limite au nombre maximum de lignes contenues en mémoire à tout instant. La valeur par défaut va jusqu'à 3*16384 lignes. Vous pouvez ajuster la propriété de base de données hsqldb.cache_scale pour changer ce nombre. Pour tout sous-jeu de lignes (random subset of the rows) de n'importe quelle table en cache qui peut être retenu dans le cache, la quantité de mémoire nécessaire pour les lignes en cache peut atteindre la somme des lignes contenant le plus grand champ de données. Par exemple si une table de 100 000 ligne en contient 40 000 d'un poids de 1 000 octets chacune et 60 000 d'un poids de 100 octets (chacune également), le cache peut grossir jusqu'à pouvoir contenir à peu près 50 000 lignes, incluant les 40 000 lignes les plus "lourdes".

Vous pouvez utiliser la propriété additionnelle hsqldb.cache_size_scale en conjonction avec la propriété hsqldb.cache_scale. Celle-ci déclare une limite en octets de la taille totale des lignes qui sont en cache. Avec les valeurs par défaut de ces deux propriétés, la limite de la taille totale des lignes est approximativement de 50 Mo. (C'est la taille des images binaires des lignes et index. Ceci est en fait traduit par plus de mémoire, typiquement 2-4 fois, utilisée pour le cache parce que les données sont représentées par des objets

Java.)

Si la mémoire est limitée, les propriétés de base de données hsqldb.cache_scale ou hsqldb.cache_size_scale peuvent l'être également. Dans l'exemple ci-dessus si hsqldb.cache_size_scale est ramenée de 10 à 8, la limite de la taille totale binaire est réduite de 50 Mo à 12,5 Mo. Ce qui permettra au nombre de lignes en cache d'atteindre 50 000 petites lignes, mais contenant seulement 12 500 des plus grandes.

Gestion des connexions à la base de données

(Managing Database Connections)

Dans tous les modes d'exécution (serveur ou in-process), les connexions multiples au moteur de la base de données sont prises en charge. Le mode in-process (standalone) prend en charge les connexions dans la même machine virtuelle Java, alors que les modes de serveur supportent des connexions sur le réseau de plusieurs clients différents.

Un logiciel pour gérer le cache des connexions (Connection pool (http://en.wikipedia.org/wiki/Connection_pool)ing software) peut être utilisé pour se connecter à la base de données mais n'est généralement pas nécessaire. Dans d'autres moteurs de base de données, on utilise les caches de connexion pour des raisons qui ne s'appliquent pas à HSQLDB.

- Le cas de permettre d'exécuter de nouvelles requêtes alors qu'une lourde (time-consuming) requête est exécutée en arrière plan n'est pas possible avec la version 1.8.0 de HSQLDB puisqu'il verrouille pendant l'exécution de la première requête puis négocie la suivante quand il en a fini avec la première. Cette possibilité est à l'étude et sera intégrée dans une version future.
- Limiter le nombre maximum de connexions simultanées à une base de données pour des raisons de performance. Ceci peut être pratique avec HSQLDB si votre application est construite d'une manière où chaque petite tâche ouvre et ferme des connexions.
- Contrôle des transactions dans une application multi-tâches. Ce peut être aussi bien utile pour HSQLDB. Par exemple, dans une application web, une transaction impliquant un traitement entre les requêtes ou les actions de l'utilisateur dans les pages web. Une connexion séparée doit être ouverte pour chaque session HTTP pour que la transaction puisse être validée ou annulée, selon les cas. Bien que ce mode de fonctionnement ne puisse être appliqué à la plupart des autres moteurs de base de données, HSQLDB est parfaitement capable de manipuler plus d'une centaine de sessions HTTP simultanées, en tant que connexions individuelles JDBC.

Une application qui n'est pas à la fois multi-tâches et sachant gérer les transactions, telle qu'une application qui enregistre les connexions et déconnexions de l'utilisateur, ne nécessite pas plus d'une connexion. Cette

connexion peut rester ouverte indéfiniment, et est réouverte seulement si des problèmes de réseau l'ont fermée inopinément.

Lors de l'utilisation d'une base de données "in-process" avec des versions antérieures à la 1.7.2 le programme d'application doit garder au moins une connexion ouverte à la base de données, sinon la base de données est fermée et les tentatives ultérieures de créer des connexions échoueront. Ce n'est plus nécessaire depuis la 1.7.2, qui ne ferme pas automatiquement une base de données "in-process" qui est ouverte par l'établissement d'une connexion. Il faut pour fermer la base de données une commande explicite SHUTDOWN, avec ou sans argument. Dans la version 1.8.0 vous pouvez utiliser une propriété de connexion pour revenir à l'ancien comportement

Lors de l'utilisation d'une base de données serveur (et, par extension, d'une base de données "in-process"), portez une attention particulière au fait d'éviter de créer et supprimer des connexions JDBC trop fréquemment. Ne pas observer ceci aboutirait à des échecs de connexion lorsque l'application est lourdement sollicitée.

Mise à jour des bases de données

(Upgrading Databases)

Toute base de données non créée avec la version 1.8.0 doit être mise à niveau vers cette version. Ceci inclue les bases de données créées avec les versions RC (Release Candidate) de la 1.8.0. Les instructions de la section Mise à niveau par la commande SCRIPT doivent être suivies dans tous les cas.

Une fois une base de données mise à jour à la version 1.8.0, elle n'est plus utilisable avec Hypersonic ou des versions précédentes de HSQLDB.

Il peut y avoir des problèmes potentiels d'héritage dans la mise à niveau qui seront résolus en éditant le fichier .script :

- La version 1.8.0 n'accepte pas de noms dupliqués pour les index comme c'était permis avant la 1.7.2.
- La version 1.8.0 n'accepte pas de noms dupliqués pour les colonnes de table comme c'était permis avant la 1.7.0.
- La version 1.8.0 ne créé pas le même type d'index pour les clés externes que les versions antérieures à la 1.7.2.
- La version 1.8.0 n'accepte pas de noms de tables ou de colonnes qui sont également des identifiants SQL sans les encadrer de guillemets (without double quoting).

Mettre à jour par la commande SCRIPT

(Upgrading Using the SCRIPT Command)

Pour mettre à niveau depuis la version 1.7.2 ou 1.7.3 vers la 1.8.0, envoyez

simplement les commandes SET SCRIPTFORMAT TEXT et SHUTDOWN SCRIPT avec l'ancienne version, puis ouvrez avec la nouvelle version du moteur. La mise à jour est alors complète.

Pour mettre à niveau des bases de données plus anciennes (1.7.1 et antérieures) qui ne contiennent pas de tables CACHED, effectuez un simple SHUTDOWN sur l'ancienne version et ouvrez avec la nouvelle. S'il y a une erreur dans le fichier .script, éditez le et essayez encore.

Pour mettre à niveau des bases de données plus anciennes (1.7.1 et antérieures) qui contiennent des tables CACHED, utilisez la procédure de script ci-dessous. Dans toutes les versions de HSQLDB et Hypersonic 1.43, la commande `SCRIPT 'NomDeFichier'` (utilisée comme une requête SQL) vous permet de sauvegarder une version complète de votre base de données, incluant les définitions d'objets et les données, dans un fichier de votre choix. Vous pouvez exporter un fichier script venant de l'ancienne version du moteur de la base de données et l'ouvrir comme une base de données avec la 1.8.0.

Procedure 5.1. Mise à niveau par une procédure de script (Upgrade Using SCRIPT procedure)

1. Ouvrir la base de données originale dans l'ancienne version du gestionnaire de base de données.
2. Envoi de la commande SCRIPT, par exemple SCRIPT 'NouvelleVersion.script' pour créer un fichier de script contenant une copie de la base de données.
3. Avec la version 1.8.0 du gestionnaire de base de données, créez une nouvelle base de données, dans cet exemple 'NouvelleVersion', mais dans un dossier différent.
4. Fermez cette base de données.
5. Copiez le fichier NouvelleVersion.script du point numéro 2 et écrasez le fichier du même nom pour la nouvelle base de données créée en 4.
6. Essayez d'ouvrir la nouvelle base de données en utilisant le gestionnaire de base de données.
7. S'il y a une quelconque inconsistance dans les données, le numéro de la ligne du script est reporté sur la console et le processus d'ouverture annulé. Éditez et corrigez tout problème dans NouvelleVersion.script avant de tenter de l'ouvrir à nouveau. Voyez les paragraphes de la prochaine section (Éditer le fichier .script). Utilisez un éditeur de texte capable de manipuler de très grands fichiers et qui ne coupe (wrap) pas les longues lignes de texte.

Édition manuelle du fichier .script

(Manual Changes to the .script File)

Dans la version 1.8.0 toute la plage des commandes ALTER TABLE est disponible pour changer les noms et structures des données. Toutefois, si une ancienne base de données ne peut être ouverte à cause d'une inconsistance

des données, ou parce que l'utilisation des index ou des noms de colonnes n'est plus compatible avec la 1.8.0, une édition manuelle du fichier SCRIPT doit être effectuée.

Les changements suivants peuvent être appliqués pour autant qu'ils n'affectent pas l'intégrité des données existantes.

- Les noms des tables, colonnes et index peuvent être changés.
- CREATE UNIQUE INDEX ... vers CREATE INDEX ... et vice versa.
  Un index unique peut toujours être converti en index normal. Un index non-unique peut être converti en index unique seulement si les données dans ce champ (cette colonne) sont uniques pour chaque ligne de la table.
- NOT NULL
  Une contrainte non-nulle peut toujours être enlevée. Elle peut être ajoutée seulement si les données de la table pour cette colonne ont toutes des valeurs non-nulles.
- PRIMARY KEY
  Une contrainte de clé primaire peut être supprimée ou ajoutée. Elle ne peut être enlevée si une clé externe fait référence à cette ou ces colonne(s).
- COLUMN TYPES
  Quelques changements de types de colonnes sont possibles. Par exemple une colonne INTEGER peut être changée en BIGINT, ou les colonnes DATE, TIME et TIMESTAMP peuvent être changées en VARCHAR.

Après avoir complété les changements et sauvegardé le fichier *.script modifié, vous pouvez ouvrir la base de données normalement.

Copies de sauvegarde

(Backing Up Databases)

Les données pour chaque base de données consistent en un maximum de cinq fichiers, tous dans le même dossier. Les suffixes des fichiers sont *.properties, *.script, *.data, *.backup et *.log (un fichier avec le suffixe *.lck est utilisé pour contrôler l'accès à la base de données et n'a pas besoin d'être archivé). Ils devraient être archivés ensemble. Les fichiers peuvent être archivés pendant l'exécution du moteur mais dans ce cas veuillez noter qu'une opération CHECKPOINT or SHUTDOWN ne pourront avoir lieu pendant l'archivage. Il est plus efficient d'effectuer la sauvegarde immédiatement après un CHECKPOINT. Le fichier *.data peut être exclu de la sauvegarde. Dans ce cas, lors de la restauration, un fichier factice (dummy) *.data - qui peut être vide, de longueur zéro - est nécessaire. Le moteur décompressera le fichier *.backup pour remplacer ce fichier factice si la sauvegarde est restaurée. Si le fichier *.data n'est pas archivé, le fichier *.properties doit être modifié pour s'assurer qu'il contient modified=yes au lieu de modified=no avant d'effectuer la restauration. Si une copie de sauvegarde suit immédiatement un checkpoint, le fichier *.log peut aussi être exclu, réduisant ainsi les fichiers signifiants aux 3

*.properties, *.script et *.backup. Les méthodes normales d'archivage, comme la compression dans un paquet unique peuvent être appliquées.